



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN
DATA SCIENCE AND COMPUTATION

Ciclo 36

Settore Concorsuale: 09/H1 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Settore Scientifico Disciplinare: ING-INF/05 - SISTEMI DI ELABORAZIONE DELLE
INFORMAZIONI

MACHINE LEARNING METHODOLOGIES FOR SUPPORTING HPC SYSTEMS
OPERATIONS

Presentata da: Martin Molan

Coordinatore Dottorato

Daniele Bonacorsi

Supervisore

Andrea Bartolini

Co-supervisore

Luca Benini

Esame finale anno 2025

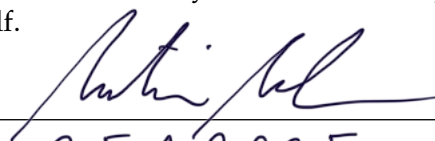
Declaration of Authorship

I, Martin MOLAN, declare that this thesis titled,

Machine Learning methodologies to support HPC systems operations and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:



Date:

25.1.2025

*"But I made myself ready to endure,
and, aided by your words, I made my way
along the rocky slope, behind my guide."*

Inferno, Canto XXIV, Dante Alighieri

ALMA MATER STUDIORUM
UNIVERSITY OF BOLOGNA

Abstract

Faculty of Engineering
Department of Electrical, Electronic, and Information Engineering

PhD in Data Science and Computation

Machine Learning Methodologies for Supporting HPC Systems Operations

by Martin MOLAN

The work presented in this thesis has been driven by two equal and interconnected principles: *performing fundamental machine learning research* and *solving real-life engineering problems*. The specific target domain is high-performance computing (HPC) systems and the specific challenges that come with adapting machine learning methodologies for their monitoring and management. However, models, findings, and methodologies developed for and motivated by this specific set of requirements have applications far beyond the original domain.

The increasing size and complexity of modern HPC systems necessitate introducing advanced data collection, monitoring, and machine learning methodologies that support their management and operations. In literature, this collection of methodologies from data collection to data processing and visualization is called operational data analytics (ODA) for HPC systems. The thesis presents and discusses the comprehensive ODA framework comprising multiple models that address some of the most pressing open problems in the field: *open-ended data exploration, unsupervised anomaly detection, and long-term anomaly prediction*.

The ODA framework first establishes the continuum of the machine learning model adoption in the HPC systems, with each part of the framework addressing a specific stage with its unique requirements and previously unanswered questions. Depending on the level of adoption of operational data analytic methodologies, HPC systems can adopt one, some, or all parts of the framework. The stages of the framework support each other; however, each stage, besides solving its primary objective, enables the adoption of the next one.

The first part of the comprehensive ODA framework is the methodology to perform open-ended *data exploration and analysis*, named the DEM (Data exploration model). DEM is the foundation of the comprehensive ODA framework as it requires no structured or labeled data and can thus be deployed as the first machine-learning model adopted by the HPC system. DEM provides insights into the operation of the nodes and allows the HPC system administrators, or other relevant stakeholders, to identify relevant metrics that require additional analysis by dedicated machine learning models. One such relevant metric is compute node availability, which was studied by anomaly detection models.

The second component of the ODA framework is *RUAD* (Recurrent Unsupervised Anomaly Detection), a novel model that addresses the limitations of current state-of-the-art anomaly detection methods. Unlike traditional approaches that require labeled data or exhibit poor performance in unsupervised settings, *RUAD* leverages temporal dependencies inherent in HPC system data. *RUAD* captures sequential patterns and trends over time by incorporating short-term memory cells into its architecture. *RUAD* was assessed on the complete ten-month history of the Marconi 100 system, encompassing data from all of its compute nodes. It outperformed all previous state-of-the-art semi-supervised and unsupervised approaches. *RUAD* achieves an area under the curve (AUC) of 0.763 in semi-supervised training

and an AUC of 0.767 in unsupervised training, which improves upon the state-of-the-art approach that achieves an AUC of 0.747 in semi-supervised training and an AUC of 0.734 in unsupervised training. It also vastly outperforms the current state-of-the-art unsupervised anomaly detection approach based on clustering, achieving the AUC of 0.548. *RUAD* outperforms all other approaches in an unsupervised setting. This indicates its ability to learn effectively on lower-quality datasets that lack labeled data and include anomalies.

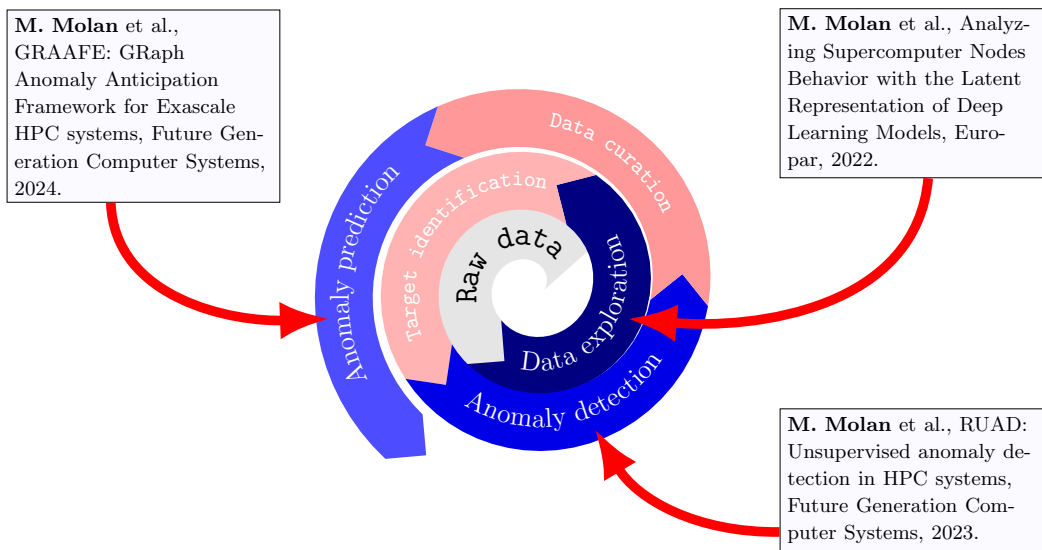
RUAD can be deployed on a dataset that contains no labels. Based on the results of the anomaly identification, if performed on an existing dataset, *RUAD* can, alongside the input from the domain experts, retroactively provide labels to that dataset. Consequently, it enables the introduction of the last part of the ODA framework: a family of long-term anomaly prediction models GRAAFE (GRaph anomaly anticipation framework).

The third component of the ODA framework, GRAAFE extends anomaly detection to anomaly prediction using graph neural networks (GNNs). Information about the physical layout of compute nodes in a compute room is represented as a graph, where compute nodes are vertices, and the graph edges represent the physical distance between compute nodes. Capturing spatial information that per-node predictive models otherwise ignore allows GNN models in GRAAFE to surpass state-of-the-art anomaly prediction methods. The GRAAFE GNN model achieves an area under the curve (AUC) from 0.91 to 0.78, surpassing state-of-the-art, achieving AUC between 0.64 and 0.5. GRAAFE is also the first work to introduce long-term (more than eight hours in advance) node failure predictions in the HPC domain.

The utility of the presented work does not end within the confines of the HPC systems. Applications far outside the HPC domain prove that scientific work driven by *concrete questions from the HPC domain* can achieve *universality and widespread applicability*. Moreover, this work underscores that machine learning models remain highly relevant even in the age of generative AI, serving not only as powerful tools for HPC management but also as versatile methodologies with broader applicability across diverse machine learning challenges.

Keywords: Machine Learning, Operational Data Analytics (ODA), High-Performance Computing Systems (HPC), Anomaly Detection, Anomaly Prediction, Data Exploration, Unsupervised Learning, Predictive Models, Graph Methodologies, Graph Neural Networks, Self-Supervised Learning

Graphical abstract



The comprehensive ODA framework, with the main scientific results, as presented in the PhD thesis, placed on the continuum of machine learning adoption in HPC systems.

Acknowledgements

I am deeply grateful to my supervisor, Professor *Andrea Bartolini*. Thank you for inviting me to pursue a PhD in Bologna five years ago, for teaching me how to write papers, for your patience, and for guiding me through the process of conducting research and mentoring students. Thank you for your ideas and for keeping me grounded when my own ideas went a little too far. Thank you for helping me adjust to life in Italy and for believing in me. And thank you for being a friend.

I am very thankful to my collaborator, advisor, and co-author, Professor *Andrea Borghesi*. Thank you for your help with our papers and for championing and supporting my ideas. I am grateful for the years of wonderful collaboration we shared.

I am also grateful to my co-supervisor, Professor *Luca Benini*, for your guidance, especially in the early stages of my PhD, and for your support and the many opportunities you opened up for me along the way. I am particularly thankful for hosting me at ETH during my period abroad, which was an invaluable experience both personally and professionally.

I also thank Professor *Andrea Acquaviva*, head of the ECS lab, for welcoming me into your lab, involving me in your research activities, and allowing me to work alongside you and your students.

To my collaborators and friends, *Junaid Ahmed Khan* and *Grafika Jati*—thank you for your great ideas and positivity. Junaid, I am grateful for your work on graph methodologies and Grafika for your contributions to automotive perception systems. I am incredibly proud of the work we accomplished together and look forward to seeing where your careers take you.

I am thankful to my friends *Alexis Lope-Bello* and *Terry Curtis*. Thank you both for always encouraging me to continue with my PhD and thank you for all the laughter along the way. Thank you for sparking the fire of my ambition. And thank you for helping me keep its flames in check.

Most of all, I am thankful to my parents, Dr. *Gregor* and Dr. *Marija Molan*, for their immense support and guidance. Because of your unwavering support and example, I have been able to pursue this PhD. You have always been the first to recognize my potential, believe in me, and help me achieve my goals. Every achievement I have made, including this thesis, and every future one, is a testament to the love and support you have always shown me.

Contents

Declaration of Authorship	iii
Abstract	viii
Acknowledgements	xi
Contents	3
List of Figures	7
List of Tables	9
List of Abbreviations	10
1 Introduction	13
1.1 History of High-Performance Computing Systems	17
1.1.1 Top HPC systems trough history	18
1.2 HPC systems in the age of Generative AI	22
1.3 CINECA supercomputing centre	23
1.3.1 Galileo	24
1.3.2 Marconi	25
1.3.3 Leonardo	26
2 Background	27
2.1 Monitoring systems	27
2.2 Machine learning applications	28
3 Data processing framework	31
4 Data exploration – DEM	35
4.1 Introduction to data exploration	35
4.2 Related work	37
4.3 Methodology for data exploration	38
4.3.1 Probabilistic Background	39
4.3.2 General overview of the approach	40
4.3.3 Autoencoder models	41
4.3.4 Feature extraction	42
Singular value decomposition	43
Representative vector	43
Matrix measures	44
4.3.5 Clustering	44
4.3.6 Evaluating clustering	45
4.3.7 Random sampling baseline	46
4.4 Results of data exploration	47

4.4.1	Experimental setting	47
4.4.2	Trained autoencoder	49
4.4.3	Cluster analysis: normal operation percentage	50
4.5	Conclusions of data exploration	52
5	Anomaly detection – RUAD	53
5.1	Introduction to anomaly detection	53
5.1.1	Motivation	56
5.2	Related work	57
5.2.1	Novelty of the developed approach	60
5.3	Methodology for anomaly detection	61
5.3.1	Node anomaly labeling	62
5.3.2	Reconstruction error and result evaluation	63
5.3.3	Trivial baseline: exponential smoothing	66
5.3.4	Unsupervised baseline: clustering	67
5.3.5	Semi-supervised baseline: dense autoencoder	68
5.3.6	Recurrent unsupervised anomaly detection: <i>RUAD</i>	69
5.3.7	Data pre-processing	71
5.3.8	Summary of evaluated methods	73
5.4	Experimental results of anomaly detection	74
5.4.1	Experimental setting	74
5.4.2	Dataset	76
5.4.3	Hyperparameters	76
5.4.4	Area under the curve (AUC)	77
5.4.5	Comparison of all approaches	80
5.4.6	F1 scores	82
5.5	Conclusions of anomaly detection	86
5.5.1	Future work	87
6	Anomaly prediction – GRAAFE	88
6.1	Introduction to anomaly anticipation	89
6.2	Related Works	90
6.2.1	Anomaly Detection & Prediction in HPC	90
6.2.2	GNNs and HPC	91
6.3	Methodology for anomaly anticipation	92
6.4	Results of anomaly anticipation	94
6.4.1	Experimental setting	94
6.4.2	Anomaly Prediction Model Performance	95
6.4.3	Anomaly prediction model probability calibration	98
6.4.4	Visualization of anomaly anticipation	100
6.4.5	Financial impact of anomaly anticipation	101
6.5	Anomaly prediction as part of a GRAAFE framework	106
6.6	Conclusion of anomaly anticipation	107
7	Looking into the Future	108
7.1	Power and Limitations of LLMs	108
7.1.1	Limitations to Perform Symbolic Reasoning	108
7.1.2	Limitation in Ability to Use Tools	110

8	Looking beyond HPC systems	114
8.1	Operational data analytics	114
8.2	Data analysis	115
8.2.1	Methodologies for Data Explorations	115
8.3	Anomaly detection	116
8.4	Anomaly prediction	117
9	Conclusions	119
A	Publications and available resources	123
B	Paper preprints	126

List of Figures

1.1	Statistics on high-performance computer (HPC) systems that present development over time according to application areas [125] such as Research, IT Services, Weather and Climate Research, Chemistry, Energy, Software, Finance, Geophysics, Information Service, Aerospace, Logistic Services, Electronics, Services, Web Services, Information Processing Service, Semiconductor, Automotive, Telecommunication, Database, Defence, and Others.	15
1.2	The <i>CDC 6600</i> : a supercomputer from the 1960s primarily used in scientific research and military applications.	18
1.3	The <i>Frontier</i> : an exascale supercomputer at the Oak Ridge National Laboratory in the United States, featuring HPE Cray architecture with AMD CPUs and GPUs.	19
1.4	Growth of supercomputer compute power [127]. The logarithmic y-axis represents performance in GFLOPS.	20
1.5	Statistics on high-performance computers that present development over time according to segments [126] such as Research, Industry, Academic, Vendor, Government, and Others.	21
1.6	Galileo, the national Tier-1 supercomputer for scientific research. . . .	24
1.7	Marconi 100, the Tier-0 supercomputer based on the Lenovo NeXtScale platform.	25
1.8	Leonardo, the BullSequana-x pre-scale Tier-0 supercomputer, worldwide ranged as the 7. larger supercomputer in 2024.	26
3.1	Developed data driven models.	32
3.2	Developed data driven models as part of the data processing framework.	33
4.1	Data flow schema. On each of the nodes (red in the picture), organized into racks, we train a separate autoencoder model (circles). From these trained models we extract features that are then used in the clustering of nodes.	40
4.2	Architecture of the state-of-the-art model, proposed as AdaHPC [30]. In this work, relevant information is extracted from the latent layer Dense (*,8). Data is collected for the ExaMon monitoring system [15]. .	42
4.3	Example for agglomerative hierarchical clustering of six trained autoencoders from AE_1 to AE_6 : Each autoencoder starts in its own cluster, and pairs of clusters are merged as one moves right the hierarchy. .	45
4.4	Architecture of the autoencoder network, adopted from Borghesi et al. (AdaHPC) [30]	49
4.5	Average error rate per cluster. Representation of nodes with a vector of singular values identifies two clusters with significantly higher anomaly rate than the whole population.	51

4.6	Average error rate per cluster. Matrix-based feature extraction performs worse than the vector methods.	51
5.1	The model M is composed of two parts: Autoencoder in the upper part and MSE in the lower part of the figure.	64
5.2	Structure of baseline model - the dense autoencoder.	69
5.3	Structure of the proposed <i>RUAD</i> model consisting of the LSTM encoder and dense decoder.	69
5.4	Data processing schema. The data flow is represented by green (training set) and orange (test set) lines. The scaler is trained on the training set and applied on test set to avoid contaminating the test set. Semi-supervised and time consistency filters are optional and applied only when required by the modeling approach as indicated in Table 5.4 . . .	71
5.5	To ensure time consistency after removing the anomalous data (A in the Figure), we first split the data into chunks of successive timestamps without anomalies. Then we remove all chunks that are shorter than the input sequence length. Training sequences are only generated on the remaining chunks (green in the Figure).	72
5.6	Combined ROC curve from all 980 nodes of Marconi 100 for the exponential smoothing baseline. Exponential smoothing performs even worse than the dummy classifier - anomaly detection based on exponential smoothing is completely unusable.	77
5.7	Combined ROC curve from all 980 nodes of Marconi 100 for the simple clustering baseline. This baseline performs only marginally better than the dummy classifier.	78
5.8	Combined ROC curve from all 980 nodes of Marconi 100 for the Dense autoencoder model. In the area interesting for practical application - True Positive Rate between 0.6 and 0.9 - semi-supervised approach outperforms unsupervised approach.	79
5.9	Combined results from all 980 nodes of Marconi 100 for <i>RUAD</i> window sizes 5 and 10	83
5.10	Combined results from all 980 nodes of Marconi 100 for <i>RUAD</i> window sizes 20 and 40	84
6.1	Finite state machine depicting the transitions between states 0 (normal operation) and state 1 (anomaly).	92
6.2	The structure of the GCN network exploits the organization of compute nodes in a rack.	93
6.3	Period of 5 timestamps or 75 minutes, starting at 17:00 and ending at 18:15.	95
6.4	Period of 10 timestamps or 2.5 hours.	96
6.5	Period of 20 timestamps or 5 hours.	96
6.6	Period of 30 timestamps or 7.5 hours.	97
6.7	Period of 50 timestamps or 12.5 hours.	98
6.8	Period of 100 timestamps or 25 hours.	99
6.9	Depending on the scenario and the associated probability of preventing the anomaly, given a specific warning window, the projected benefit is achieved by deploying the predictive system at different future windows.	105
6.10	The software architecture of the GRAAFE framework, built for the deployment of an anomaly prediction model for HPC systems.	106

7.1	Structure of the application LLMs	110
7.2	Adoption of LLMs in real world applications: Large language model (RAG) including broader software infrastructure using tools ($T_1, T_2 \dots, T_n$).	111
7.3	LLMs and tools	111
7.4	Future of operation data analytics	113
8.1	Position of <i>Anomaly prediction</i> in the developed data driven model. . .	117

List of Tables

4.1	A list of features used in training of an anomaly detection model. An anomaly detection model is created only on hardware and application monitoring features. More granular information regarding individual jobs is not collected to ensure the privacy of the HPC system users. . . .	48
4.2	Minimum average availability within clusters identified by different feature extraction methods presented in 4.3.4. Vector of singular values identifies a cluster with the lowest average availability (highest anomaly rate). This is the most interesting method as it separates the target variable (node availability) the best. None of the proposed methods identify a cluster with a single node.	50
5.1	Summary of anomaly detection approaches on HPC systems	60
5.2	Comparison between <i>removed from production</i> and <i>node availability</i> . The anomalies studied in this work (node availability) significantly differ (and are more reliable) from anomalies studied in previous works. The new labels also mark much fewer events as anomalous.	62
5.3	Average percentage of removed normal data due to semi-supervised and time consistency filters.	73
5.4	Short names and training strategies for examined methods. $DENSE_{semi}$ is the AdaHPC [30].	74
5.5	Comparison of implemented approaches relating to the training set requirements.	74
5.6	AUC performance of model baselines. According to expectations and existing work [30], semi-supervised dense autoencoder outperforms unsupervised dense autoencoder (highlighted by the higher AUC score).	80
5.7	$RUAD$ and $RUAD_{semi}$ <i>outperform all previous baselines</i> presented in Table 5.6. In contrast to the dense autoencoders, the proposed approach $RUAD$ performs best in <i>unsupervised manner</i>	81
5.8	Combined F1 scores for all compute nodes. F1 scores worse than the trivial classifier (decision threshold 1) are greyed out. $RUAD$ outperforms all previous approaches, including the previous state-of-the-art ($DENSE_{semi}$ and $DENSE_{un}$).	85
6.1	AUC scores of prediction methods. The GNN outperform all other methods across all future windows (FW).	94
6.2	GNN anomaly predictor is well-calibrated for all future windows (FW), as evidenced by the low Brier scores.	100
6.3	Three scenarios are defined based on the probability that the system administrators can prevent an anomaly if given a signal within a future window (in hours).	102

List of Abbreviations

Notation	Description
AdaHPC	Anomaly detection and anticipation in high performance computing systems
AE	Autoencoders
AI	Artificial intelligence
AUC	The area under the curve
CDC6000	Mainframe computers manufactured by Control Data Corporation
CINECA	A public university consortium and the main supercomputing centre in Italy (Consorzio Interuniversitario del Nord-Est per il Calcolo Automatico)
CLU	Ranking anomalous high performance computing sensor data using unsupervised clustering
CPU	Central processing unit
DEM	Data exploration model
DENSE	Type of deep neural network, which can be characterized by different topology
DL	Deep learning
DNN	Deep neural network
DT	Decision tree
EuroHPC-JU	European high-performance computing joint undertaking
ExaMon	Framework for holistic monitoring of a large plant
EXP	Exponential smoothing is implemented to demonstrate that the anomalies we observe are not simply unexpected spikes in the data signal
FLOPS	Floating point operations per second
FPGA	Field programmable gate array
FW	Future window
GBT	Gradient boosting tree

Abbreviations continued on next page

Abbreviations continued from previous page

Notation	Description
GCN	Graph convolutional neural network
GNN	Graph neural networks
GPT	Generative pre-trained transformer
GPU	Graphics processing unit
GRAAFE	Graph anomaly anticipation framework for exascale HPC systems
Grafana	Multi-platform open source analytics and interactive visualization web application
HPC	High performance computers
HW	Computer hardware
LiDAR	Light detection and ranging
LLaMa	Family of autoregressive large language models
LSTM	Long short-term memory
ML	Machine learning
MSE	Mean squared error
NN	Neural network
NVIDIA CUDA	NVIDIA's software platform
ODA	Operational data analytics
PBS	Workload manager and job scheduler
RAG	Retrieval-augmented generation
RF	Random forest
ROC	Receiver operating characteristic
<i>RUAD</i>	Recurrent unsupervised anomaly detection model
SLURM	Simple Linux utility for resource management
SPECFEM3D	Simulator of 3D seismic wave propagation in any region of the Earth
SW	Computer software
Tier-0	The most privileged assets and accounts in an IT environment
TOP500	500 most powerful non-distributed computer systems

*To my mother. My role model, my inspiration, and my rival.
And the greatest scientist I know.*

Chapter 1

Introduction

Each individual has experience growing up and developing from childhood to adulthood. Just as we have an ear and understanding for falling in a small child, we expect an ever-higher level of skill, increasing reliability, and fewer mistakes as we grow up. The expectation we have about technology is similar. At the beginning of the development of automobiles, our predecessors were satisfied with a few miles and errors, stoppages, and problems that arose relatively quickly. With the development of automotive technology, vehicles have become increasingly reliable and overcome increasing distances, breakdown, failure, or unreliability is becoming less and less acceptable.

A similar tendency is also encountered in hardware and software development. The first computers were accepted as a miracle, but it was assumed that only some things worked as the developers imagined. However, development went forward rapidly, and increasing demand conditioned increasingly complex devices.

Computers are becoming larger and more powerful. Simultaneously, ever increasing amounts of data are being collected. Just as, 100 years ago, we marveled at a light bulb connected by a thin wire, early computers with minimal capacity were considered revolutionary and entirely sufficient for their time. However, as development progressed, advancements in information technology gave rise to increasingly complex systems. Similarly, in the early days, small hydroelectric plants on rivers met electricity needs adequately. Yet, with industrial growth and the relentless rise in demand, extensive energy facilities, such as large power plants, became necessary, struggling to keep pace with the ever-growing need for power.

As electricity demand grows, so does the volume of data. Increasingly vast quantities of data are generated, made accessible, and partially stored. However, processing these immense volumes necessitates the development of newer and more powerful computers.

Large computer systems, commonly referred to as supercomputers or high performance computing (HPC) systems, were initially designed to process research data in physics. Accelerators in high-energy physics are growing larger, and the volume of data they generate is increasing exponentially. However, physics is only one of many fields requiring fast and robust systems.

The advent of artificial intelligence (AI) has marked a new era in data processing, offering solutions to complex challenges across diverse areas of human activity and the environment. AI has empowered us to uncover archaeological patterns, explore the biological intricacies of our surroundings, and address critical issues in human health.

Recognizing the importance of processing vast amounts of data has led to the development of computer systems capable of tackling these challenges. HPC systems were initially designed to address problems in basic research, with applications in health research soon following. Today, they are essential for addressing the survival of humanity as a species and improving the quality of life globally.

While computer systems meet the immediate needs of researchers and users upon installation, evolving demands necessitate larger and more capable HPC systems. These systems face significant challenges, including the need for efficiency, continuous operation, and minimized operating costs. As the demand for more powerful and complex systems grows, energy efficiency, sustained operation, and full utilization of resources become increasingly critical. The capabilities of HPC systems have expanded beyond researchers, serving industrial development and technology. They now play a vital role in addressing global challenges and supporting efforts to preserve the planet.

This reliance on HPC systems underscores a significant challenge for humanity, one made more manageable by collecting and processing information. AI tools are invaluable in this learning process but simultaneously drive the need for even more powerful computing systems. AI tools themselves accelerate the demand and growth of HPC systems.

However, humanity's needs cannot be met solely by building ever-larger HPC systems. Effective planning, maintenance, and management are equally important. Optimized software for HPC systems represents a critical avenue for enhancing their functionality and ensuring they meet required performance levels. The challenge of managing and expanding supercomputers lies in meeting the unrelenting demand for processing immense volumes of data.

HPC systems have become indispensable in helping us master the world and drive innovation across various fields. Their evolution is key to tackling the challenges of the present and unlocking opportunities for the future.

An ever-increasing volume of collected information necessitates processing by more sophisticated and powerful devices. In this context, HPC systems represent the definitive solution to these demands.

However, HPC systems remain relatively rare and are reserved for addressing the most complex problems. These systems are limited in number due to their immense size, high cost, significant energy consumption, and rapid obsolescence. Their service life is relatively short, leaving no margin for errors. Consequently, they are expected to operate as reliably and stably as possible throughout their lifespan.

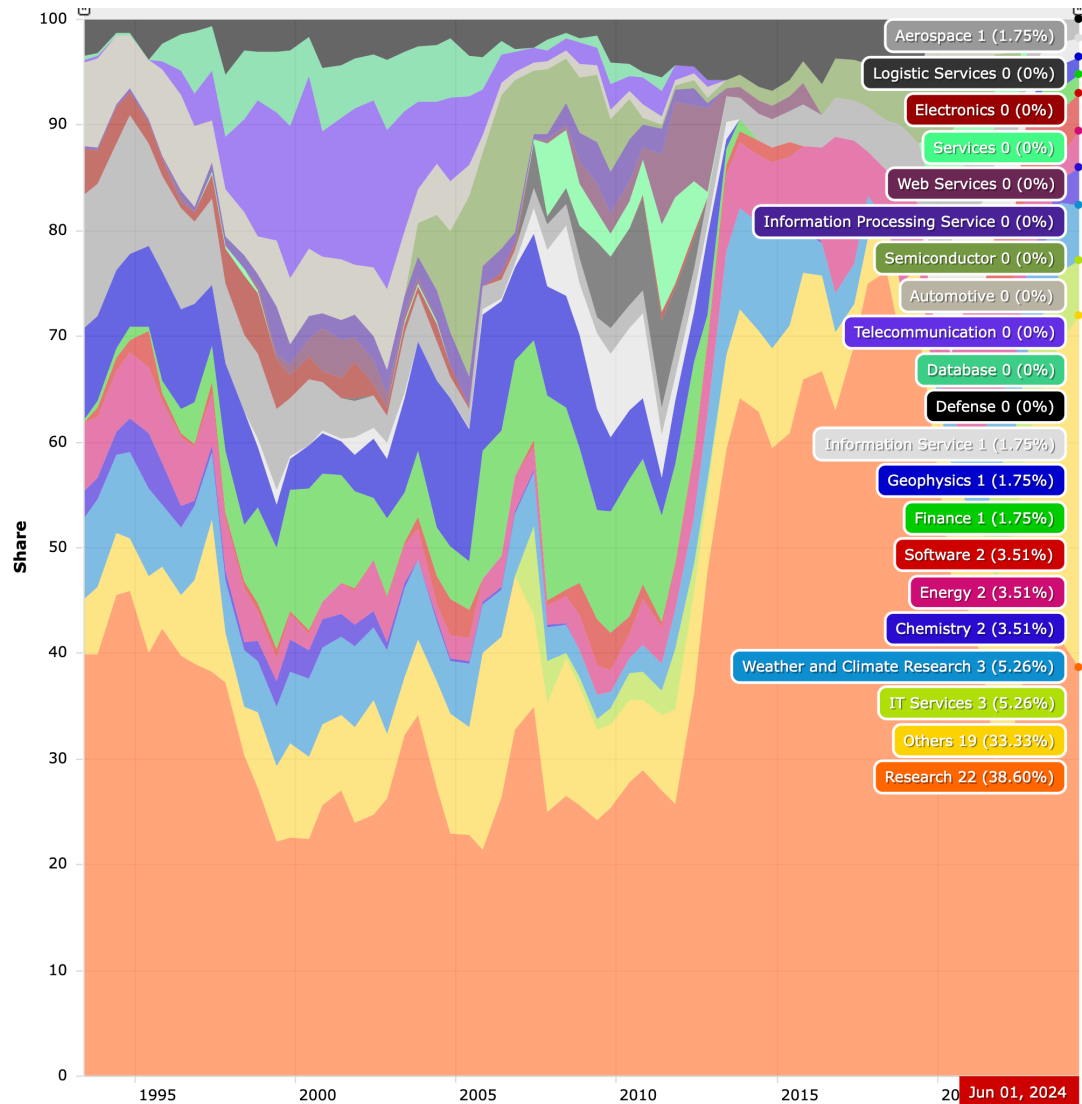


FIGURE 1.1: Statistics on high-performance computer (HPC) systems that present development over time according to application areas [125] such as Research, IT Services, Weather and Climate Research, Chemistry, Energy, Software, Finance, Geophysics, Information Service, Aerospace, Logistic Services, Electronics, Services, Web Services, Information Processing Service, Semiconductor, Automotive, Telecommunication, Database, Defence, and Others.

while maintaining reasonable energy efficiency. These expectations present significant challenges for large-scale HPC systems.

Ensuring the stable availability and operation of HPC systems is a critical responsibility for managers and system administrators. Traditional, manual management approaches are no longer feasible. Instead, advanced control systems that incorporate the latest innovations, including machine learning, are now essential. During their operational lifespan, HPC systems generate vast amounts of data, which must

be thoroughly analyzed to enhance system stability. These analyses also serve as invaluable insights for the design and planning of future systems. Timely error detection, advanced data analysis, and predictive models for system stability are among the most pressing challenges in the world of HPCs.

To address these challenges, modern tools and AI are leveraged extensively to ensure reliable operation, timely responses, and optimal efficiency. Developing support methodologies for data analysis, error detection, and reliability prediction is crucial and forms a central theme of the research presented in this dissertation. Effectively managing and interpreting the vast amounts of data, identifying potential issues, and forecasting the future trajectory of system performance remain complex but essential endeavors.

High-performance computing (HPC) systems stand out for their unparalleled ability to process data and execute complex calculations at extraordinary speeds. These systems are predominantly employed in scientific research, engineering simulations, and other domains requiring immense computational power and data handling capabilities. Their primary characteristics include [9]:

- **Computational Power:** HPC systems possess significantly higher computational power than conventional computers, thanks to their multiple high-speed processors. Currently, petascale systems—HPC systems capable of performing 10^{15} floating-point operations per second—are in operation, and exascale systems—HPC systems capable of performing 10^{18} floating-point operations per second—are becoming a reality. The first exascale system is the "Frontier" supercomputer at the Oak Ridge National Laboratory in the United States. It achieved exascale performance, surpassing one exaflop, and has been the world's leading HPC since its deployment in 2021 and full capability achievement in 2022 [130].
- **Parallel Processing:** HPC systems employ parallel processing, enabling them to perform numerous calculations simultaneously. This is typically achieved through multicore and multiple processors working in tandem. For example, the current-generation pre-exascale HPC system at CINECA Bologna, Leonardo, ranked No. 4 in 11/2022 and No. 7 in 06/2024 on the TOP500 list of the most powerful HPC systems, contains more than 5,000 compute nodes. Each node features multiple processors and accelerators working in parallel to solve complex computational problems [43].
- **Energy Efficiency:** Modern HPC systems are designed to optimize energy efficiency by minimizing power consumption while maximizing performance.

This is achieved through techniques such as using low-power processors, advanced cooling systems, and energy-aware scheduling. Specifically, the Leonardo HPC system is equipped with software tools for dynamic power adjustment. The Bull Energy Optimiser monitors energy and temperature profiles via IPMI and SNMP protocols. These tools can interact with the SLURM scheduler to fine-tune features, such as selecting jobs based on expected power consumption or dynamically capping CPU frequencies to reduce overall consumption. The current generation pre-exascale HPC system Leonardo, hosted at Cineca, consumes just 31 kW per petaflops (Pflops) of computational power [43].

- **Reliability:** HPC systems are constructed with redundant components and fault-tolerant mechanisms to ensure high availability and minimize system failures. Reliability and availability are further enhanced through machine learning tools like anomaly detection and prediction, alongside robust hardware and software design. For instance, on the Marconi 100, a petascale HPC system at CINECA, system downtime accounted for less than 1% of all operational time [44].

1.1 History of High-Performance Computing Systems

The history of High-Performance Computing (HPC) is marked by continuous advancements in technology, leading to increases in size, complexity, computational power and energy efficiency.

- **Early Developments:** The origins of HPC can be traced back to the 1960s with the development of supercomputers like the CDC 6600, designed by Seymour Cray. These early systems were primarily used in scientific research and military applications [45].
- **The Rise of Parallel Processing:** In the 1980s and 1990s, parallel processing became a defining feature of HPC. Supercomputers like the Cray-2 and the Connection Machine introduced new architectures, significantly boosting computational capabilities [133].
- **The Era of Teraflop and Petaflop Computing:** In the early 21st century, HPC systems achieved computational power milestones, reaching teraflop (10^{12} floating-point operations per second) and then petaflop (10^{15} floating-point operations per second) speeds. Notable systems include ASCI Red, IBM's Roadrunner, and China's Tianhe-1A [133].



FIGURE 1.2: The CDC 6600: a supercomputer from the 1960s primarily used in scientific research and military applications.

- **Exascale Computing:** The current frontier in HPC is exascale computing, which aims to build systems capable of performing a quintillion calculations per second (10^{18} floating-point operations per second). This era is marked by significant advancements in hardware, such as GPUs alongside CPUs, and innovations in energy efficiency and cooling technologies [46].

In the 1990s, an initiative was launched to define criteria for monitoring the development and capabilities of HPC systems worldwide. Since 1993, the list of the 500 most powerful HPCs globally has been published biannually, in June and November [53]. The development of HPCs and their applications reflects the global level of technological advancement and showcases the capabilities of specific regions. Investments in HPC power and the focus areas of new systems provide insight into the priorities and strategies of different environments.

Each generation of HPC systems has witnessed increased computational power and complexity, integrating more advanced technologies in processing, networking, and storage. These advancements have significantly expanded the scope and capabilities of computational research and practical applications.

1.1.1 Top HPC systems through history

Since 1993, the global development of High-Performance Computers (HPC) has been meticulously tracked in terms of power, size, and geographical distribution.



FIGURE 1.3: The *Frontier*: an exascale supercomputer at the Oak Ridge National Laboratory in the United States, featuring HPE Cray architecture with AMD CPUs and GPUs.

The preparation of the TOP500 list of the world's largest HPCs serves as a significant indicator of technological advancement. This list, updated annually in June and November, provides a comprehensive overview of the global HPC landscape, particularly the 10 most powerful HPCs within the TOP500.

In 1993, among the ten largest HPCs in the world were 8 computers in the USA, 1 in Canada and 1 in Japan. In 1994, there were 5 largest HPC systems in the US and 5 in Japan. But the situation is changing. In 2004, 6 of the largest HPC systems were in the US, 2 in Japan, 1 in China and 1 in Europe. In 2014, 5 of the largest HPC systems were in the USA, 1 in China, 1 in Japan and 3 in Europe. In 2024, 5 of the largest computers were in the US, 1 in Japan, and 4 in Europe. Of these, 1 each is in Italy, Switzerland, Spain and Germany.

From this data, it is evident that the United States leads in both the number of HPC systems and the total computational power they represent. For several years, the American Frontier has held the position as the world's most powerful HPC. The global distribution of HPC systems mirrors the economic, financial, and technological strengths of individual regions. This distribution provides valuable insights into the priorities, conditions, and specific needs of different environments.

In the initial periods of monitored HPC development, the use in research activities dominated. As much as 70 % of all installed HPC power was dedicated to research activity. However, it is necessary to realize that the installed power was significantly lower. In 2024, however, the relationship is significantly different. Only 38.6% of HPC systems are allocated to research activities, amounting to a total of 22 systems globally. The lowest share of HPC power dedicated to research was

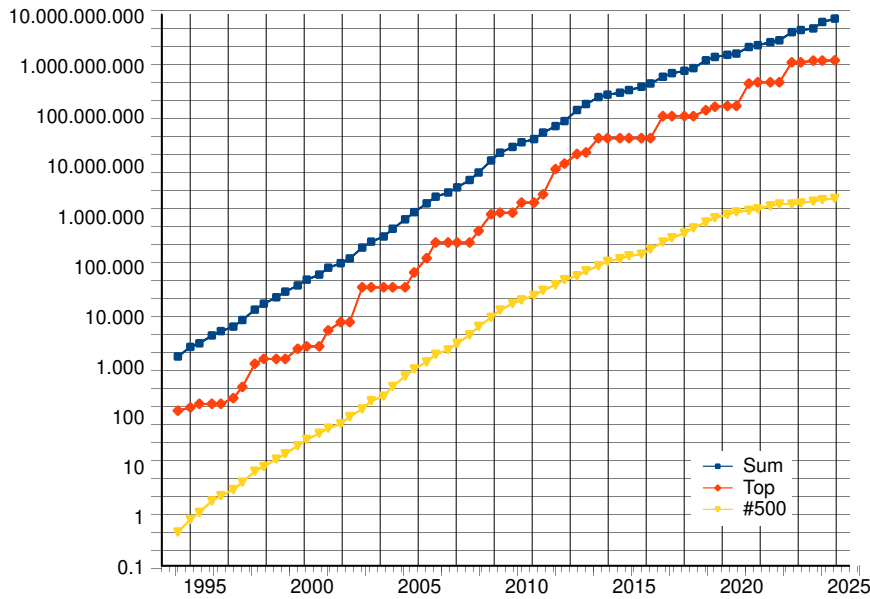


FIGURE 1.4: Growth of supercomputer compute power [127]. The logarithmic y-axis represents performance in GFLOPS.

in November 2021, when it was only 30 %. Some other areas have also begun to take advantage of HPC's power and performance. Thus, in the initial periods of HPC development, HPC was only rarely used for research and activities in the field of water and climate research as it is presented in Figure 1.1. The proportion of HPC computational power dedicated to climate and hydrological research has been steadily increasing, reaching 8.7% in 2021. This is probably the result of the influence of changing conditions, which will represent a severe threat to planet Earth in the future.

The power of HPC was discovered in information technology, mainly with the renewed boom in the use of AI. Thus, the share allocated to the information services field was 5.26 % in June 2024. It is noteworthy that the share of HPC usage across various activities has been steadily increasing. The widespread availability of HPC systems globally has enabled the effective use and development of AI tools.

With the development of AI tools and their use in all areas of human life, in all activities of maintaining a better quality of life for humanity and preserving the planet. HPCs are becoming the environment that enables research and development of solutions or new services, measures that will maintain the quality of life in the future. The use of HPC is becoming more evenly distributed between pure research activities and the development of practical solutions. These two areas cover 1/3 of the HPC capacity. There are still HPC uses in some places that were more important in the past, and services that no longer need HPC power have been developed.

The development of HPC, its global distribution, and the correlation between HPC deployment and the economic and financial strength of individual regions

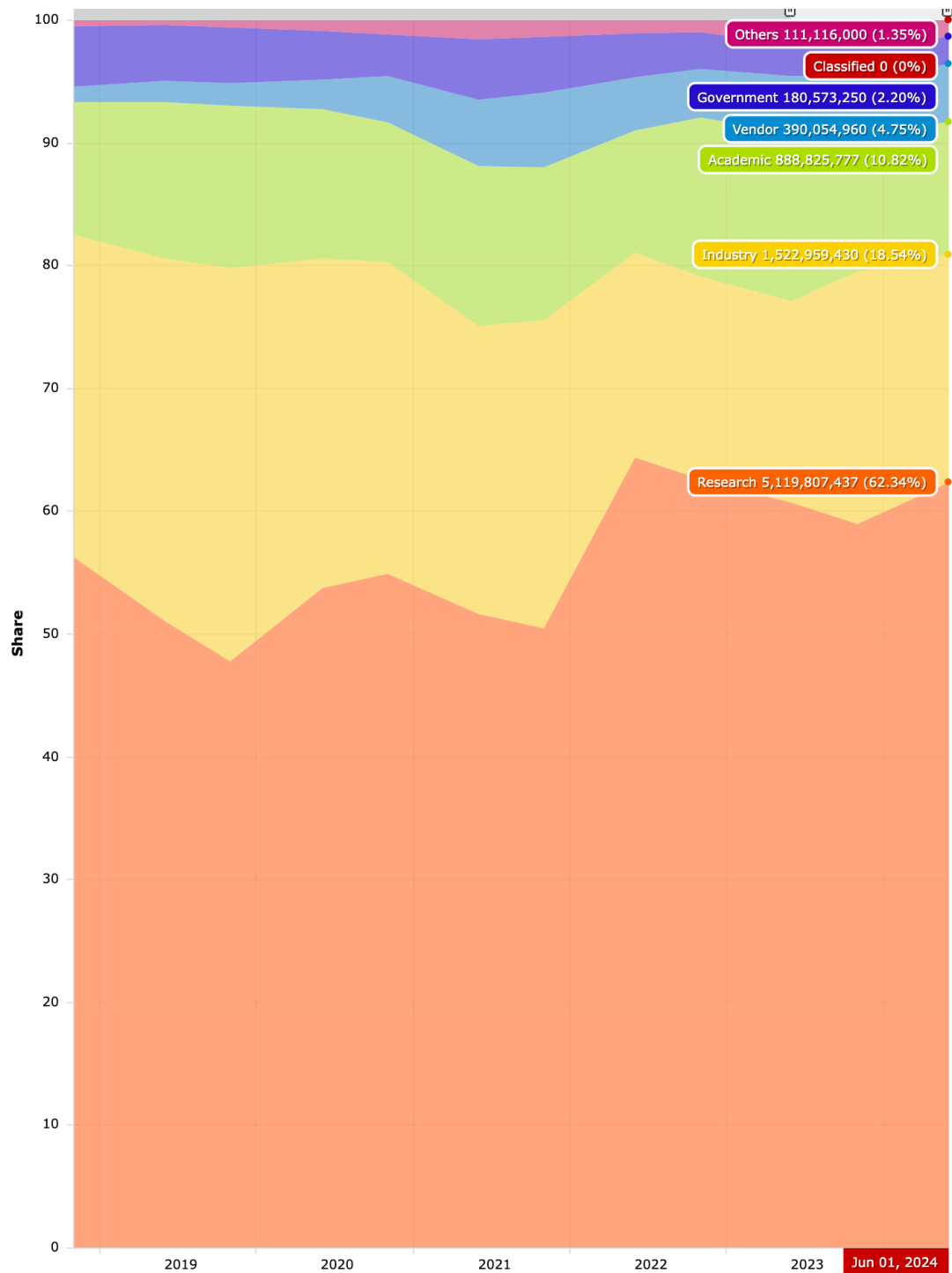


FIGURE 1.5: Statistics on high-performance computers that present development over time according to segments [126] such as Research, Industry, Academic, Vendor, Government, and Others.

highlight the significant dependence of HPC on its surrounding environment. These systems are inherently large and costly, requiring substantial investment over a relatively short operational lifespan. High energy consumption is a characteristic feature of HPC systems; however, their rapid technological evolution also makes them more

prone to obsolescence compared to other systems designed for broader applications.

Short lifetimes, high power and service requirements, the desire for continuous operation and availability present major challenges to the operators of large HPCs. Due to the growth of needs and the ever-increasing number of areas where HPCs are used, not as an exclusive environment, but to enable the development of solutions to various problems, it is becoming an error-free environment. Expectation of stable operation are increasing, and the occurrence of errors, anomalies, and shutdowns is becoming less and less acceptable. The anomaly is expected to be detected in time or even announced when malfunctions occur. These challenges are faced by HPC systems worldwide. The larger the system, the higher the expectations and stakes, yet the operational lifespan of HPC systems remains relatively short, typically only a few years.

In 2024, HPC systems are utilized as follows: 62.34% for research, 18.54% by industry, 10.82% for pure basic research and the academic environment, 4.75% for market-oriented research and services, and 2.2% by state institutions, as shown in Figure 1.5. These users typically fund access to HPC resources, although the specific activities they undertake may vary. These activities are predominantly those that cannot be performed in conventional environments.

1.2 HPC systems in the age of Generative AI

The proliferation of AI systems has increased the importance of High-performance Computing (HPC) systems as the training and hosting infrastructure for the new generations of ever-more-powerful and compute-intensive models. The widespread adoption of compute-intensive AI workloads, particularly generative AI, compels us to reevaluate the role of HPC systems in society. Driven by these advancements, HPC infrastructure is evolving from primarily supporting scientific applications such as computational fluid dynamics, high-energy physics, and computational chemistry to serving as the backbone of a modern AI-driven economy. According to a 2023 report by McKinsey, industry leaders across various sectors have, on average, allocated 20% of their digital budgets to AI-related technologies, with this percentage expected to increase in the coming years [88]. As AI's importance and investment continue to grow, so does the demand for robust hosting infrastructure.

This increased demand for the hosting capabilities for computationally expensive AI models has directly translated into a significantly increased investments into HPC capabilities: Datacenter Dynamics reports that the total investment into HPC capabilities, driven exclusively by the AI demand, accounted for 36 billion USD in 2023 with 2024 projected to surpass this investment by 24.4% [50]. Increased investment and demand intensify the problem of efficient and sustainable operation for the HPC infrastructure. A recent report published in Nature estimates the energy consumption of training a large-scale large language model (such as GPT 4) to be

as high as an annual consumption of 6500 US households [104]. The energy intensiveness of the training of such large models further exemplifies the importance of efficient utilization and management of the HPC infrastructure; a report by MIT suggests that even simple power-management strategies such as capping GPU power usage can reduce the energy consumption of a model training by as much as 12-15% [78].

Another often overlooked aspect of data center sustainability is the opportunity cost of hardware failures. Determined by the rapid hardware development, HPC infrastructure is, on average, across all applications, replaced every 4.2 years, with flagship high-performance systems being replaced even more often [47]. Every downtime or period of HPC sub-utilization thus carries the opportunity cost in terms of investment and the embodied carbon associated with the manufacturing of the system [8].

All these challenges motivate the importance of efficient and holistic management. This discipline is called operational data analytics (ODA), and it is based on high-quality monitoring data, different data processing methodologies, and the data visualization layer. The objective of ODA is to empower the HPC system administrators, users, and other stakeholders to make the most informed decisions about the use and management of the system [106].

1.3 CINECA supercomputing centre

Cineca (*Cineca Consorzio Interuniversitario*) is a non-profit Consortium with 118 members: 2 Ministries, 70 Italian Universities, and 46 Italian National Institutions and agencies. Cineca was established in 1969 as a consortium of publicly held entities on a not-for-profit basis, dedicated to advancing the common good and serving the interests of its consortium members and the national system.

The Cineca High-Performance Computing (HPC) department is the largest computing centre in Italy and one of the largest in Europe. It is the main centre for scientific computing in Italy, with an extensive computing infrastructure available to Italian and European Researchers. Cineca HPC is also available to important Italian companies within a programme for supporting national industrial competitiveness.

Cineca HPC aims to accelerate scientific discovery by providing high-performance computing resources, data management and storage systems and tools, and HPC services and expertise at large. The department seeks to develop and promote technical and scientific services related to high-performance computing for the Italian and European research community [17].

CINECA empowers world-class scientific research by operating and supporting leading-edge supercomputing technologies. With their extensive expertise, the HPC staff offer invaluable support and consultancy in HPC tools and techniques across

various scientific domains, ensuring that researchers feel guided and supported in their endeavours.

In 2024, Cineca holds several prominent roles in the high-performance computing community. It is an NVIDIA CUDA Research Center, recognized for the vision, quality, and impact of its research leveraging GPU technology. Additionally, Cineca serves as an Intel Parallel Computing Center, working to accelerate the development of open-standard, portable, and scalable parallel applications by integrating computational science, hardware, programming tools, compilers, and libraries with domain-specific expertise. Cineca's focus on Quantum Espresso and SPECfem3D underscores its dedication to cutting-edge research and innovation. Moreover, it is one of the Advanced Training Centres, responsible for coordinating training and education activities that empower the European research community to fully utilize the computational infrastructure.

Cineca, one of the large-scale facilities in Europe and a EuroHPC-JU Tier-0 hosting site, is home to Tier-0 systems Leonardo and Marconi and Galileo, the Tier-1 system. All HPC's are running on Linux operating system and has InfiniBand interconnections.

1.3.1 Galileo



FIGURE 1.6: Galileo, the national Tier-1 supercomputer for scientific research.

Galileo is the national Tier-1 system for scientific research. IBM set it up and initially opened and started production in 2015. In June 2015, it was the 105th largest supercomputer in the world. In September 2021, it was upgraded to Galileo100. Its

latest ranking in the top 500 world supercomputers was 397. place in November 2017.

The current hardware of the Galileo consists of 528 computing nodes each 2 x CPU Intel CascadeLake 8260, with 24 cores each, 2.4 GHz, 384GB RAM. It has 50232 computer cores for compute power of 0.68 Pflops. The power consumption of Galileo100 is 2825.55 kW. This means that it needs 4155 kW per Pflops.

1.3.2 Marconi

Marconi is the Tier-0 system, co-designed by CINECA and based on the Lenovo NeXtScale platform. IBM set it up, and the preliminary production started in June 2016 and ended in September 2018.

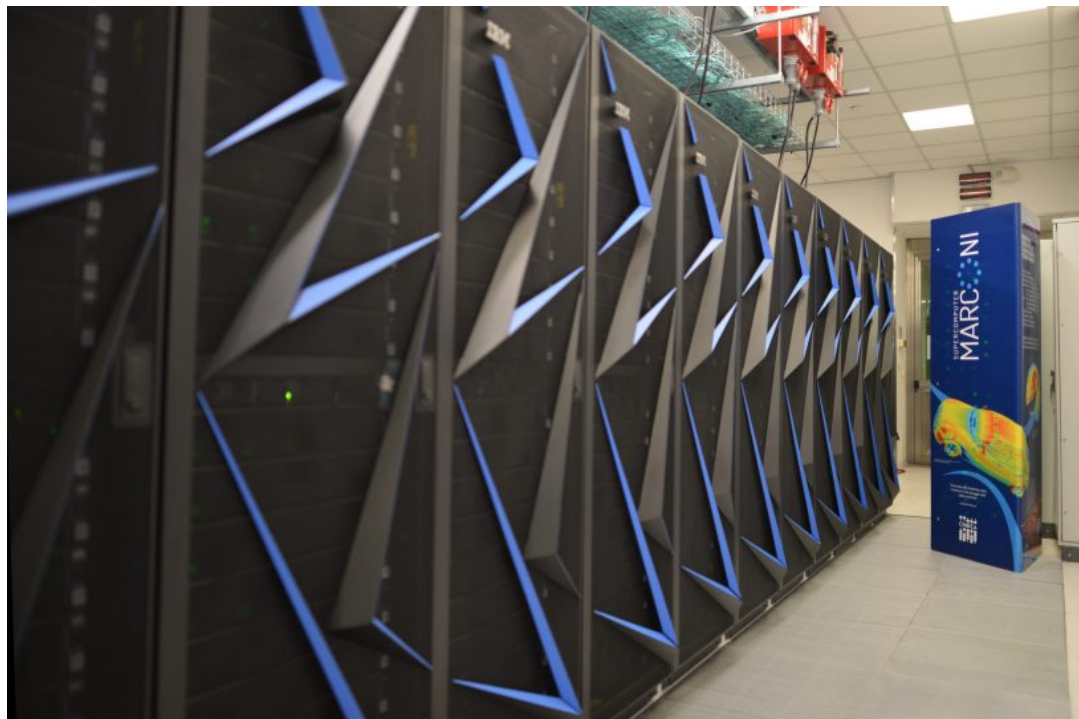


FIGURE 1.7: Marconi 100, the Tier-0 supercomputer based on the Lenovo NeXtScale platform.

The second update was the upgrade to 250000 cores to reach the computational power of 11 Pflops, which lasted until January 2020.

The third upgrade of Marconi started in August 2017 and, in November 2018, was upgraded to total computation power of 20 Pflops. The current configuration consists of 2982 nodes to enable 347776 computer cores for 21.64 Pflops of computer power, while the power consumption is 1476 kW. It has relatively much better efficiency than Galileo: it needs 68 kW per Pflops.

1.3.3 Leonardo

Leonardo is Cineca's largest and newest high-performance computer. As a pre-exascale Tier-0 EURO HPC system, it ranks as the 7th largest supercomputer globally. It was developed and deployed by Eviden, a company within the Atos Group.



FIGURE 1.8: Leonardo, the BullSequana-x pre-scale Tier-0 supercomputer, worldwide ranged as the 7. larger supercomputer in 2024.

Leonardo was announced in June 2019, powered on in October 2022, inaugurated in November 2022, started with pre-production in May 2023, and officially opened with the production phase on August 3rd, 2023.

The current configuration consists of 155 system racks and 4992 computer nodes to enable 1,824,768 computer cores for 241 Pflops. It has 110 PB of storage and takes 600 m2 footprint size. Leonardo's power consumption is 7500 kW. It has much better efficiency than Galileo and Marconi: it needs 31 kW per Pflops.

Chapter 2

Background

The size and complexity of modern HPC systems necessitate the introduction of machine learning methodologies. These methodologies aim to support the work of the system operators in managing the system and to improve its overall efficiency. Machine learning methodologies are integrated within an operational data analytics (ODA) framework. According to Netti et al. 2021 [107], an ODA framework consists of (i) a monitoring system that collects the relevant operational data, (ii) machine learning applications that analyze this data and provide functional insights, and (iii) a combination layer that relates the analysis results to the end users. Only the tight integration of all three operational layers can provide added value to the end users.

2.1 Monitoring systems

The monitoring layer is the foundation of machine learning applications and operational data analytics in HPC systems. The monitoring layer collects data from the component, facility, and node levels. This data is then communicated back to the centralized database for storage and to the application layer for processing. There are two primary possible sources of monitoring data from a supercomputer system: log monitoring data and node telemetry data.

Log monitoring data is present in many systems, and it's an extension of the built-in monitoring of the servers and/or application layers. It is not primarily developed and deployed to provide the dataset for a comprehensive monitoring solution. Depending on the deployment, it can also pose a security risk as it exposes a lot of information about the applications that are running on the systems. This presents an issue in HPC systems, which are inherently multi-tenant and where user privacy is paramount.

For this reason, many HPC systems, including the one studied in this dissertation (Marconi 100 based in CINECA), do not support log collection and log monitoring. In contrast to log monitoring, node telemetry monitoring collects time series information from the hardware and software sensors embedded in the compute nodes of the HPC system, data about system availability, like the availability to accept compute jobs, job scheduler data, like the status of the compute jobs submitted to the

HPC system and the facility data, which is data about other hardware present in the compute room.

Different monitoring data types necessitate using different machine-learning methods. Log data, which is unstructured or semi-structured text data, must first be transformed into informative features or processed by natural language processing techniques [72]. The most basic method for analysis of log data is log parsing, a foundational step in system analysis involving the extraction of structured information from log messages using rule-based or machine learning-driven approaches. This includes retrieving timestamps and event details for further examination and analysis. Named Entity Recognition [72] identifies and categorizes entities like IP addresses and hostnames, facilitating contextual understanding. Sentiment analysis gauges the emotional tone of logs, aiding in issue identification. Anomaly detection employs statistical models or machine learning to identify abnormal patterns indicative of security threats or system anomalies. Topic modeling categorizes log messages into themes, revealing patterns, and recurring events. Using deep learning models, contextual analysis captures nuanced meanings in log text [140]. Sequence-to-sequence models assess temporal dependencies in log sequences, predicting system behavior. Dependency parsing analyzes grammatical structures for key event extraction, and word embeddings represent log message words as dense vectors for semantic relationships. Lastly, query expansion improves log search queries by incorporating synonyms or related terms. Employing this array of natural language processing (NLP) techniques enables comprehensive analysis, monitoring, and optimization of data center and HPC system log data.

In contrast to log data, telemetry monitoring collects time series data that the data analysis layer can directly use. Depending on the target application, different machine learning methods used for time series data can be directly applied to the (processed) time series data. Since (node) telemetry monitoring is employed at CINECA, this work will only study the methods for processing time series node telemetry data. The most extensive collection of node telemetry data relating to HPC operation has been made available by the researchers at the University of Bologna in 2023 [31].

2.2 Machine learning applications

Machine learning methodologies have become integral to HPC systems. The main uses include:

1. **Optimizing System Performance:** Machine learning algorithms can predict and manage workloads on HPC systems, optimizing the usage of resources like CPU, GPU, and memory for higher computational efficiency and energy savings [32].

2. **Fault Detection and Prediction:** These systems are prone to failures. Machine learning models can predict and detect system faults or failures before they occur, reducing downtime and maintenance costs [98].
3. **Enhancing Data Management:** In HPC environments, managing large amounts of data efficiently is crucial. Machine learning assists in intelligent data placement and retrieval strategies [107].
4. **Automating System Administration:** Tasks such as system monitoring, load balancing, and tuning can be automated using machine learning, reducing the need for human intervention [121].
5. **Improving Simulation and Modeling:** Machine learning models can enhance the accuracy and efficiency of simulations run on HPC systems [85].
6. **Enhancing Cybersecurity:** Machine learning can help detect and respond to security threats more rapidly and effectively in HPC systems [149].
7. **Assisting in Workflow Management:** Machine learning can optimize the overall workflow of HPC systems, leading to better utilization of resources [148].
8. **Real-time Analytics and Decision Making:** Machine learning algorithms can process and analyze data much faster than traditional methods for real-time processing [25].

Machine learning methodologies can significantly optimize HPC system performance, particularly when integrated with HPC scheduling. The following are key areas where machine learning supports optimization [137]:

- Machine learning models can accurately **predict resource requirements** for various jobs, allowing for efficient resource allocation. This predictive resource allocation is crucial for optimizing system performance.
- By analyzing past workloads, machine learning can **forecast** future demands on the HPC system. This workload forecasting helps optimize job queues and improve overall system utilization.
- Machine learning algorithms can monitor the system for unusual patterns indicating potential problems. Anomaly detection like this enables **preventive maintenance** and can avoid system downtime.
- Dynamic scheduling is another area where machine learning proves beneficial. The technology can dynamically adjust scheduling policies in real-time, **prioritizing urgent or high-priority tasks** without significantly disrupting other processes.

- In terms of energy efficiency, machine learning models can schedule tasks to **minimize energy consumption**. This is achieved by consolidating jobs to reduce idle times or scheduling energy-intensive tasks during off-peak hours [32].
- Failure prediction is another critical area. Machine learning can analyze system logs and performance metrics to **predict hardware and software failures**, thus avoiding allocating jobs to potentially unreliable resources.
- Accurately **predicting job runtimes** can be challenging. Machine learning models trained on historical job performance data can provide more accurate runtime estimates, which the scheduler can use to optimize job placement and system throughput [42].
- Balancing load across nodes is essential for **preventing the overuse of certain nodes** while others remain underutilized. Machine learning helps understand patterns using different nodes and evenly distributes the workload [151].
- For distributed computing tasks, communication between nodes can be a bottleneck. Machine learning predictive models **optimize the placement of tasks** to minimize communication overhead and improve overall performance [138].
- Finally, machine learning models can **learn the effectiveness of different scheduling decisions** over time. This adaptive learning for scheduling policies leads to refining scheduling policies to improve efficiency and adapt to changing workloads [137].

Chapter 3

Data processing framework

In order to address the open questions in the operational data analytics (ODA) field, this work introduces new technologies in the analytics/data processing layer. They are intended to work in concert with each other and cover different aspects of the ODA tasks. Inspired by the characterization by Netti et al. [106], all tasks focus on various aspects of HPC system availability.

Different algorithmic approaches described in this work can be used as independent modules or integrated into any ODA framework between the data acquisition and communication layers, or they can be used together as a comprehensive model toolkit. Since the models described in this work are intended to be used in the HPC systems of CINECA, they are integrated with ExaMon [26] data acquisition and monitoring layer and the ExaMon visualization layer based on Grafana [79]. The modeling approaches, however, are general and can be used with any data acquisition and visualization technology stack.

The most foundational model, described in Chapter 4, is the **model for data exploration (DEM)**. This model aims to provide general insights into the operation of the HPC system by comparing the behavior of different compute nodes. Unlike the applications of other models in this work, data exploration is not fixed to one specific modeling task, like system availability, but tries to provide general insights into the system's functioning. Since it does not rely on observing a particular label or modeling tasks, it can be used even in systems where only rudimentary node telemetry data is collected. It can be used as a first step in deploying data-driven ODA applications, as it doesn't require system administrators to specify the target modeling tasks explicitly.

When used alone, it aims to give the system administrators a model that can identify clusters of *interesting* or potentially problematic compute nodes. When used as a part of a larger model zoo, such as the one proposed in this work, the data exploration model provides a general overview of the system, which helps to identify variables or features that require special attention and specific models. In work done on HPC systems of CINECA, data exploration uncovered clusters of compute nodes with significantly lower average availability, which, alongside the literature and the feedback of system administrators, motivated future work focusing on detecting and

predicting node failures.

When deploying a data-driven model, the first focus should be increasing the availability of the HPC system [106]. Enhancing a system's availability boosts its scientific throughput and, by improving the amortization of its embodied carbon cost, also contributes to greater environmental sustainability. The first part of introducing data-driven approaches for improving system availability is the introduction of **anomaly detection models**, as described in Chapter 5. Anomaly detection models detect periods of operation when compute nodes are unavailable to accept compute jobs (anomalies). As described in this thesis, the proposed anomaly detection model can, as documented for the first time in literature [98], work even if the labels for the periods of anomalous operation are unavailable. This allows the system administrators to deploy data-driven anomaly detection even on systems that do not collect accurate labels about periods of compute node unavailability. The data collected by the anomaly detection system and then curated by the system administrators can be the basis for the transition to anomaly prediction.

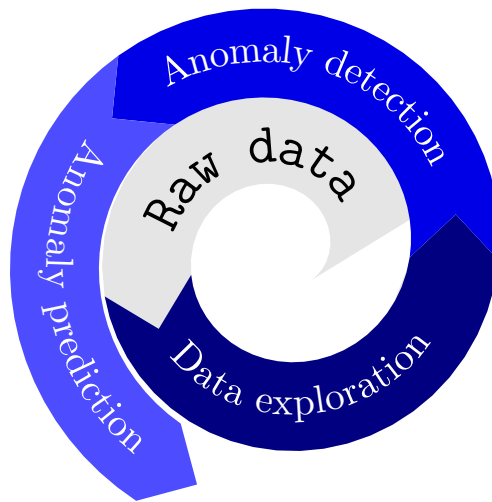


FIGURE 3.1: Developed data driven models.

The transition from anomaly detection to **anomaly prediction** is the primary focus of the final substantive chapter, Chapter 6, of this thesis. While anomaly prediction, as opposed to anomaly detection, offers significant practical advantages if deployed in a production HPC system, it also carries substantial difficulties in implementation. The most important obstacle to introducing anomaly prediction is the need for a high-quality dataset containing accurate anomalous behavior labels. As the anomaly prediction models are supervised as opposed to self-supervised anomaly detection models, the performance of the anomaly prediction model directly relates to the quality of the anomaly labels. Another necessity is additional information about the compute node structure in a compute room, which is then encoded as a graph. The anomaly prediction model cannot be built if this information

is not present.

Due to the differences in the data required for the anomaly prediction and anomaly detection models, anomaly prediction should be considered as something other than an upgrade in capability. Instead, it is an additional capability that comes at the price of higher data quality. Anomaly detection, in turn, can be deployed even when the quality of collected data is significantly lower and labels are unavailable. If deployed with the feedback of the systems administrators, anomaly detection can even help collect the necessary data (periods of failures or anomalies) for the eventual deployment of anomaly prediction.

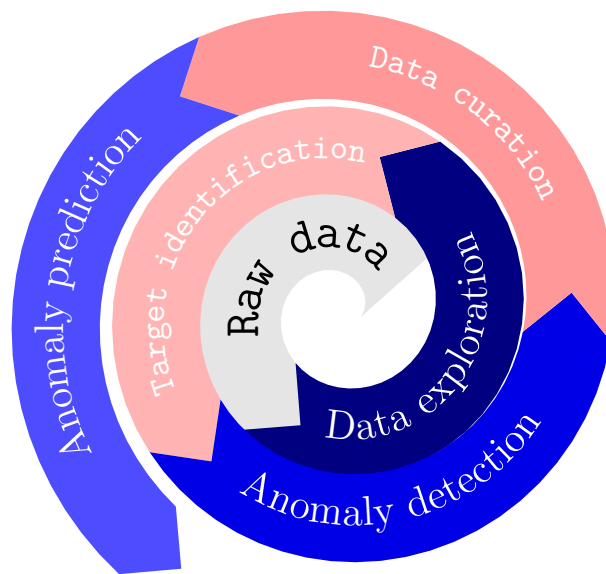
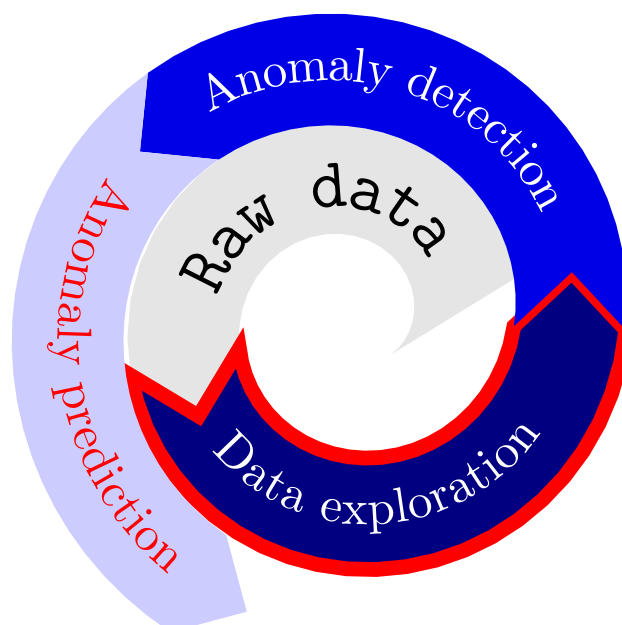


FIGURE 3.2: Developed data driven models as part of the data processing framework.

Chapter 4



Data exploration – DEM

The work presented in this chapter is based on the paper Analysing Supercomputer Nodes Behaviour with the Latent Representation of Deep Learning Models by Molan et al., presented at the Europar 2022 conference [94].

4.1 Introduction to data exploration

High Performance Computing systems have been steadily rising in size and complexity in the last years, as revealed by the exponential increase of the worldwide supercomputer installation [131]. HPC systems are typically composed by replicating a large number of components, usually, in the order of thousands of computing nodes, each of them constituted of a collection of smaller functional parts, such as CPUs, RAM, interconnections, storage, etc. Even if similar by design, each computing node is affected by manufacturing variability and variations in the operating conditions.

The sheer size and complexity of supercomputers create huge challenges in terms of optimal management of the compute components and their significant energy

footprint [60]. The race towards HPC systems capable of performing more than 10^{18} operations per second or more than one Exaflop [77], continues to make these challenges ever more pressing [128].

Overall, it is a daunting task for system administrators and facility managers to optimize supercomputer performance and power consumption, identify anomalous behaviors faulty situations, and guarantee systems operate in optimal conditions. The scale of the problem motivates the development of automated procedures for anomaly detection and faulty node identification in current supercomputers and this need will become even more pressing for future Exascale systems [150].

The fact that most of today's HPC computing systems are endowed with monitoring infrastructures [124] that gather data from software (SW) and hardware (HW) components can be of great help toward the development of data-driven automated approaches. Historically, system management was performed through hand-crafted scripts and direct intervention of system administrators; most of the data is stored in log files, and anomalies are investigated a posteriori to find the source of reported problems (e.g., when many users recognize the failure and report it to administrators).

At the finer granularity, each core of the processing element is equipped with performance counters which can monitor several micro-architectural events (i.e., cache misses, stalls, throughput) and physical means (i.e., temperature, power consumption, and clock frequency). Processing units as well as the motherboard, the power distribution units, the onboard voltage regulators, the PCIe devices, and the fans are equipped with hardware (HW) sensors and counters.

Similarly, software components can provide useful information as well, ranging from the details about jobs submitted by users (e.g., information gathered by job dispatchers such as SLURM [71] or PBS [143]) to software tools performing health-check of various subsystems [48] and I/O monitoring [147].

As the amount of data is overwhelming for human operators, automated processes could be highly beneficial in improving the data center usage to ease the burden of human operators and lower the response time to failures. In this context, AI can provide significant benefits, as it allows to exploit the available big data effectively and to create decision support tools for HPC system administrators and facility managers [14, 144].

In the past, many works from the literature and the practice demonstrated the possibility to extract useful information using data collected from HPC computing nodes and employing supervised Deep Learning (DL) models [135, 108, 33] and semi-supervised ones [24, 28, 30]. These methods have been applied to detect nodes' availability, defined as operation without anomalies.

Availability rate is the percentage of time the compute node is available to accept compute jobs. Availability and the corresponding error rate, which is calculated as

$$1 - \text{availability rate},$$

is a key metric of the node's performance, and a target for optimization of the HPC system operation [107]. Due to its importance, we focus on the availability rate in the experimental part of this work.

Borghesi et al. in [28] show that semi-supervised anomaly detection models trained on individual nodes data outperform a single model trained on multi-node data. This suggests that the semi-supervised model can learn differences between nodes even if the nodes share the same design and composition.

Theoretically, the learned model encapsulates the node's characteristics, however to the best of our knowledge, no one has ever evaluated the feasibility of using the disparities between trained DL models to evaluate the differences between the behavior of the corresponding nodes.

In this work, we answer this question by introducing a novel approach that focuses on the latent representation of the trained DL models, in particular on the coefficients, the weights of the latent layer). The approach can identify clusters that deviate from the overall node population's average availability, relying on the DL model parameters.

We focused on a Tier-0 supercomputer composed of 985 nodes for which we trained a series of per-node semi-supervised DL models based on autoencoders (AE), as proposed as Anomaly Detection and Anticipation in High Performance Computing Systems - AdaHPC [30], the state-of-the-art for semi-supervised anomaly and fault detection in HPC systems. We focus on semi-supervised methods as the availability of labels cannot be taken for granted in a supercomputer due to the non-negligible cost of annotating the vast wealth of monitored data. We explored different approaches to extract features from the weights and biases of the latent layer of the AE model.

A systematic approach to data analysis is presented in *Data Exploration Model (DEM)*, which covers all the necessary principles for effective data exploration.

4.2 Related work

Since anomalies in HPC systems are rare events, the problem of anomaly detection cannot be treated as a classical supervised learning problem [24, 4]. The majority of works that treat it in a fully supervised fashion have been tested using synthetic [135, 2] or injected anomalies [108]. Instead of learning the properties of both relevant classes, the standard approach is to learn just the properties of the system's normal

operation - anything deviating from this normal operation is then recognized as an anomaly.

Machine learning models are trained only on normal data to learn the characteristics of the normal operation. This training of ML models on normal data is called *semi-supervised* training [28].

The state-of-the-art for anomaly detection on the HPC system is to train a particular class of neural networks – called autoencoders – in a semi-supervised way [30]. Autoencoders are a specific type of neural networks that are trained to reproduce an input signal while simultaneously learning the most efficient latent representation of the data [12].

The latent representation of the data has a lower dimension than the original data; this lower dimension of the latent layer naturally leads to the idea of using autoencoders as pre-processing step before applying clustering techniques [122, 82, 139], as most of the clustering algorithms have worse performance in high-dimensional spaces [6]. The autoencoders are first trained on the whole dataset when using autoencoders as a dimension-reduction step before clustering. Then the dataset is projected by the encoder part of the network into a lower-dimensional latent layer [122].

Current approaches that combine clustering and autoencoder neural networks use a single trained autoencoder to encode each instance into a latent space. The state-of-the-art for HPC anomaly detection, however, is to train multiple models - a different model for each node in the system [30]. The fact that the models trained on individual nodes outperform the model trained on combined data of all nodes [30, 24, 28] suggests that there are significant differences between the behavior of the compute nodes and, consequently, the corresponding trained models.

This work explores the possibility of leveraging the fact that we are training multiple AE models to explore the relationship between the nodes. Specifically, we explore the possibility to extract features from the trained neural networks to perform the clustering of the *whole operation history* of the compute nodes.

4.3 Methodology for data exploration

In this section we present the architecture of the proposed approach as *Data Exploration Model (DEM)*. We start by providing the probabilistic perspective underlying the foundations of our approach in Section 4.3.1. We then describe in more detail the general architecture (Section 4.3.2) and the proved the more detailed description of the method in Section 4.3.3 and Section 4.3.4.

4.3.1 Probabilistic Background

The idea of extracting information and comparing trained neural networks extends the standard methodology of statistical modeling, where two or more populations, or generally a collection of instances, are compared by contrasting parameters of fitted distributions.

Comparing the parameters of fitted functions is the key idea underlying the proposed approach. Let us consider as an example the common statistical problem of comparing two populations of individuals - specifically, we want to compare a specific random variable X in two distinct populations (e.g., height in two different countries).

The first point of comparison in such cases is to calculate an empirical mean depicted in equation (4.1)

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i \quad (4.1)$$

and an empirical variance depicted in equation (4.2).

$$\frac{1}{n-1} \sum_{i=1}^N (x_i - \bar{x})^2 \quad (4.2)$$

Two populations can be compared by looking at the empirical mean and variance of the random variables of interest. These are observed variables present inside each population.

- The mathematical foundation of comparing mean and variance between two populations is directly in line with the idea of this work.
- If we are observing two large populations we know from the central limit theorem [51] that the sum of the variables will tend towards a Gaussian distribution.
- Two parameters determine Gaussian distribution: expected value μ and variance σ^2 [51]; to fit the Gaussian distribution to the data (population), we thus have to estimate these two parameters.
- If we fit the distribution via the Maximum Likelihood Estimation (MLE) method, [38], we see that the best estimator for the expected value is *empirical mean* and for variance, the best estimator is *empirical variance*.
- From the probability theory, we know that the difference of two random Gaussian variables is Gaussian variable with mean that is the difference of means and variance that is the sum of variances [51].
- Comparing population mean and population variance is thus actually equivalent to comparing the *Gaussian distributions* fitted to the data.

Another perspective from which to examine the problem of comparing populations is that we fit a *function* to the data, such as the Gaussian distribution. For some problems, like HPC system monitoring, autoencoders, a type of neural network, achieve state-of-the-art results [30, 29]. As autoencoders are the class of functions that best describe this specific class of problems, namely the behavior of compute nodes in an HPC system, we examine whether we can compare the *compute nodes* by comparing the parameters of the fitted autoencoders.

4.3.2 General overview of the approach

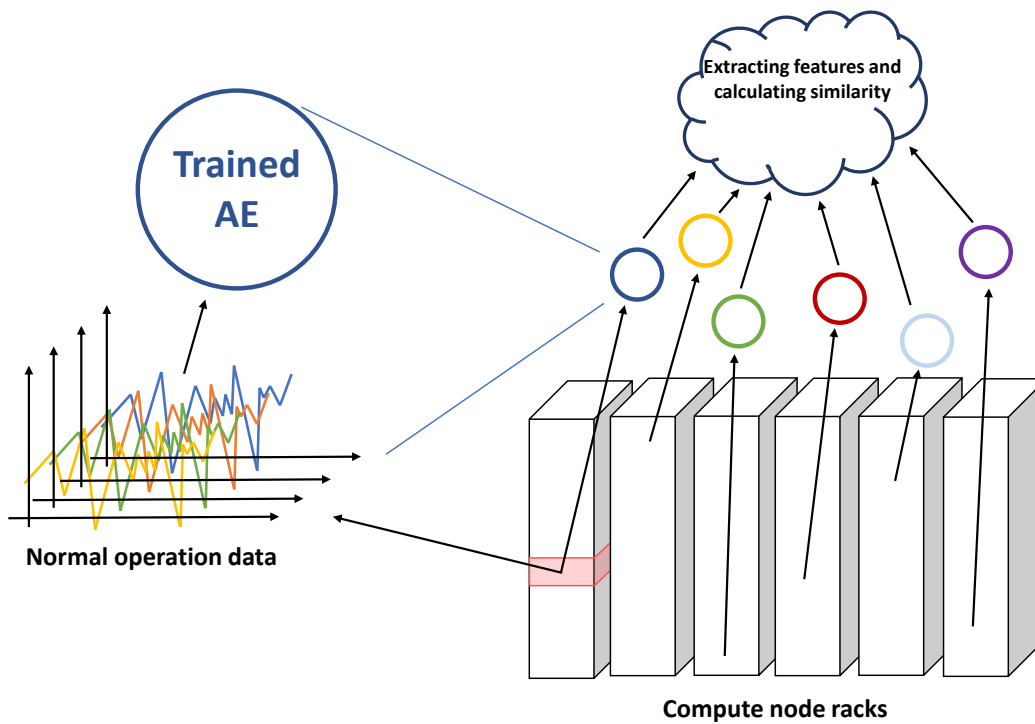


FIGURE 4.1: Data flow schema. On each of the nodes (red in the picture), organized into racks, we train a separate autoencoder model (circles). From these trained models we extract features that are then used in the clustering of nodes.

The key idea is to apply a geometric transformation to the weight matrix underlying the latent layer of the trained AEs; we opted to explore a variety of transformations; namely, we compute:

1. the vector of singular values,
2. the singular vector corresponding to the largest singular value,
3. the map of the representative vector (with and without bias),
4. the weights matrix similarity in L1, L2, and absolute L2 norm,
5. the affine (augmented matrix) similarity in L1, L2, and absolute L2 norm.

The empirical evaluation demonstrates that the vector of singular values identifies interesting clusters among the different methods to extract salient features from the latent representation.

Figure 4.1 reports the block diagram of the proposed methodology. We can identify the following steps:

1. On each node, a separate autoencoder model is trained. Semi-supervised training of per-node autoencoder models is adopted from the work of Borghesi et al. [30].
2. After models are trained on each node, features are extracted (as described in Section 4.3.4) from the deep learning models.
3. Based on these extracted features, the similarity between nodes is calculated. Calculation of similarity can be done as the autoencoder projects the input features into a *latent* representation where only the most salient correlations between the input variables are preserved. The similarity measure is calculated by comparing the representation maps - specifically, the parameters of the latent layer.
4. This similarity measure is then used in hierarchical clustering.

4.3.3 Autoencoder models

Dense autoencoders are a type of deep neural network, which can be characterized by different topology; those used in this work have a distinct hourglass shape, a choice motivated by the results obtained by previous works in the state-of-the-art. These contractive autoencoders use a latent layer significantly smaller than the input and output layers. This shape forces the autoencoder to learn efficient representation (encoding) of the data, driven by the small size of the latent layer.

The most relevant information of the network is encoded in the latent layer. In this particular type of autoencoder, the latent layer is the layer in the middle of the network and contains the fewest neurons. It is preceded by the encoder and succeeded by the decoder, each composed of one or multiple layers.

The encoder and decoder layers used in this work have a symmetrical architecture, which, generally speaking, is not strictly required. The fundamental role of the network is to efficiently encode the information from the input in a compressed representation in the latent layer.

Training of the autoencoder is driven by the reproduction error produced by the decoder; reproduction error, which is the difference between the real input and the reconstructed signal, is minimized during training. The architecture of the network used in this work is presented in Figure 4.2. It is adapted from AdaHPC presented by Borghesi et al. [30] where it has been shown to produce state-of-the-art results in detecting anomalies on an HPC system.

The set of autoencoders - as in original work [30] - are individually trained on each node in a semi-supervised setting. Semi-supervised training means that the data for training is filtered of all anomalies and that only the normal instances are used in training the model.

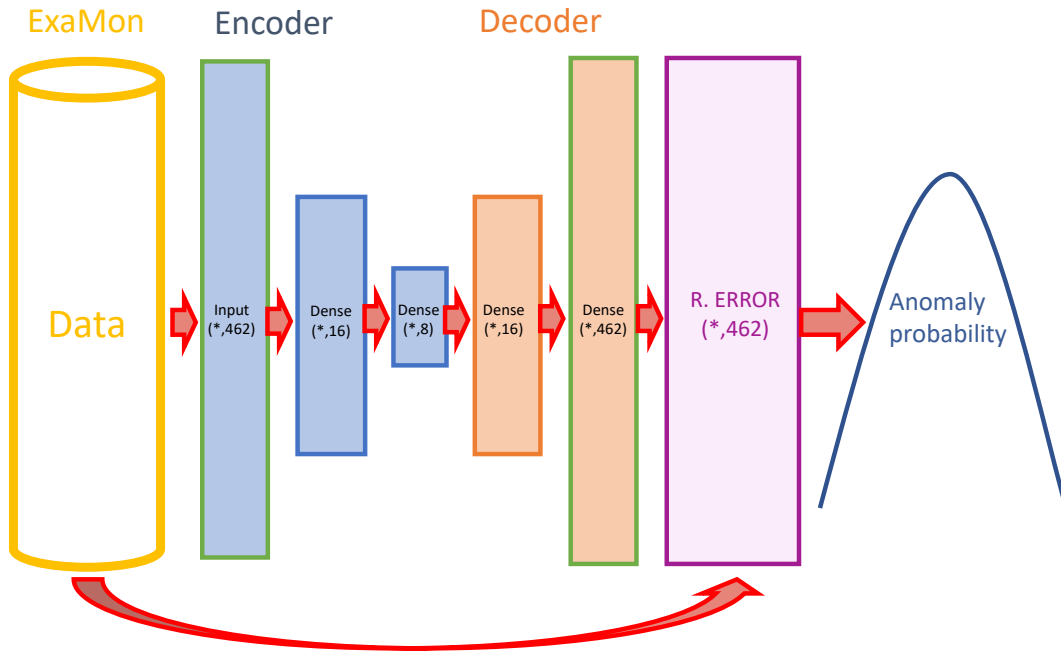


FIGURE 4.2: Architecture of the state-of-the-art model, proposed as AdaHPC [30]. In this work, relevant information is extracted from the latent layer Dense (*,8). Data is collected for the ExaMon monitoring system [15].

4.3.4 Feature extraction

From now on, we will use the following symbols in this section.:

W ... weight matrix,

\vec{a} ... activation vector,

\vec{b} ... bias vector,

f ... nonlinear activation function.

Due to the architecture of the neural network used in this work, as discussed in Section 4.3.3, we extract the relevant features from each node, each with its own dataset; these features are embedded in the weights of the latent layer.

The latent layer is described by the weights matrix W and the bias vector \vec{b} . The activation of the latent layer is given by \vec{a}' in equation (4.3)

$$\vec{a}' = f(W\vec{a} + \vec{b}) \quad (4.3)$$

where f is a nonlinear activation function. In the next subsections, we will describe different encoding approaches of the latent layer information, which will then be used to extract features.

Singular value decomposition

Singular value decomposition represents matrix A as $A = U\Sigma V^T$ where Σ is a diagonal matrix containing singular values [136]. In this work, we used singular value decomposition on W , and we extracted the *vector* of singular values (abbreviated to singular values in the future) and a singular *vector* corresponding to the largest singular value (abbreviated singular vector).

Given any $m \times n$ matrix A , the singular value decomposition of A is an expression of the form:

$$A = U\Sigma V^T$$

Where:

U ... orthogonal matrix with dimension $m \times m$,

Σ ... diagonal matrix with dimension $m \times n$ and non-negative real numbers on the diagonal (these are known as the singular values of A),

V^T ... transpose of an $n \times n$ orthogonal matrix V .

The columns of U and V are known as the left-singular vectors and right-singular vectors of A , respectively.

Representative vector

Let denote the vector $\vec{1}$ of ones as

$$\vec{1} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

We use the vector of ones $\vec{1}$ as a representative vector as it corresponds to the activation of all neurons in a latent layer. It can serve as a proxy for the transformation of the linear part of the latent layer. For each node, we have thus calculated the product of $W\vec{1}$ (abbreviated vector of ones) and $W\vec{1} + \vec{b}$ (abbreviated vector of ones plus bias).

Matrix measures

In this work, we leveraged the L1 and L2 norms induced in the matrix space. Norms L1 and L2 are induced by p norms for $p = 1$ and $p = 2$ for vectors [19] presented in equation (4.4).

$$\|x\|_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}} \quad (4.4)$$

Based on these norms we propose two ways to calculate distance between matrices *distance* depicted in (4.5)

$$distance = \|A - B\|_p \quad (4.5)$$

and *absolute distance* depicted in (4.6).

$$absoulte\ distance = \left| \|A - B\|_p \right| \quad (4.6)$$

where $p = 1$ or $p = 2$ correspond to L1 and L2 respectively. Since the L1 measure is already symmetric, we do not separate a case with absolute distance. We introduce the absolute value as we want our distance measure to be symmetric.

We calculate the distance between nodes as a distance between the weights matrices of autoencoders trained on them. Additionally, since the linear part of the neural network is an affine transform, we introduce an augmented matrix A :

$$A = \left(\begin{array}{c|c} W & \vec{b} \\ \hline 0...0 & 1 \end{array} \right).$$

This matrix A captures the affine transform since $\vec{a}' = W\vec{a} + \vec{b}$ is equivalent to

$$\begin{pmatrix} \vec{a}' \\ 1 \end{pmatrix} = A \begin{pmatrix} \vec{a} \\ 1 \end{pmatrix}.$$

Another way to calculate the distance between nodes is to calculate the distance between an affine transform that is determined by the (affine $W\vec{a} + \vec{b}$) part of the latent layer of the corresponding autoencoder.

4.3.5 Clustering

The calculated distance between clusters is an input for clustering. In this work, we use agglomerative hierarchical clustering: each instance, in our case node, starts as its cluster. At every step of the iteration, the two closest clusters are connected. The connection between clusters is the closest distance between two instances in

corresponding clusters. The combining of clusters is repeated until we reach the predetermined number of clusters.

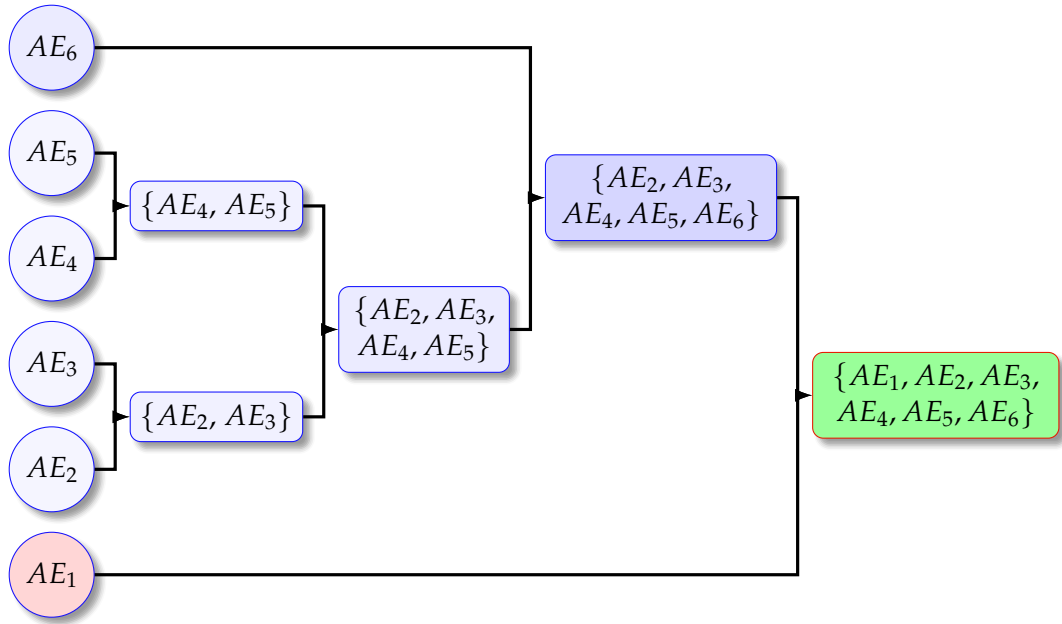


FIGURE 4.3: Example for agglomerative hierarchical clustering of six trained autoencoders from AE_1 to AE_6 : Each autoencoder starts in its own cluster, and pairs of clusters are merged as one moves right the hierarchy.

4.3.6 Evaluating clustering

There are several possible measures to evaluate the goodness of the clustering (e.g., silhouette score) [102]. The silhouette score is a measure used to evaluate the quality of clusters in a clustering algorithm. It ranges from -1 to 1, where a high value indicates that the object is well-matched to its cluster and poorly matched to neighboring clusters. It is calculated in the following way:

1. Calculation for Each Point:

- For each point, calculate two distances:
 - a : The average distance from the point to the other points in the same cluster (measuring cohesion).
 - b : The smallest average distance from the point to points in a different cluster, minimized over clusters (measuring separation).

2. Silhouette Score for Each Point:

- The silhouette score S for a single data point is calculated using:

$$S = \frac{b - a}{\max(a, b)}$$

- This score is between -1 and 1, indicating how well the point is clustered.

3. Overall Silhouette Score:

- The overall silhouette score for the clustering is the mean of all points, giving an overall measure of the clustering quality.

Because the silhouette score only focuses on the distance between the clusters, it gives no information on how informative the constructed clusters are. We evaluate different possible feature extraction methods from the trained autoencoders described in the 4.3.4. These different feature extraction approaches produce different feature spaces. Thus we cannot compare the clustering score, like silhouette score, *between different spaces*. For this reason, we evaluate the relevance of our clustering approaches by evaluating how “interesting” the created clusters are.

The interest of clusters is reflected in how well they separate a specific variable. Since clustering is an unsupervised method, it is reasonable to assume that not all clusters will separate the same variable, as such clustering would produce distinctly *uninteresting* clusters. However, we expect at least one cluster to show a distribution of the target variable that is significantly different from that of the entire dataset.

In this chapter, the target variable is system availability. In other words, clusters will group computing nodes based on the autoencoder model’s latent layer encoding, forming groups with similar availability and, consequently, similar failure rates. From a practical standpoint, this means that an autoencoder model for each node is trained solely on “normal” operation samples and encapsulates information about the likelihood of the node being available or not failing. Clusters of nodes sharing the same likelihood of failure can then be used to optimize the maintenance procedure.

In the whole dataset, the system is available 0.96179% of the time. The most interesting cluster is thus the one where the average availability of a cluster will be as far away from this population average. The best clustering method is the random sampling baseline producing the most interesting cluster.

4.3.7 Random sampling baseline

The relevance of the produced clusters determines the relevance of feature extraction and, consequently, of clustering approaches. Specifically, we observe how well the clusters separate a target variable, in this case, the node’s availability. To claim the relevance of the clustering approaches, we compare them to random sampling. We compare how well the target variable is separated by random sampling to how well clustering methods separate it. We are particularly interested in clustering methods that produce clusters and separations that are very unlikely to occur through random separation.

This work implemented random clustering by producing a random matrix, of the same size and range as the extracted features, which is then passed to clustering algorithms. The produced clusters are thus equivalent to random sampling without replacement. The generation of random clusters is repeated several (in this work 10) times. For each cluster, the distribution of the target variable is calculated; this distribution is then compared to distributions given by clustering methods. In the results (Section 4.4), the range of randomly generated distributions is presented as a box with whiskers plot. Distributions outside the range of random distributions represent interesting patterns uncovered by the clustering method.

4.4 Results of data exploration

This section presents the results of the experimental analysis conducted on a tier-0 supercomputer, Marconi 100, hosted at CINECA. The results were obtained from a statistically significant fraction of the supercomputing nodes, exceeding two hundred, and cover a 10-month time span of the system’s production activity.

4.4.1 Experimental setting

As explained in the methodology Section 4.3, an individual model was trained on each of the 241 randomly selected nodes of Marconi 100. Models were trained semi-supervised, meaning that only normal operation data was used for training. The whole dataset consists of 10 months of operational data collected on Marconi 100.

- Training set: The *first eight* months of the data.
- Test set: The *last two* months of the data.

Autoencoder models were trained on the train set. The cluster analysis was performed only on the test set.

The dataset used in this work comprises a combination of information recorded by Nagios [13], the system administrator’s tool used to visually check the health status of the computing nodes, and the ExaMon monitoring systems. The data spans the first ten months of operation of the Marconi 100 system. The features collected in the dataset are listed in Table 4.1. To align the different sampling rates of various reporting services, 15-minute aggregates of data points were created. This interval was chosen because it is the native sampling frequency of the Nagios monitoring service, from which the labels are derived. Four values were calculated for each 15-minute period and each feature: minimum, maximum, average, and variance.

Features extracted from trained autoencoders are passed to hierarchical clustering. Hierarchical clustering has been chosen as it only requires the pairwise distance between the instances without making any assumptions about the space induced by the distance measure. The number of clusters is set to 20 for all experiments. A

Source	Features
Hardware monitoring	ambient temp., dimm[0-15] temp., fan[0-7] speed, fan disk power, GPU[0-3] core temp., GPU[0-3] mem temp., gv100card[0-3], core[0-3] temp., p[0-1] io power, p[0-1] mem power, p[0-1] power, p[0-1] vdd temp., part max used, ps[0-1] input power, ps[0-1] input voltage, ps[0-1] output current, ps[0-1] output voltage, total power
System monitoring	CPU system, bytes out, CPU idle, proc. run, mem. total, pkts. out, bytes in, boot time, CPU steal, mem. cached, stamp, CPU speed, mem. free, CPU num., swap total, CPU user, proc. total, pkts. in, mem. buffers, CPU idle, CPU nice, mem. shared, PCIe, CPU wio, swap free

TABLE 4.1: A list of features used in training of an anomaly detection model. An anomaly detection model is created only on hardware and application monitoring features. More granular information regarding individual jobs is not collected to ensure the privacy of the HPC system users.

number of clusters is not a tuned parameter; 20 clusters represents roughly 10% of all nodes and is a randomly chosen number.

4.4.2 Trained autoencoder

The trained autoencoder is adopted from the current state-of-the-art semi-supervised approach for anomaly detection [30]. The structure of the autoencoder is presented in Figure 4.4. The autoencoder, used as a binary classifier based on the normalized reconstruction error, achieves an AUC of 0.7602 on the test set.

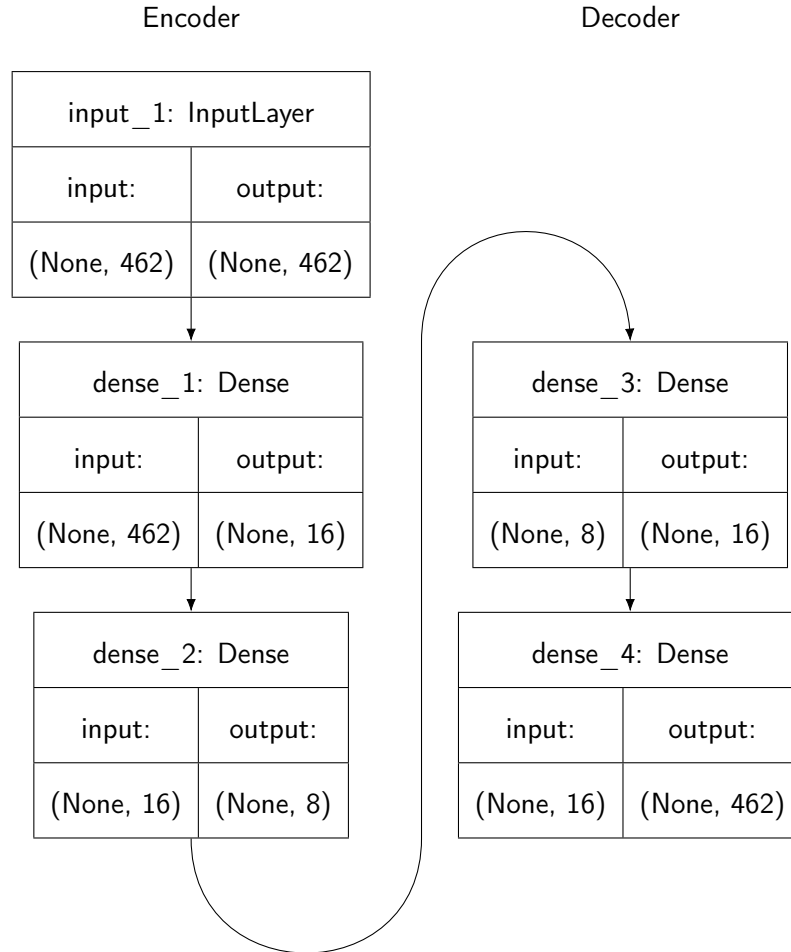


FIGURE 4.4: Architecture of the autoencoder network, adopted from Borghesi et al. (AdaHPC) [30]

Normal operation, defined as the data on which the autoencoder is trained, is determined by the label, system availability, provided by the monitoring systems.

4.4.3 Cluster analysis: normal operation percentage

The proposed approach aims to identify meaningful clusters of nodes that exhibit similar behavior. This similarity in behavior is reflected in the fact that a cluster will have comparable values for at least one relevant feature. In this section, we evaluate the similarity in the average availability rate. In other words, we aim to determine whether the clustering methods can identify clusters with particularly low availability (high failure rate). Among the 241 identified nodes in the test set, the average failure rate is 0.96179. Our goal is to identify clusters with significantly lower availability rates.

In Table 4.2, the minimum average availability rates in a cluster, unidentified by a specific feature extraction approach, are reported. The table shows that the vector of singular values combined by the euclidean distance metric identifies a cluster with the minimum average availability. This availability is also lower than the minimum availability ever achieved by the random method.

Distance measure	Average availability in minimum cluster	Number of nodes in minimum cluster
Sing. vector (Euc.)	0.9286	6
Vector of sing. values (Euc.)	0.8809	7
$W\vec{1} + \vec{b}$ (Euc.)	0.9126	8
$W\vec{1}$ (Euc.)	0.9367	5
W (absolute L2)	0.9191	7
A (absolute L2)	0.9276	5
W (L2)	0.9239	7
A (L2)	0.9124	10
W (L1)	0.9303	8
A (L1)	0.9303	8
Random sampling	0.9021	Not applicable

TABLE 4.2: Minimum average availability within clusters identified by different feature extraction methods presented in 4.3.4. Vector of singular values identifies a cluster with the lowest average availability (highest anomaly rate). This is the most interesting method as it separates the target variable (node availability) the best. None of the proposed methods identify a cluster with a single node.

In Figure 4.5 and Figure 4.6 average availability per node is plotted (red dots). Results of random sampling without replacement are presented as a box plot. The average error rate across all nodes (0.96179) is marked with a violet dotted line. Area

of values, observed in a random process, are marked with gray. Values *never* observed by the random process are left white.

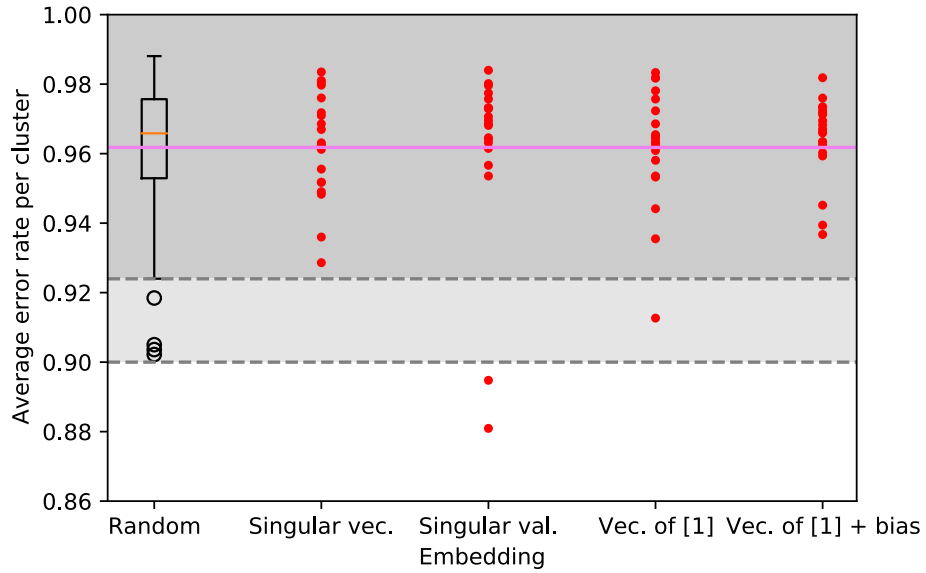


FIGURE 4.5: Average error rate per cluster. Representation of nodes with a vector of singular values identifies two clusters with significantly higher anomaly rate than the whole population.

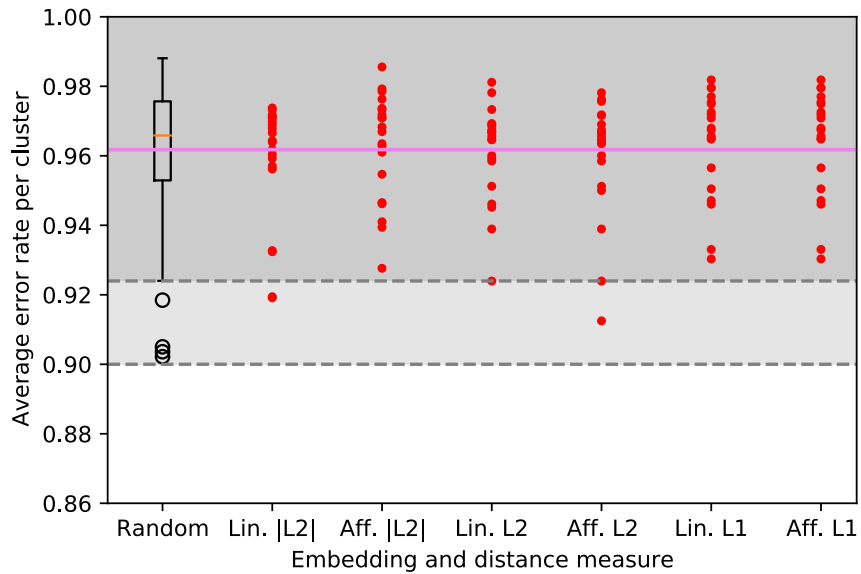


FIGURE 4.6: Average error rate per cluster. Matrix-based feature extraction performs worse than the vector methods.

Analyzing Figures 4.5 and 4.6 we observe that only the vector of singular values produced cluster with averages never observed in random samples.

We propose to use the deviation from population average availability to evaluate the goodness of the clustering results. The vector of singular values, extracted from the weights matrix of the latent layer of the trained autoencoder, identifies two clusters with an average overall availability **lower than 89%** (compared to the 96% population average). The proposed method’s ability to identify these clusters is significant as the autoencoders have no access to the availability label during training.

The clustering method based on a vector of singular values combined with euclidean distance identifies two clusters with particularly low average availability. Such low average availability has also never occurred in a random selection of clusters. Low average availability means that hierarchical clustering based on singular value decomposition of weights matrix produces non-trivial clusters that are extremely unlikely to be matched by a random selection of clusters.

Identifying interesting clusters regarding availability is a non-trivial result as a neural network has no access to that label during training.

This promising result suggests that the created clusters share similar availability, and thus clusters can be created based on autoencoder semi-supervised models latent layer information. This cluster can then be used during the system’s lifetime to create canaries to focus the maintenance over nodes belonging to the same cluster of the canary node.

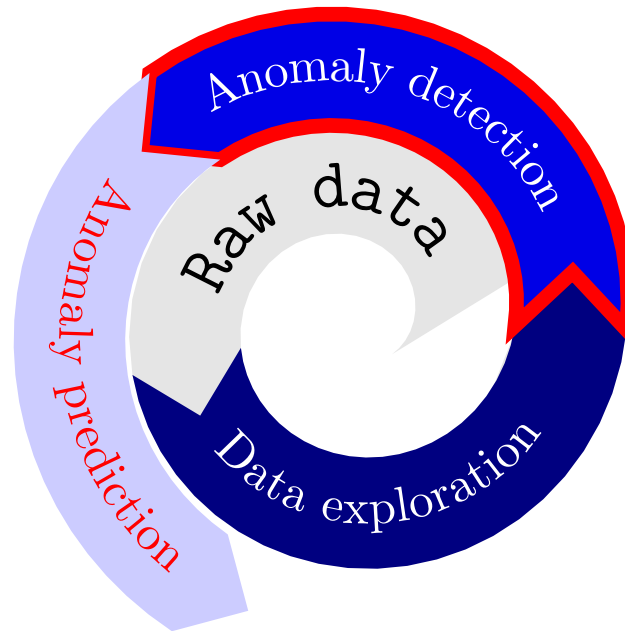
4.5 Conclusions of data exploration

This work opens the possibility of extracting additional information from the state-of-the-art approach towards anomaly detection in the HPC setting. Besides using per-node autoencoder models for anomaly detection like several works by Borghesi et al. such as AdaHPC [30], [24] or [28], it is also possible to construct informative clusters from the parameters of the trained neural networks themselves.

We demonstrate the usefulness of the identified clusters on a concrete example: identifying clusters with the abnormal failure rate. This result is significant as the neural networks, from where the features are extracted, have no *access to that label during training*. Still, our approach can identify two clusters of nodes with lower availability (higher failure rate) than the population average.

We stress the fact that with this approach, clusters can be created based on a model trained on the first month of operations and then applied for the remaining lifetime of the system to focus maintenance to the nodes belonging to the same cluster containing the node which has experienced failures. System administrators focus their regular inspections only on canary nodes, each representative of one cluster.

Chapter 5



Anomaly detection – *RUAD*

The work presented in this chapter is based on the paper *RUAD: Unsupervised anomaly detection in HPC systems* by Molan et al., published in the *Future Generation Computer Systems* journal in 2023 [98].

5.1 Introduction to anomaly detection

Recent trends in the development of HPC systems (such as heterogeneous architecture and higher-power integration density) have increased the complexity of their management and maintenance [121]. A typical contemporary HPC system consists of thousands of interconnected nodes; each node usually contains multiple different accelerators such as graphical processors, field programmable gate arrays (FPGAs), and tensor cores [90]. Monitoring the health of all those subsystems is an increasingly daunting task for system administrators. To simplify this monitoring task and reduce the time between anomaly insurgency and response by the administrators, automatic anomaly detection systems have been introduced in recent years [107].

Anomalies that result in downtime or unavailability of the system are expensive events. Their cost is primarily associated with the time when the HPC system cannot accept new compute jobs. Since HPC systems are costly and have a limited service lifespan [114], it is in the interest of the system’s operator to reduce unavailability times. Anomaly detection helps in this regard as it can significantly reduce the time between the fault and the response by the system administrator, compared to manual reporting of faulty nodes [30].

Modern supercomputers are endowed with monitoring systems that give the system administrators a holistic view of the system [107]. Data collected by these monitoring systems and historical data describing system availability are the basis for machine learning anomaly detection approaches [24, 27, 108, 109, 55], which build data-driven models of the supercomputer and its computing nodes. In this work, we focus on CINECA Tier-0 HPC system (Marconi 100 [65, 22] ranked 9th in Jun. 2020 TOP500 list [131]), which employs a holistic monitoring system called ExaMon [15].

Production HPC systems are reliable machines that generally have very few downtime events - for instance, in Marconi 100 at CINECA, timestamps corresponding to faulty events represent, on average, only 0.035% of all data. However, although anomalies are rare events, they still significantly impact the system’s overall availability - during the observation period, there was at least one active anomaly (unavailable node) 14.4% of the time.

State-of-the-art methods for anomaly detection on HPC systems are based on supervised and semi-supervised approaches from the Deep Learning (DL) field [30]; for this reason, these methods require a training set with accurately annotated periods of downtime (or anomalies). In turn, this requires the monitoring infrastructure to track downtime events; in some instances, this can be done with specific software tools (e.g., Nagios [13]), but properly configuring these tools is a complex and time-consuming task for system administrators.

So far, the challenges of anomaly detection on HPC systems have been approached by deploying anomaly reporting tools by training the models in a supervised or semi-supervised fashion [30, 92, 135, 108]. The need for an accurately labelled training set is the main limitation of current approaches as it is expensive, in terms of time and effort of the system administrators, to be applied in practice.

Downtime tracking also has to be able to record failures with the same granularity as the other monitoring services. Some methods in production HPC systems only record downtime events by date [121, 90, 107]. In most production HPC systems, accurate anomaly detection is thus not readily achievable.

For this reason, the majority of the methods from the literature were tested on historical or synthetic data or in supercomputers where faults were injected in a carefully controlled fashion [105].

Another limitation for the curation of an accurately labeled anomaly dataset is the short lifetime of most HPC systems. In the HPC sector, a given computing node and system technology have a lifetime of between three and five years. Short lifetime means, in practice, that the vendor has no time to create a dataset for training an anomaly detection model before the system is deployed to the customer site.

A completely unsupervised anomaly detection approach could be deployed on a new node or even on an entirely new HPC system. It would then learn online and without any interaction with the system administrators. Additionally, such a system would be easier to deploy as it would require no additional framework to report and record anomalous events (in addition to the monitoring infrastructure needed to build the data-driven model of the target supercomputer - a type of infrastructure which is becoming more and more widespread in current HPC facilities [107]).

Unsupervised anomaly detection approaches for HPC systems exist such as [49, 102, 36]. They either work on log or sensor data. Approaches based on log data [49, 36], while useful, can only offer a post-mortem and restricted view of the supercomputer state.

The current state-of-the-art for anomaly detection on sensor data [102] is based on clustering, which requires a degree of manual analysis from system administrators and offers poor performance compared to semi-supervised methods. The semi-supervised methods [30, 24, 25], based on the dense autoencoders, which are trained to reproduce their input, could be trained in an unsupervised fashion. However, none of the presented works has explored this possibility. According to the current state-of-the-art, the models would perform worse as the dense autoencoder is also capable of learning the characteristics of the anomalies [30, 24, 25].

We propose an *unsupervised* approach: *RUAD* (Recurrent Unsupervised Anomaly Detection) that works on sensor data and *outperforms* all other approaches, including the current state-of-the-art semi-supervised approach (Anomaly Detection and Anticipation in High Performance Computing Systems - AdaHPC) [30] and current state-of-the-art unsupervised approach [102].

RUAD achieves that by taking into account temporal dependencies in the data. We achieve that by using Long Short-Term Memory (LSTM) cells in the proposed neural network model structure, which explicitly take into consideration the temporal dimension of observed phenomena.

We also show that the *RUAD* model, comprising of LSTM layers, is capable of learning the characteristic of the normal operation even if the anomalous data is present in the test set - the *RUAD* model is thus able to be trained in an *unsupervised manner*.

RUAD targets single HPC computing nodes: we have different anomaly detection models for each computing node.

The motivation behind this is *scalability*: in this way, each node can be used to train its own model with minimal overhead - moreover, this strategy would work in

larger supercomputers as well, as if the number of nodes increases, we just have to add new detection models.

5.1.1 Motivation

The primary motivation for this work is to propose a novel approach that relies *only on the fact that the anomalies are rare events* and works at least equally well when trained in an *unsupervised manner* as it does when trained in *semi-supervised manner* - this has not been the case in the current state-of-the-art.

A completely unsupervised anomaly detection approach - such as the one presented in this chapter - could be deployed on a new node or even on an entirely new HPC system. It would then learn online and without any interaction with the system administrators. Additionally, such a system would be easier to deploy as it would require no additional framework to report and record anomalous events, apart from the monitoring infrastructure needed to build the data-driven model of the target supercomputer—a type of infrastructure that is becoming increasingly widespread in current HPC facilities. There exist unsupervised approaches for anomaly detection such as [49, 102, 36]. They either work on log data or on sensor data. Approaches based on log data [49, 36], while useful, can only offer a post-mortem and restricted view of the supercomputer state. The current state-of-the-art for anomaly detection on sensor data [102] is based on clustering which requires a degree of manual analysis from system administrators and offers poor performance compared to semi-supervised methods.

Among the semi-supervised approaches that have been tested on real, in-production anomalies, the current state-of-the-art approach [30] is based on a dense autoencoder and takes (almost) no advantage of temporal dependencies within the data (the only temporal dependency-aware element of the approach presented in [30] is the exponential moving average anomaly threshold).

Interestingly, it was also shown that the signal produced by such a model, in some cases, anticipates the occurrence of an anomaly (the model predicts the anomaly before it occurs).

The capability to anticipate an anomaly means that the anomalies are events that have temporal dynamics - part of the characteristic behavior of an anomaly occurs *before* standard software-based monitoring tools (such as Nagios) record an anomaly. This happens as these tools need to be configured by system administrators, which, as human, might only take into account a limited set of indicators known to be important, while an automated DL method can extract information from *all* available sensors. For this reason, we seek to extend the approach presented by Borghesi et al. [30] by taking into account temporal dependencies in the data.

We achieve that by using Long Short-Term Memory (LSTM) cells in the proposed neural network model structure, which explicitly takes into consideration the temporal dimension of observed phenomena.

We also show that the *RUAD* model, comprising of LSTM layers, is capable of learning the characteristic of the normal operation even if the anomalous data is present in the test set - *RUAD* model is thus able to be trained in an *unsupervised manner*.

5.2 Related work

The drive to detect events or instances that deviate from the norm (i.e. *operational anomalies*) is present across many industrial applications. One of the earliest applications of anomaly detection models was credit card fraud detection in the financial industry [103, 1]. Recently, anomaly detection (and associated predictive maintenance) has become relevant in manufacturing industries [80, 118], internet of things (IoT) [86, 40, 146], energy sector [57], medical diagnostics [152, 10], IT security [119], and even in complex physics experiments [93].

Typically, anomalies in an HPC system refer to periods of (and leading to) sub-optimal operating modes, faults that lead to failed or incorrectly completed jobs, or node and other components hardware failures. While HPC systems have several possible failure mitigation strategies [59] and fault tolerance strategies [89], anomalies of this type still significantly reduce the amount of compute time available to users [23]. The transition towards Exascale and the increasing heterogeneity of hardware components will only exacerbate the issues stemming from failures, and anomalous conditions that already plague HPC machines [121, 107, 66]. A DARPA study estimates that the failures in future exascale HPC systems could occur as frequently as once every 35-39 minutes [20], thus significantly impacting the supercomputing availability and system administrator load.

However, when looking at specific components and not at the entire HPC system (e.g., considering a single computing node), faults remain very rare events, thus falling under the area of anomaly detection, which can be seen as an extreme case of supervised learning on unbalanced classes [112].

Because data regarding normal operation far exceeds data regarding anomalies, classical supervised learning approaches tend to overfit the normal data and give a sub-optimal performance on the anomalous data [113].

In order to mitigate the problem of unbalanced classes, the anomaly detection problem is typically approached from two angles. Approaches found in the current state-of-the-art that address the class imbalance either modify the data [81] or use specialized techniques that work well on anomaly detection problems [30].

Data manipulation approaches address the dataset imbalance either by decreasing the data belonging to normal operation (*under sampling the majority class*) or by oversampling or even generating anomalous data (*over sampling minority class*) [81].

Data manipulation for anomaly detection in HPC systems has not yet been thoroughly studied. Conversely, most existing approaches rely on synthetic data generation, e.g., injection of anomalies in real (non-production) supercomputers or HPC simulators [30].

Another research avenue exploits the abundance of normal data from HPC systems using a different learning strategy, namely semi-supervised ML models. Instead of learning on a dataset containing multiple classes – and consequently learning the characteristics of all classes – semi-supervised models are trained only on the normal data. Hence, they are trained to learn the characteristics of the normal class (the majority class in the dataset). Anomalies are then recognized as anything that does not correspond to the learned characteristic of the normal class [112, 24, 28, 25, 145].

Regarding the type of data used to develop and deploy anomaly detection systems, we can identify two macro-classes: system monitoring data collected by holistic monitoring systems (i.e. ExaMon [15]) and log data. This data is then annotated with information about the system or node-level availability, thus creating a label associated with the data points.

The label encodes whether the system is operating normally or experiencing an anomaly. Since it is expensive and time-consuming to obtain labelled system monitoring data, a labelled dataset for supervised learning can be obtained by "injecting" anomalies into the HPC system (like the method proposed by Netti et al. [105]). Labels are important for supervised, semi-supervised and unsupervised approaches. In the first case, they are used to compute the loss, in the second case to identify the training dataset and validation, and in the third case, only for validation.

This data can then be used in a supervised learning task directly or after processing new features (feature construction). Examples of this approach are [134, 135, 2] where authors use supervised ML approaches to classify the performance variations and job-level faults in HPC systems. For fault detection, [108, 105] propose a supervised approach based on Random Forest (an ensemble method based on decision trees) to classify faults in an HPC system.

All mentioned approaches use synthetic anomalies injected into the HPC system to train a supervised classification model.

Approaches proposed by Borghesi et al. [30] and Molan et al. [92] are among the few that leverage *real* anomalies collected from production HPC systems as opposed to injected anomalies. In this work, we are interested in real anomalies, and thus, we will not include methods using synthetic/simulated data or injected anomalies in our quantitative comparisons.

All mentioned approaches do not take into account temporal dependencies of data, as the models are not trained on time series but on *tabular data containing no temporal information*.

System monitoring data approach [5] is the first to take into account temporal dependencies in data by calculating statistical features on temporal dimension (aggregation, sliding window statistics, lag features). Most approaches that deal with *time series anomaly detection* do so on system log data. Labelled anomalies are either analyzed with log parsers [16] or detected with deep learning methods.

Deep learning methods for anomaly detection are based on LSTM neural networks as they are a proven approach in other text processing fields.

Compared to labelled training sets, much less work has been done on unlabelled datasets - despite this case being much more common in practice. So far, all research on unlabelled datasets has focused on system log data. [49] propose a *k*-means based unsupervised learning approach that does not take into account temporal dynamics of the log data.

A clustering-based approach on sensor data is proposed by [102]. This approach will serve as one of the baselines in the experimental section, as it is the only unsupervised approach that uses sensor data instead of log data.

An approach [36] works on time series data in an unsupervised manner. It uses the LSTM-based autoencoder and is trained on the existing log data dataset. The proposed anomaly detector on system log data achieves the AUC (area under the receiver operating characteristic curve) of 0.59. Although it works on a drastically different type of dataset (log data as opposed to system monitoring data), it is the closest existing work to the scope of the research presented in this work.

As we show later in the chapter, we can achieve much better results than the one reported for the log data models [36] by deploying an unsupervised anomaly detection approach on system monitoring data on a per-node basis. Table 5.1 summarizes the most relevant approaches described in this section, focusing on the training set and temporal dependencies.

The novelty of the proposed approach is, in relation to the existing works, three-fold:

- it introduces an *unsupervised time-series based* anomaly detection model named *RUAD*;
- it proposes a deep learning architecture that captures *time dependency*;
- the approach is evaluated on a *large scale production* dataset with *real anomalies* – this is the largest scale evaluation ever conducted on this kind of problem, to the best of our knowledge.

	Tabular data	Time series
Supervised	<p>E2EWatch: An End-to-End Anomaly Diagnosis Framework for Production HPC Systems [3]</p> <p>Online Fault Classification in HPC Systems through Machine Learning [109]</p>	<p>Proctor: A Semi-Supervised Performance Anomaly Diagnosis Framework for Production HPC Systems [5]</p> <p>Interpretable Anomaly Detection for Monitoring of High Performance Computing Systems [16]</p> <p>DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning [55]</p>
Semi-supervised	<p>Anomaly Detection and Anticipation in High Performance Computing Systems [30]</p> <p>Anomaly detection using autoencoders in HPC systems [24]</p> <p>A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems [28]</p> <p>Online anomaly detection in hpc systems [25]</p>	
Unsupervised	<p>K-means Application for Anomaly Detection and Log Classification in HPC [49]</p> <p>Ranking Anomalous High Performance Computing Sensor Data Using Unsupervised Clustering [102]</p>	<p>Anomaly Detection From Log Files Using Unsupervised Deep Learning [36]</p>

TABLE 5.1: Summary of anomaly detection approaches on HPC systems

5.2.1 Novelty of the developed approach

To recap, in this chapter, we propose an anomaly detection framework that can handle complex system monitoring data, scale to large-scale HPC systems, and be trained even if no labeled dataset is available. The key contributions presented in this work are:

- We propose a completely *unsupervised* anomaly detection approach (RUAD) that exploits the fact that the anomalies are rare and explicitly considers the *temporal dependencies* in the data by using LSTM cells in an autoencoder network.
 - The resulting deep learning model *outperforms* the previous state-of-the-art semi-supervised approach [30], based on time-unaware autoencoder networks.

- On the dataset presented and analysed in this work (collected from the Marconi 100 supercomputer), the previous approach achieves an AUC test set score of 0.7470.
 - In contrast, our unsupervised approach achieves the best test set AUC score of 0.7672.
 - To the best of our knowledge, this work is the first time such an approach has been applied to the field of HPC system monitoring and anomaly detection.
- We have conducted a *very large-scale experimental evaluation* of our methods. We have trained four different deep learning models for each of the 980+ nodes of Marconi 100.
 - To the best of our knowledge, this is the largest scale experiment relating to anomaly detection in HPC systems, both in terms of the number of considered nodes and length of time.
 - Previous works only evaluate the models on a subset of nodes with a short observation time (for instance, only analyzed 20 nodes of the HPC system over two months [30]). Per-node training of models also demonstrates the feasibility of *per node* models for large HPC systems. The training time for the individual model was under 30 minutes on a single NVIDIA Volta V100 GPU.

5.3 Methodology for anomaly detection

In this section, we describe the proposed approach for unsupervised anomaly detection. We do not directly introduce the proposed method (the LSTM autoencoder deep network) as we want to show how it is a significant extension to the current state-of-the-art; thus, we start by introducing three baseline methods:

- (i) exponential smoothing (serving as the most basic method for comparison,
- (ii) unsupervised clustering,
- (iii) the dense autoencoder used in [30].

We then describe our approach in detail and highlight its key strengths (the unsupervised training regime and the explicit inclusion of the temporal dimension).

5.3.1 Node anomaly labeling

We aim to recognize the severe malfunctioning of a node that prevents it from executing regular compute jobs. This malfunctioning does not necessarily coincide with removing a node for the production, as reported by Nagios.

In our discussions with system administrators of CINECA, we have concluded that the best proxy for node availability is node status state, as reported by Nagios.

For this reason, we have created a *new label* called *node anomaly* that has a value 1 if any subsystem reported by Nagios reports a critical state. From these events (reported anomalies), we then filter out known false positive events based on reporting tests or configurations in Jira.

Jira is a popular project management tool developed by Atlassian. It’s widely used for issue tracking, project management, and agile software development [142]. In CINECA, it is used to track maintenance work on HPC systems. Jira logs are supplied by CINECA. The labels used in our previous work [30] do not apply to Marconi 100 as they were extensively used to denote nodes being removed from production for testing and calibration.

In this work, we are examining the early period of the HPC machine life-cycle, when several rounds of re-configuration were performed, thus partially disrupting the normal production flow of the system. Comparing the two labelling strategies in Table 5.2, we can see that the overlap between the two is minimal. Additionally, there are far fewer anomalies as reported by the *node anomaly* mainly because the Marconi 100 went through substantial testing periods in the first ten months of operation where nodes are marked as removed from production while still functioning normally.

In this work, class 0 or class 1 will *always* refer to the value of *node anomaly* being 0 or 1 respectively. Normal data is all data where *node anomaly* has value 0 and *anomalies* are instances where *node anomaly* has value 1. The dataset, preserving the same meaning behind the labels, was expanded in a publication in Nature [31], where RUAD was featured as a case study.

	Node anomaly	
	0	1
Removed from production: False	12 139 560	4 280
Removed form production: True	15 783	12

TABLE 5.2: Comparison between *removed from production* and *node availability*. The anomalies studied in this work (node availability) significantly differ (and are more reliable) from anomalies studied in previous works. The new labels also mark much fewer events as anomalous.

5.3.2 Reconstruction error and result evaluation

In this part, we formally present the theoretical background underpinning the practical approaches for anomaly detection described in the subsequent sections.

The problem of anomaly detection can be formally stated as a problem of training the model \mathbf{M} that **estimates the probability \mathbf{P}** that a sequence of vectors of length W ending at time t_0 represents an anomaly at time t_0 . Suppose the following:

- (i) Model M .
- (ii) Time t_0 .
- (iii) Size W of the window for input data.
- (iv) The past window of a time series $\vec{x}_{t_0-W+1}, \dots, \vec{x}_{t_0-1}$.
- (v) Data \vec{x}_{t_0} at the current time t_0 .
- (vi) Probability P of an anomaly.

Let's define the model M gives the **probability of the anomaly of the vector \vec{x}_{t_0} at time t_0** as

$$M : \vec{x}_{t_0-W+1}, \dots, \vec{x}_{t_0-1}, \vec{x}_{t_0} \longrightarrow P(\vec{x}_{t_0} \text{ is an anomaly}). \quad (5.1)$$

based on the whole size W of data $\vec{x}_{t_0-W+1}, \dots, \vec{x}_{t_0-1}, \vec{x}_{t_0}$.

Vector \vec{x}_t collects all feature values at time t ; the features are the sensor measurements collected from the computing nodes. W is the size of the past window that the model M takes as input. In practice, this means that we are looking for a model that given a time series of a certain time length (W) produces as output the probability that the final data point in the time series (\vec{x}_{t_0}) correspond to an anomalous point, rather than a normal one.

Better explanation of the Equation 5.1 with clear presentation of the possible larger window of the input data is in the following Equation 5.2.

$$M : \underbrace{\vec{x}_{t_0-W+1}, \dots, \vec{x}_{t_0-1}}_{\text{the past window}}, \vec{x}_{t_0} \longrightarrow P(\vec{x}_{t_0} \text{ is an anomaly}). \quad (5.2)$$

If the model does not take past values into account - like the dense model implemented as a baseline [30] - and the window size W is 1, the problem can be simplified. For instance, the problem statement for the window W of size equal to 1 can be summarised as:

$$M : \vec{x}_{t_0} \longrightarrow P(\vec{x}_{t_0} \text{ is an anomaly}). \quad (5.3)$$

The crucial difference between Equations 5.1 and 5.3 is that the former takes as input entire time sequences, while the latter works on single data point, disregarding their past. This is one of the key differences between the model proposed in this work (*RUAD*) which *does* take into account temporal sequences, and the current

state-of-the-art based on previous works of Borghesi et al. [30], which takes as input single data points.

In the case of autoencoders, model M is composed of two parts: autoencoder (a neural network) and the anomaly score, which is computed using the reconstruction error of the autoencoder, see Figure 5.1. The reconstruction error is calculated as the sum of the absolute difference between the output of model A and the normalized input value for each feature:

$$Error(t_0) = \sum_i^N |\hat{x}_i - x_i|$$

where N is the number of features and \hat{x}_{t_0} is the output of the model A .

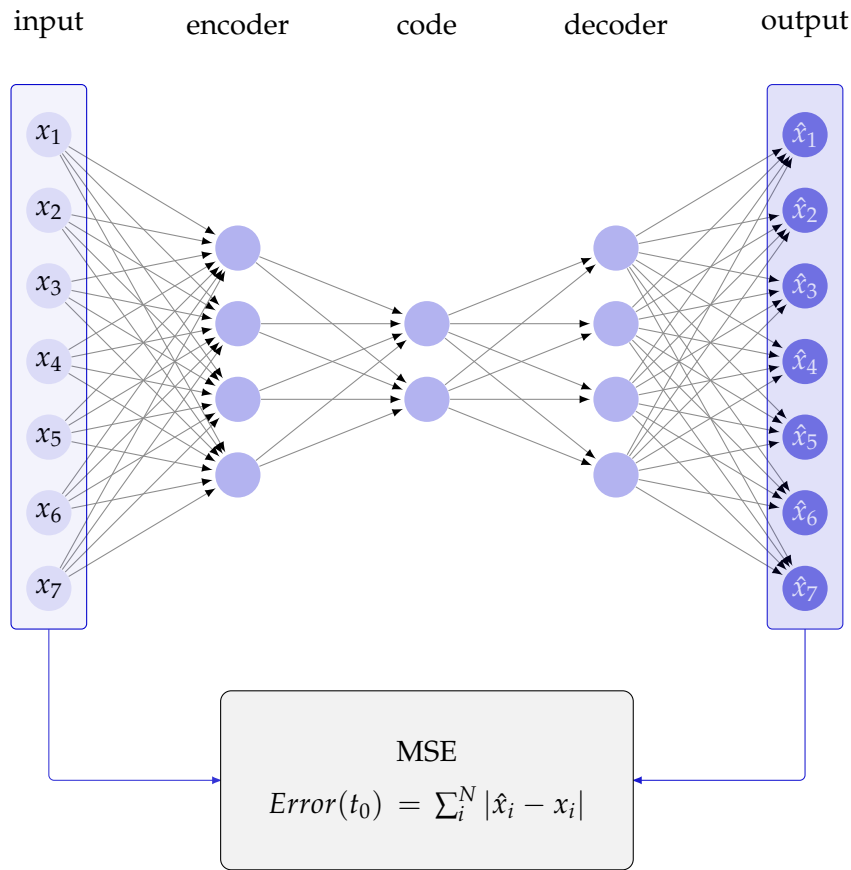


FIGURE 5.1: The model M is composed of two parts: Autoencoder in the upper part and MSE in the lower part of the figure.

The error is then normalized by dividing it by the maximum error on the training set:

$$Normalized\ error(t_0) = \frac{Error(t_0)}{\max(Error(t))}$$

Using the normalized error is then possible to estimate the probability that a certain data point (\vec{x}_{t_0}) is an anomaly, as summarized in the following equation:

$$P(\vec{x}_{t_0} \text{ is an anomaly}) = \begin{cases} 1, & \text{if : Normalized error} \geq 1, \\ \text{Normalized error}, & \text{otherwise} \end{cases} \quad (5.4)$$

The probability is directly proportional to the normalized error:

- A lower error indicates a low probability of the point being an anomaly.
- Larger values represent points that are very likely to be anomalous.
- The maximum probability is capped to 1, in order to maintain its meaning.
- Based on probability $P(\vec{x}_{t_0} \text{ is an anomaly})$, the classifier makes the prediction whether the sequence $\vec{x}_{t_0-W}, \dots, \vec{x}_{t_0}$ belongs to class 1 (anomaly) of class 0 (normal operation).
- This prediction depends on a threshold T , which is a tunable parameter.

The mechanism to decide whether a data point is anomalous or not depending on the threshold T , is summarized in the following equation:

$$\text{Class}(\vec{x}_{t_0}) = \begin{cases} 1, & \text{if : } P(\vec{x}_{t_0} \text{ is an anomaly}) \geq T, \\ 0, & \text{otherwise} \end{cases} \quad (5.5)$$

If the normalized error is larger or equal to T then the data point is classified as an anomaly with a probability equal to one, i.e., we are certain that the point is an anomaly; alternatively, the data point is classified as corresponding normal behaviour.

However, this approach suffers from the fact that it is not trivial to decide which is the optimal threshold – different thresholds might lead to different outcomes (see [29] for an extensive analysis of this phenomenon in a similar anomaly detection setting).

To avoid selecting a specific threshold T , we introduce the receiver operating characteristic curve (ROC curve) as a performance metric, a standard practice to assess the quality of an anomaly detection approach regardless of the threshold choice. It allows us to evaluate the performance of the classification approach for all possible decision thresholds [116]. The receiver operating characteristic curve plots the true-positive rate in relation to the false-positive rate. The random decision represents a linear relationship between the two – for a classifier to make sense, the ROC curve needs to be above the diagonal line. For each specific point on the curve, the better classifier is the one whose ROC curve is above the other.

The overall performance of the classifier can be quantitatively computed as the area under the ROC curve (AUC); a classifier making random decisions has the AUC equal to 0.5. AUC scores below 0.5 designate classifiers that are worse than random choice. The best possible AUC score is 1, which is achieved by a classifier with a true-positive rate equal to 1 while having a false-positive rate equal to 0. Broadly speaking, this is only achievable on trivial datasets or very simple learning tasks.

AUC has been chosen as the main evaluation metric as it is among the most widely used metric in the anomaly detection domain [75]. In particular, it has been suggested that the AUC protects from pitfalls that are common to other relatively simpler metrics (e.g., pure accuracy detection rate or F-score measures) [56].

However, another metric was also added to the experimental evaluation, namely the F-score, which combines precision and recall. Raw accuracy is noted to be very misleading when dealing with anomaly detection tasks rather than pure classification due to the inherent imbalance between normal and abnormal classes.

5.3.3 Trivial baseline: exponential smoothing

Exponential smoothing is implemented as a trivial baseline comparison. It is a simple and computationally inexpensive method that detects rapid changes (jumps) in values. If the anomalies were simply rapid changes in values with no correlation between features, a simple exponential smoothing method would be able to discriminate them. Therefore, we chose exponential smoothing as a first baseline as it is computationally inexpensive and requires no training set.

Additionally, if exponential smoothing performs poorly, this underlines that we are indeed solving a non-trivial anomaly detection problem, for which more powerful models are needed.

For the baseline, we choose to implement exponential smoothing per feature independently. Exponential smoothing for feature i at time t is calculated as:

$$\hat{x}_t^i = \alpha x_t^i + (1 - \alpha) \hat{x}_{t-1}^i, \forall i \in F \quad (5.6)$$

where \hat{x}_t^i is an estimate of x_t^i at time t and α is a parameter of the method called the *smoothing factor* ($0 \leq \alpha \leq 1$), which governs the degree of smoothness of the resulting processed series.

Larger values of α reduce the level of smoothing, and if $\alpha = 1$ the output series is equal to the observed one. In practice, smaller smoothing factors give more importance to past data points, while larger values focus on more recent data. We do this for all features in set F . The estimate at the beginning of the observation is equal to the actual value at time t_0 : $\hat{x}_{t_0}^i = x_{t_0}^i$.

5.3.4 Unsupervised baseline: clustering

A possible approach to unsupervised anomaly detection is to use standard unsupervised machine learning techniques such as k-means clustering proposed by [102]. The clusters are determined on the train set; each new instance belonging to the test set is associated with one of the pre-trained clusters.

We opted for this particular unsupervised technique for the comparison, as it is the only unsupervised method found in the literature, to the best of our knowledge, that uses sensor data instead of logs, thereby ensuring a fair comparison.

It has to be noted, however, that clustering, while belonging to the field of unsupervised machine learning *cannot detect anomalies* in an unsupervised manner - for each of the clusters determined on the train set, the probability for the anomaly has to be calculated. This probability can only be calculated using the labels.

In this work, the clustering approach inspired by [102] is implemented to prove the validity of the obtained results. We have used K-means clustering [49] like it has been proposed in [102]. We have trained the clusters on the train set.

The **Silhouette score** is a measure of performance for a clustering method. It is calculated as

$$S_{score} = \frac{b - a}{\max(a, b)}$$

where a is the mean inter-cluster distance, and b is the mean nearest cluster distance for each sample.

The silhouette score measures how similar an instance is to others in its own cluster compared to instances from the other clusters [120].

Optimal number of clusters is the number of clusters that produces the highest silhouette score on the train set.

Based on the silhouette score on the train set, we have determined the optimal number of clusters for each node.

The percentage of instances that belong to class 1 is calculated for each of the determined clusters. We use this percentage of anomalous instances as the anomaly probability for each instance assigned to a specific cluster. The train and test set split is the same as in all other evaluated methods.

For the practical implementation of the clustering baseline, an anomaly probability has to be assigned to each cluster. To generate and associate this anomaly probability a *post-training* action should be performed by domain experts who should manually identify anomalous clusters of instances, making it a costly operation. This analysis, however, cannot produce anomaly probability estimation that is better than the probability estimated from the actual labels.

By analysing the actual labels, which are not available in real time during an actual deployment and can only be obtained through *post mortem* data analysis, we obtain an upper bound for the performance of the clustering method.

5.3.5 Semi-supervised baseline: dense autoencoder

The competitive baseline method is based on the current state-of-the-art dense autoencoder model proposed by [30]. Autoencoders are types of neural networks (NN) trained to reproduce their input. The network is split into two (most often symmetric) parts: encoder and decoder.

The role of the encoder is to compress the input into a more condensed representation. This representation is called the *latent layer*. To prevent the network from learning a simple identity function, we choose the latent layer to be smaller than the original input size (number of input features) [24].

The role of the decoder is to reconstruct the original input using the latent representation.

Dense autoencoders are a common choice for anomaly detection since we can restrict their expressive power by acting on the size of the latent layer. Compressing the latent dimension forces the encoder to extract the most salient characteristics from the input data; unless the input data is highly redundant, the autoencoder cannot correctly learn to recreate its input after a certain latent size reduction.

In the current state-of-the-art for anomaly detection in production supercomputers [30] the dense autoencoder is used in a semi-supervised fashion, meaning that the network is trained using only data points corresponding to the normal operation of the supercomputer nodes (Class 0).

Semi-supervised training is doable as the normal points are the vast majority and thus are readily available; however, this requires having labelled data or at least a certainty that the HPC system was operating in normal conditions for a sufficiently long period of time.

Once the autoencoder has been trained using only normal data, it is able to recognize similar but previously unseen points.

Conversely, it will struggle to reconstruct new points which do not follow the learned normal behaviour, that is, the anomalies we are looking for; hence, the reconstruction error will be higher.

The structure of the autoencoder model is presented in Figure 5.2.

The dense autoencoder does not take into account the temporal dynamics of the data – its input and target output are the same vector (the set of metrics collected from a computing node). The input and output data for the AdaHPC anomaly detection approach is summarized in the following equation:

$$AdaHPC : \vec{x}_{t_0} \rightarrow \vec{x}_{t_0}. \quad (5.7)$$

This is the standard approach of autoencoder neural networks, which strive to reproduce their input as faithfully as possible.

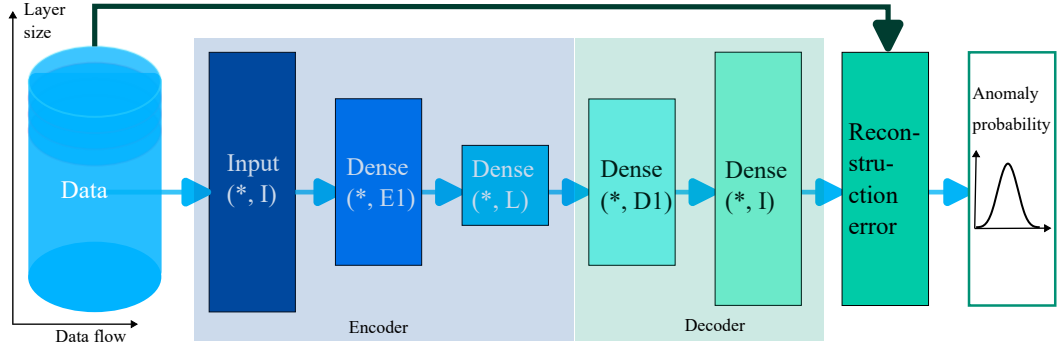
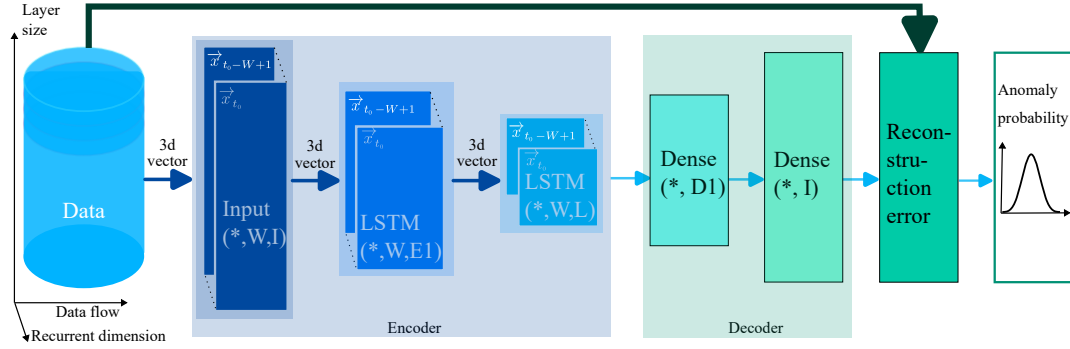


FIGURE 5.2: Structure of baseline model - the dense autoencoder.

FIGURE 5.3: Structure of the proposed *RUAD* model consisting of the LSTM encoder and dense decoder.

The proposed approach replaces the encoder of the baseline model in Figure 5.2 with the LSTM autoencoder in Figure 5.3. The last layer of LSTM encoder returns a vector (not a temporal sequence) which is then passed to the fully connected decoder. W is the window size, I is the size of the input data, L is the size of the latent layer and $E1$ and $D1$ are sizes of encoder and decoder layer respectively. Chosen parameters for L , W , $E1$ and $D1$ are listed in Section 5.4.3.

5.3.6 Recurrent unsupervised anomaly detection: *RUAD*

Moving beyond the state-of-the-art model, we propose a different approach, *RUAD*. It takes as input a sequence of vectors and then tries to reconstruct only the *last vector in the sequence*. The input and output data for the proposed model *RUAD* is depicted in equation 5.8:

$$RUAD : \vec{x}_{t_0-W+1}, \dots, \vec{x}_{t_0} \rightarrow \vec{x}_{t_0} \quad (5.8)$$

The input sequence length is a tunable parameter that specifies the size of the observation window W . The idea of the proposed approach is similar to the dense autoencoder in principle, but with a couple of significant innovations:

1. We are encoding an input sequence into a more efficient representation (latent layer).
2. We train the autoencoder in an unsupervised fashion thus removing the requirement of labelled data.

The key insight in the first innovation is that while the data describing supercomputing nodes is composed of multi-variate time series, the state-of-the-art does not explicitly consider the temporal dimension – the dense autoencoder has no notion of time nor of *sequence* of data points.

To overcome this limitation, our approach works by encoding the sequence of values *leading up to the anomaly*.

The encoder network is composed of Long Short-Term Memory (LSTM) layers, which have been often proved to be well suited to the context where the temporal dimension is relevant [83]. An LSTM layer consists of recurrent cells that have an input from the previous timestamp and from the long-term memory.

To address the scale of current pre-exascale and future exascale HPC systems that will consist of thousands of nodes [107], we want a scalable anomaly detection approach.

The most scalable approach currently for anomaly detection on a whole supercomputer is a node-specific approach as each compute node can train its own model. Still, we want to achieve this by minimally impacting the regular operation of the HPC system. This is why it is important for the proposed solution to have a small overhead.

Additionally, since we want to train a per-node model, we want the method to be data-efficient. To address these requirements, we choose not to make the decoder symmetric to the encoder. The proposed approach is thus comprised of a Dense decoder and an LSTM encoder. LSTM encoder output is passed into a dense decoder trained by reproducing the final vector in an input sequence. The decoder network is thus composed of fully connected dense layers.

The architecture of the proposed approach presented in Figure 5.3 is compared to the state-of-the-art approach presented in Figure 5.2 more advanced.

The reduced complexity of training allows us to train a separate model for each compute node. As shown previously [29], node-specific models provide better results than a single model trained on all data.

We decided to adopt this scheme (one model per node) after a preliminary empirical analysis showed no significant accuracy loss while the training time was vastly reduced (by approximately 50%); this is very important in our case as we trained one DL model for each of the nodes of Marconi 100 (980+), definitely a non-negligible computational effort.

5.3.7 Data pre-processing

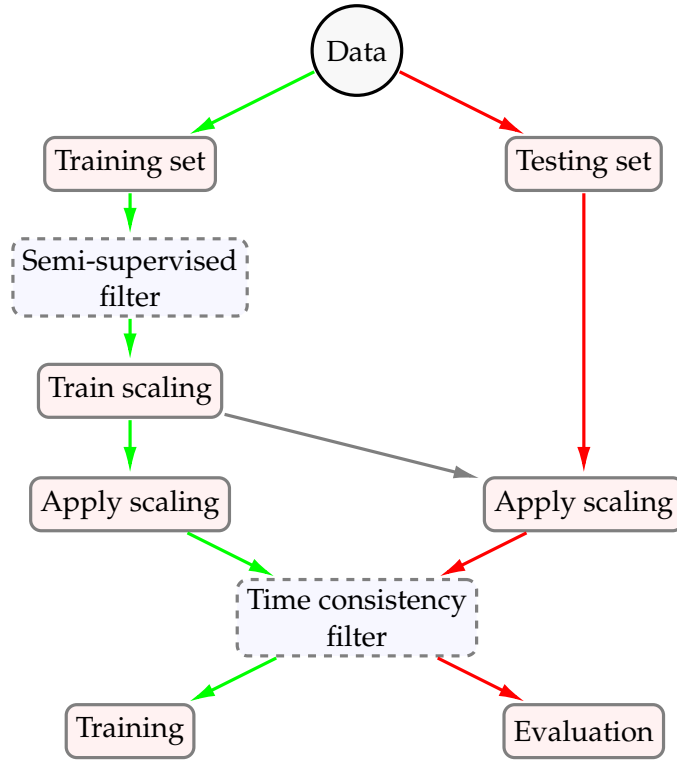


FIGURE 5.4: Data processing schema. The data flow is represented by green (training set) and orange (test set) lines. The scaler is trained on the training set and applied on test set to avoid contaminating the test set. Semi-supervised and time consistency filters are optional and applied only when required by the modeling approach as indicated in Table 5.4

As introduced in Section 5.3.6, our proposed methodology consists of training a model for each node. Figure 5.4 describes the whole data pre-processing pipeline. The data from each node is first split into training and test sets. The training set consists of 80% of the data, while the test set comprises the remaining 20%, which corresponds approximately to the last two months of data. The data is divided into time-consecutive chunks.

It is important to stress that we have chosen to have two non-overlapping datasets for training and testing. This avoids the cross-transferring of information when dealing with sequencing. Moreover, the causality of the testing is preserved: the model is trained without using data from the future, as no random split has been performed, which is more common when not dealing with temporal data. This ensures that the results are valid for practical usage.

For semi-supervised training, the training set is filtered by removing anomalous events, which are identified by the *node anomaly* label as described in Section 5.3.1. This filter is referred to as the *semi-supervised filter*, as shown in Figure 5.4. For unsupervised learning, the training set is not filtered since both normal and anomalous

points are utilized.

- For both the cases (unsupervised and semi-supervised learning), labels are used to evaluate the results.
- After filtering, a scaler is fitted to training data.
- A scaler is a transformer that scales the data to the $[0, 1]$ interval.
- In the experimental part, a min/max scaler is used on each feature [115].

After being fitted on the training data, the scaler is applied to the test data. For rescaling the test set, the minimum and maximum values of the training set are used, following standard practice in deep learning methods.

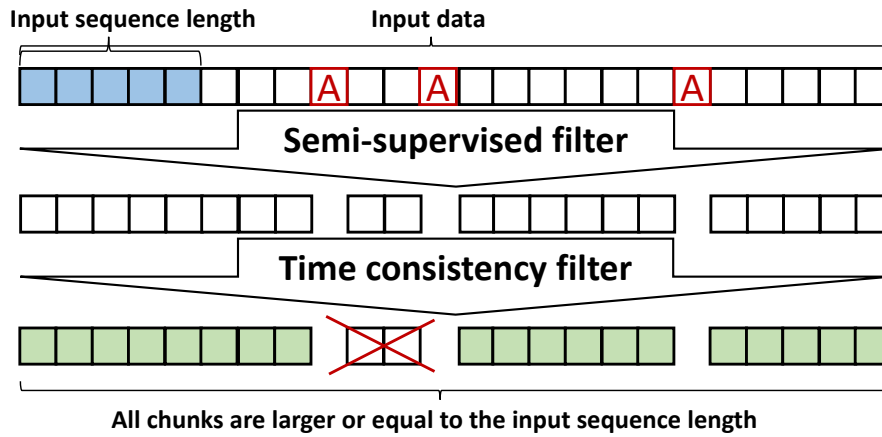


FIGURE 5.5: To ensure time consistency after removing the anomalous data (A in the Figure), we first split the data into chunks of successive timestamps without anomalies. Then we remove all chunks that are shorter than the input sequence length. Training sequences are only generated on the remaining chunks (green in the Figure).

After scaling, both training and test sets are filtered out to ensure time consistency (as depicted in Figure 5.5).

The time consistency filter consists of two steps:

1. The data is divided into chunks of successive timestamps. For semi-supervised training, these successive timestamps do not contain anomalies.
2. All chunks that are smaller than the input sequence of length W are dropped. Batches of sequences of length W are then generated for each chunk individually.

Timestamps with missing values are dropped from the dataset (for all methods); missing timestamps – like anomalies – also split that dataset into smaller chunks. This – unlike anomalies – has no significant impact on the comparison of the methods as it is done for all analysed approaches.

Removing anomalous data points results in splitting the dataset into smaller chunks. As illustrated in Figure 5.5, all data between two anomalous data points closer together than the length of the input sequence W are removed from the training set.

An important observation is that because we have to ensure time consistency, *removing anomalous data points results in also removing normal data points*. This, as discussed in Section 5.4.5, causes *RUAD* (which is unsupervised) to slightly outperform *RUAD_{semi}* (which is semi-supervised). *RUAD* outperforms *RUAD_{semi}*

For larger W , more (normal) data points are dropped from the training set. This contributes to the decline in performance of *RUAD* and *RUAD_{semi}* with larger W (this is fully discussed in Section 5.4.5).

Input sequence length	Average percentage of removed normal data
5	4.7%
10	7.9%
20	13.8%
40	23.5%

TABLE 5.3: Average percentage of removed normal data due to semi-supervised and time consistency filters.

The average percentage (across all nodes in the dataset) of *normal data* that is removed, depending on the length of the input sequence, is collected in Table 5.3.

5.3.8 Summary of evaluated methods

We compare our proposed approach *RUAD* against established semi-supervised and unsupervised baselines. Summary of pre-processing filters is presented in Table 5.4.

The semi-supervised filter is applied to all semi-supervised approaches. A time consistency filter is applied to methods that explicitly consider the temporal dimension of the data: Exponential smoothing and *RUAD*. *RUAD* and the current state-of-the-art anomaly detection approach based on dense autoencoders [30] is evaluated in both semi-supervised and unsupervised versions.

We wish to highlight that, unlike the unsupervised learning baseline [102], our proposed method *RUAD* requires no additional action after the training of the model (like the manual analysis of the clusters).

The approach *RUAD* proposed in this work, works on an *unlabeled dataset* and requires no additional *post training analysis*. A summary of approaches relating to training set requirements is presented in Table 5.5.

Model	Filters		Name
	Semi-supervised	Time consistency	
Trivial baseline: exponential smoothing	NO	YES	EXP
Unsupervised baseline: clustering	NO	NO	CLU
DENSE autoencoder baseline semi-supervised	YES	NO	DENSE _{semi}
DENSE autoencoder baseline unsupervised	NO	NO	DENSE _{un}
RUAD semi-supervised	YES	YES	RUAD _{semi}
RUAD unsupervised	NO	YES	RUAD

TABLE 5.4: Short names and training strategies for examined methods. DENSE_{semi} is the AdaHPC [30].

Method	Training set required	Post-training
EXP	Unlabeled dataset	No action required
CLU [102]	Unlabeled dataset	Assigning anomaly probability to clusters
DENSE _{semi} [30]	Labeled dataset	No action required
RUAD	Unlabeled dataset	No action required

TABLE 5.5: Comparison of implemented approaches relating to the training set requirements.

5.4 Experimental results of anomaly detection

5.4.1 Experimental setting

HPC systems are composed of multiple compute nodes organized in racks. The focus of this work are node-level failures. We thus consider, in this work, nodes of the HPC system to be independent. Future works will study multi-node failures. For this reason, in this work and in line with the state-of-the-art works [105, 24, 30], anomaly detection is performed per node.

The focus of the experimental part of this work is Marconi 100 HPC system, located in the CINECA supercomputing centre. It is a tier-0 HPC system that consists of 980 compute nodes organized into three rows of racks. Each compute node has 32 core CPU, 256 GB of RAM and 4 Nvidia V100 GPUs. In this work, nodes of the HPC

system are considered independent. This is also in line with the current state-of-the-art works [105, 24, 30] where anomaly detection is performed per node. Future works will investigate inter-node dependencies in the anomaly detection task.

Our prior in this case is that the failure of a single node is not strongly linked with other nodes. The assumption that the node failures are independent depends on the type of faults that are registered by the status monitoring service, which does not currently include multi-node failures. For this reason, anomaly detection is performed *per node*.

The monitoring system in an HPC setting typically consists of hardware monitoring sensors, job status and availability monitoring, and server room sensors. In the case of Marconi 100, hardware monitoring is performed by ExaMon [15], and system availability is provided by system administrators [13]. This raw information provided by Nagios, however, contains many false-positive anomalies. For this reason, we have constructed a new anomaly label called *node anomaly* described in Section 5.3.1.

For each of the 980 nodes of Marconi 100, a separate dataset was created. Dataset details are explained in Section 5.4.2.

DENSE and *RUAD* models were trained and evaluated on the node-specific training and test sets for each node. The training set consisted of the first eight months of system operation, and the test set comprised the remaining two months. Such testing split ensures a fair evaluation of the model as described in Section 5.4.2.

For the baseline, the exponential smoothing operation, defined in equation (5.6), was applied only over the test set (as the approach requires no training).

For each node, the scaler (for min and max scaling) was trained on training data and applied to test data. All results discussed in this section are *combined results from all 980 nodes of Marconi 100*.

The dense autoencoder and the *RUAD* model were trained in two different regimes: semi-supervised and unsupervised.

For the semi-supervised training, the semi-supervised filter was applied that removed all data points corresponding to anomalies.

In the unsupervised case, no such filtering was performed. It can hence be noticed one of the key advantages of the unsupervised approach:

No data pre-processing needs to be done and no preliminary knowledge about the computing nodes condition is required.

For all three approaches

1. exponential smoothing,
2. dense autoencoder and
3. the *RUAD*,

the probability for an anomaly (class 1) was estimated from reconstruction error as explained in Section 5.3.2.

The probabilities from the test sets of all nodes from a single modelling approach (e.g. RUAD with observation window of length $W = 40$) were collected together to plot the receiver operating characteristic (ROC) curve that is a characteristic for the modelling approach across all nodes.

For clustering baseline and exponential smoothing (worst performing baselines), the ROC curve is compared against a dummy classifier which randomly chooses the class.

5.4.2 Dataset

The dataset used in this chapter consists of a combination of information recorded by Nagios (the system administration tool used to visually check the health status of the computing nodes) and the ExaMon monitoring systems; the data encompasses the first ten months of operation of the Marconi 100 system.

The procedure for obtaining a *node anomaly label* is described in Section 5.3.1. The features collected in the dataset are listed in Table 4.1 in Subsection 4.4.1. The dataset contains 462 features. The data covers 980 compute nodes and five login nodes. Login nodes have the same hardware as the compute nodes but are reserved primarily for job submission and accounting. Thus we removed them from our analysis. The data is collected by the University of Bologna with approval from CINECA¹.

In order to align different sampling rates of different reporting services (each of the sensors used has a different sampling frequency).

Fifteen (15) minute aggregates of data points were created, with the 15-minute interval selected as it corresponds to the native sampling frequency of the Nagios monitoring service, from which our labels are derived. For each 15-minute period and each feature, four values were calculated: minimum, maximum, average, and variance.

5.4.3 Hyperparameters

Hyper-parameters for all methods discussed in this work were determined based on initial exploration on the set of 50 nodes. The chosen parameters demonstrated the best performance during testing on the initial exploration nodes, achieving the highest AUC score on the test set. Results from the initial exploration set are excluded from the results discussed further in the chapter. Tuned hyperparameters include the structure of the neural nets (number and size of layers) and the smoothing factor of the exponential smoothing:

- Exponential smoothing: smoothing factor $\alpha = 0.1$

¹CINECA is a public university consortium and the main supercomputing centre in Italy [141].

- Clustering: hyper-parameter (number of clusters) is trained on a train set for each node independently.
- Dense autoencoder: Structure of the network consists of 5 layers of shapes: $(*,462), (*,16), (*,8), (*,16), (*,462)$.
- RUAD (LSTM encoder, dense decoder): Structure of the network consists of 5 layers of shapes: $(*,W,462), (*,W,16), (*,W,8), (*,16), (*,462)$. W is the length of the input sequence (observation window). Chosen input sequence lengths W were: 5, 10, 20, 40.

5.4.4 Area under the curve (AUC)

The most basic baseline, exponential smoothing (EXP) is implemented to demonstrate that the anomalies we observe are not simply unexpected spikes in the data signal. Furthermore, exponential smoothing is applied to each feature independently of other features. As shown in Figure 5.6, exponential smoothing performs even worse than a dummy classifier (random choice). Poor performance of exponential smoothing shows that the anomalies we are searching for are more complex than simple jumps in values for a feature.

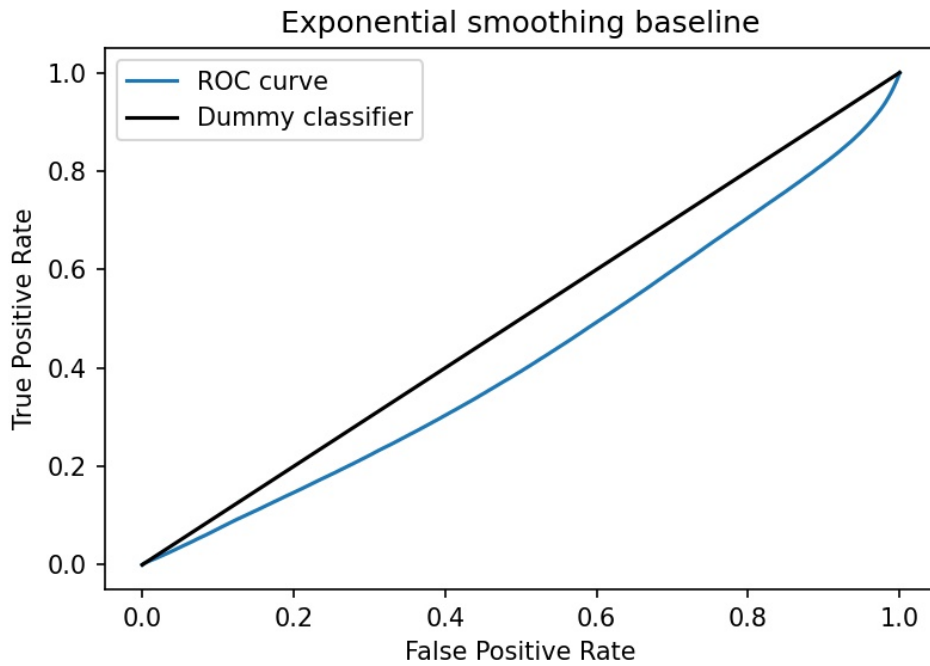


FIGURE 5.6: Combined ROC curve from all 980 nodes of Marconi 100 for the exponential smoothing baseline. Exponential smoothing performs even worse than the dummy classifier - anomaly detection based on exponential smoothing is completely unusable.

The simple clustering baseline performs better than the exponential smoothing baseline and better than the dummy classifier, as seen in Figure 5.7. However, as

we will illustrate in the following sections, it performs worse than any other autoencoder method.

This demonstrates that the problem we are addressing (anomaly detection on an HPC system) requires more advanced methodologies like semi-supervised and unsupervised autoencoders.

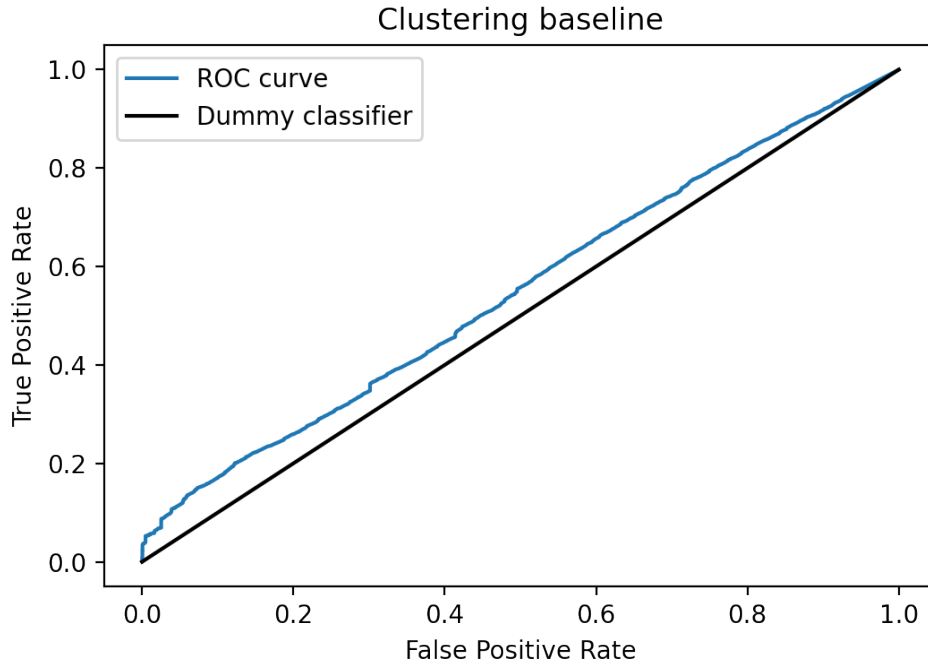


FIGURE 5.7: Combined ROC curve from all 980 nodes of Marconi 100 for the simple clustering baseline. This baseline performs only marginally better than the dummy classifier.

We now consider the dense autoencoder. We train a different network for each computing node of Marconi 100.

These parameters were identified after a preliminary empirical evaluation, albeit not exhaustive, which suggested the best values; furthermore, the initial parameter space was constrained by adhering to indications from previous works in the same domain.

The network structure, including the number and size of layers, was determined through a thorough preliminary exploration, which identified the best values; additionally, the initial parameter space was constrained by adhering to indications provided by previous works in the same domain, as outlined by Borghesi et al. [29].

In line with the existing work [30], the semi-supervised learning approach $DENSE_{semi}$ slightly outperforms the unsupervised learning approach $DENSE_{un}$ as seen in Figure 5.8. The improved performance of the semi-supervised model is due to the peculiarities of the autoencoder network, namely its capability to reconstruct its input.

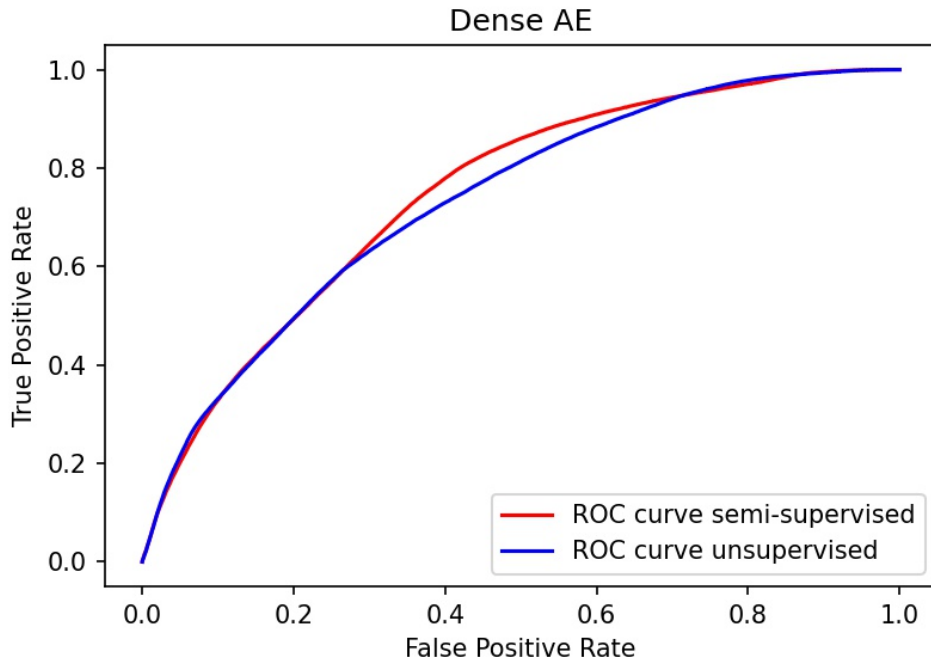


FIGURE 5.8: Combined ROC curve from all 980 nodes of Marconi 100 for the Dense autoencoder model. In the area interesting for practical application - True Positive Rate between 0.6 and 0.9 - semi-supervised approach outperforms unsupervised approach.

For instance, let consider the unsupervised case. If anomalous examples are fed as input to the unsupervised model during the training phase, the autoencoder will learn to represent them in its latent space. Thus, it will be capable to partially reconstruct them as well, albeit with much more difficulty, as these anomalous points are extremely rare (the scarcity of anomalies is a critical assumption for the correct functioning of the unsupervised model).

In turn, this implies that the unsupervised model are slightly less capable of differentiating normal and anomalous points than the semi-supervised model, as the latter has been strictly fed only with normal examples.

The most important parameter of the RUAD model (unsupervised LSTM autoencoder) is the length of the input sequence W that is passed to the model. This parameter encodes our expectation of the length of the dependencies within the data. Since each data point represents 15 minutes of node operation, the actual period we observe consists of $W \times 15 \text{ min}$. In this set of experiments, we selected the following time window sizes: 5 (75 minutes), 10 (2h30), 20 (5h), 40 (10h). These period lengths were obtained after a preliminary empirical evaluation; moreover, these time frames are in line with the typical duration of HPC workloads, which tend to span between dozens of minutes to a few hours [37].

Comparison of different window lengths for the *RUAD* model. For all window lengths, performances of semi-supervised and unsupervised approaches are similar.

Performance of the proposed model (red and blue line) is compared to the state-of-the-art baseline semi-supervised autoencoder proposed by Borghesi et al. [30].

We have trained the model in both semi-supervised $RUAD_{semi}$ and unsupervised $RUAD$ fashion for each selected window length. Results across all the nodes are collected in Figure 5.9 and Figure 5.10.

5.4.5 Comparison of all approaches

The main metric for evaluating model performance is the area under the ROC curve (AUC). This metric estimates the classifiers' overall performance without the need to set a discrimination threshold [116]. The closer the AUC value is to 1, the better the classifier performs. AUC scores for implemented methods are collected in Table 5.6 and Table 5.7.

Method	Combined AUC score
EXP	0.4276
CLU	0.5478
$DENSE_{semi}$	0.7470
$DENSE_{un}$	0.7344

TABLE 5.6: AUC performance of model baselines. According to expectations and existing work [30], semi-supervised dense autoencoder outperforms unsupervised dense autoencoder (highlighted by the higher AUC score).

From Table 5.7, where rows correspond to different training regimes and columns to the window size for the $RUAD$ network, and Table 5.6, where rows represent the performance of different implemented baselines, we observe that the proposed approach outperforms the existing baselines.

The highest AUC achieved by the previous baselines is 0.7470, achieved by the $DENSE_{semi}$. This is outperformed by $RUAD$ for *all window sizes*.

The best performance of $RUAD$ is achieved by selecting the windows size 10 where it achieves an **AUC of 7.672**. This result clearly shows that some temporal dynamics contribute to the appearance of anomalies.

The final consideration is the impact of observation window length W on the performance of the $RUAD$ model. One might expect that considering longer time sequences would bring benefits, as more information is provided to the model to recreate the time series. This is, however, not the case (as seen in Table 5.7) as the $RUAD$ achieves the best performance of 0.7672 with window size 10.

The performance then reduces sharply with window size 40, only achieving an AUC of 0.7473. Several factors might explain this phenomenon. For instance, in

Method	Combined AUC score			
Input sequence length	5	10	20	40
$RUAD_{semi}$	0.7632	0.7582	0.7602	0.7446
$RUAD$	0.7651	0.7672	0.7655	0.7473

TABLE 5.7: $RUAD$ and $RUAD_{semi}$ outperform all previous baselines presented in Table 5.6. In contrast to the dense autoencoders, the proposed approach $RUAD$ performs best in *unsupervised manner*.

tens of hours, the workload on a given node might change drastically. Considering longer time series might thus force the $RUAD$ model to concentrate on multiple workloads, hindering its learning task. Finally, an issue arises from the presence of gaps, or periods of missing measurements, in the collected data, which is a common challenge in many real-world scenarios.

Longer sequences mean that more data has to be cut from the training set to ensure time-consistent sequences; this is because we are not applying gap-filling techniques. We decided not to consider such techniques for the moment, as we wanted to focus on the modelling approach and gap-filling methods tend to require additional assumptions and to introduce noise in the data. Thus, sub-sequences missing some points need to be removed from the data set. Combining these two factors contributes to the model’s decline in performance with longer observation periods.

Considering all discussed factors, the optimal approach is to use the proposed model architecture with window size $W = 10$ (i.e. 2 hours and 30 minutes), trained in an *unsupervised* manner. This configuration outperforms semi-supervised $RUAD_{semi}$ as well as the dense autoencoder. As mentioned in the related work, labelled datasets are expensive to obtain in the HPC setting.

Good unsupervised performance is why this result is promising - it shows us that if the anomalies represented a small fraction of all data, we could train an anomaly detection model even on an unlabeled dataset (in an unsupervised manner).

Such a model not only achieves the state-of-the-art performance but *outperforms* semi-supervised approaches. The best AUC, achieved by the previous AdaHPC $DENSE_{semi}$, is 0.7470. The best AUC score achieved by $RUAD$ is 0.7672. Moreover, unsupervised training makes this anomaly detection model more applicable to a typical HPC (or even datacentre) system.

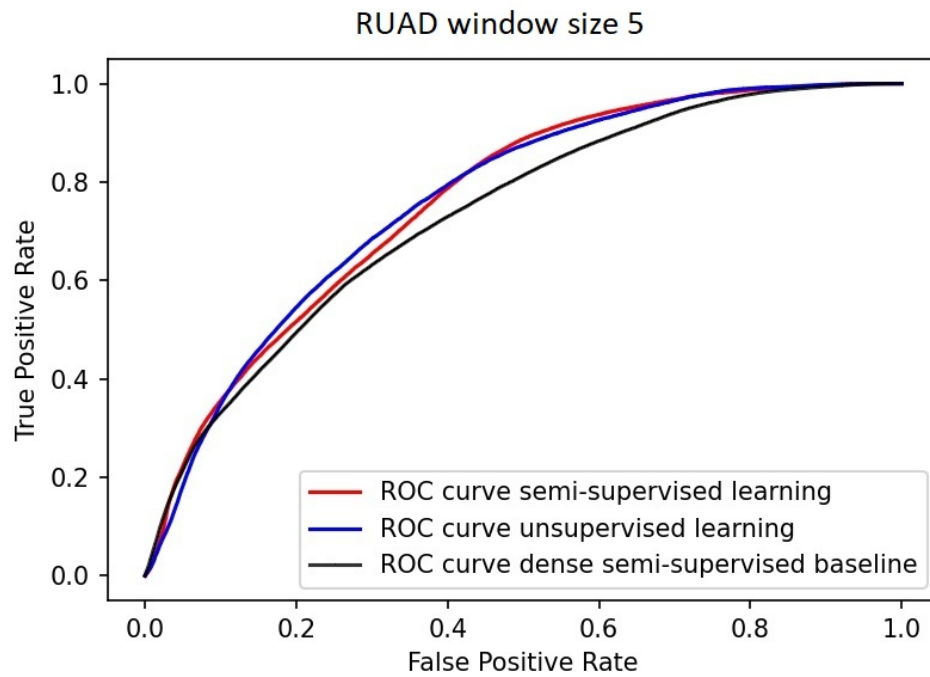
Compared to the previous state-of-the-art for completely unsupervised anomaly detection CLU , which achieves an AUC of just 0.5478 $RUAD$ achieves significantly better results with an AUC of 0.7672. $RUAD$ thus sets a new state-of-the-art for *unsupervised anomaly detection*.

5.4.6 F1 scores

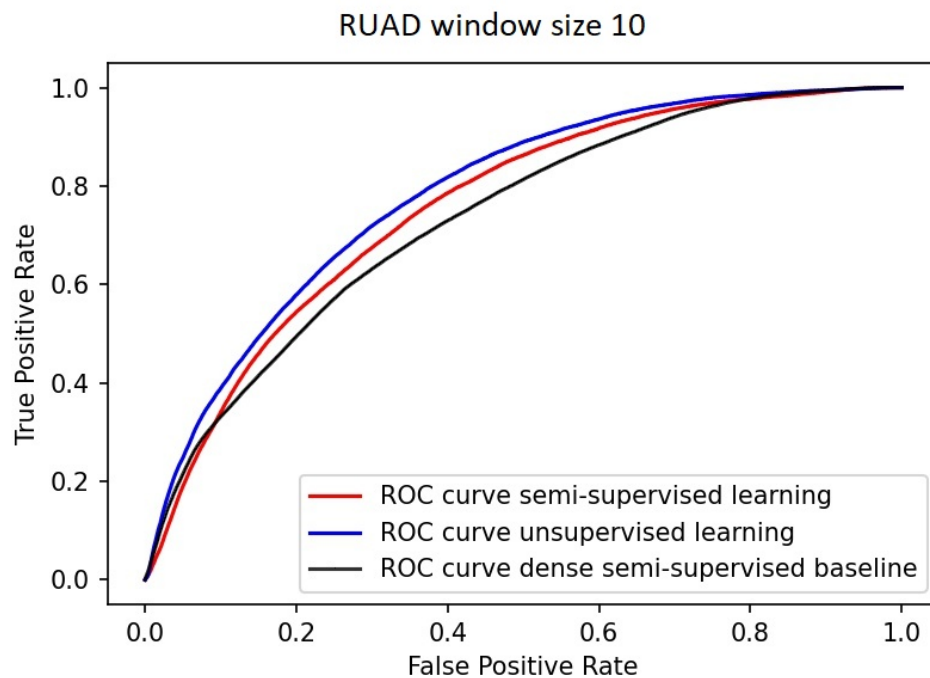
To illustrate the performance of the proposed classifiers, we present the comparison of F1 scores of all the methods with different thresholds in Table 5.8. The threshold 1 represents a trivial classifier that classifies all data as normal operation.

Scores lower than the scores of this trivial classifier represent classifiers that are useless for practical evaluation and should be excluded from the analysis (in Table 5.8 are greyed out). For all autoencoder-based approaches (*RUAD*, *RUAD_{semi}*, *DENSE_{un}*, *DENSE_{semi}*) the difference in F1 score is small for different thresholds. This suggests that the predicted probabilities are either very low (for normal operation) or very high (for anomalies).

The difference between semi-supervised training and unsupervised training is less noticeable but *RUAD* still *outperforms all other approaches*. Particularly interesting is the comparison to the unsupervised benchmark *CLU* that, for the chosen thresholds, performs particularly bad – even worse than exponential smoothing.

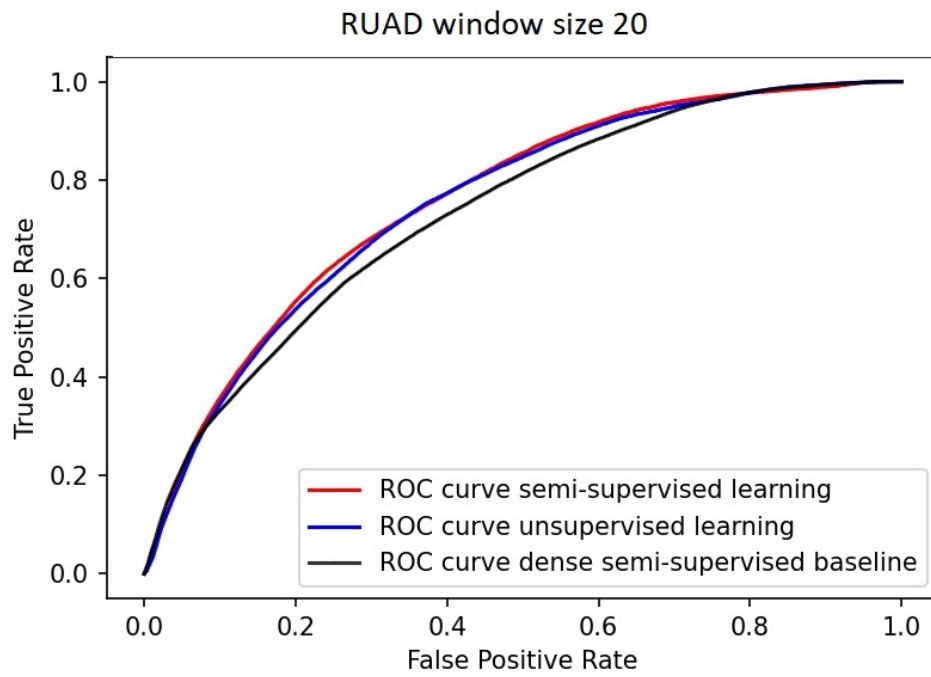


(A) Input sequence length 5

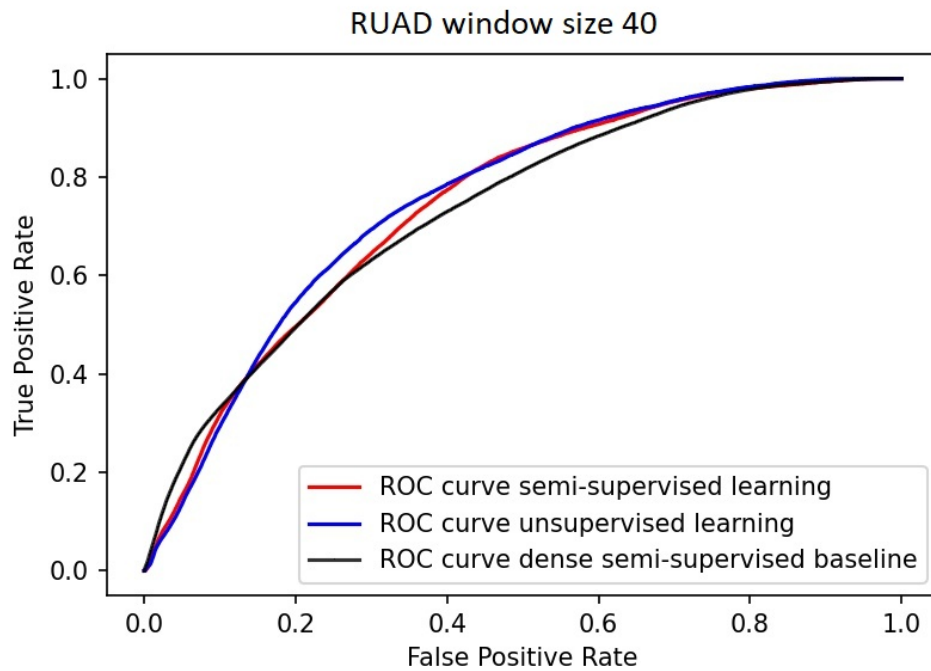


(B) Input sequence length 10

FIGURE 5.9: Combined results from all 980 nodes of Marconi 100 for *RUAD* window sizes 5 and 10



(A) Input sequence length 20



(B) Input sequence length 40

FIGURE 5.10: Combined results from all 980 nodes of Marconi 100 for RUAD window sizes 20 and 40

			F1 score											
			Threshold	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Method	EXP		EXP	0.009	0.767	0.893	0.927	0.941	0.947	0.948	0.948	0.948	0.948	0.894
			CLU	0.009	0.887	0.891	0.895	0.898	0.899	0.899	0.898	0.895	0.895	0.894
			DENSE _{semi}	0.009	0.465	0.941	0.952	0.953	0.953	0.953	0.953	0.953	0.953	0.894
			DENSE _{un}	0.009	0.298	0.926	0.951	0.952	0.953	0.953	0.953	0.953	0.953	0.894
	RUAD _{semi}		5	0.009	0.391	0.936	0.957	0.960	0.960	0.960	0.960	0.960	0.960	0.894
			10	0.009	0.395	0.935	0.957	0.960	0.960	0.960	0.960	0.960	0.960	0.894
			20	0.009	0.420	0.937	0.957	0.960	0.960	0.960	0.960	0.960	0.960	0.894
			40	0.009	0.960	0.960	0.960	0.960	0.960	0.960	0.960	0.960	0.960	0.894
	RUAD		5	0.009	0.333	0.928	0.956	0.960	0.960	0.960	0.960	0.960	0.960	0.894
			10	0.009	0.307	0.924	0.957	0.959	0.960	0.960	0.960	0.960	0.960	0.894
			20	0.009	0.439	0.938	0.958	0.960	0.960	0.960	0.960	0.960	0.960	0.894
			40	0.009	0.960	0.960	0.960	0.960	0.960	0.960	0.960	0.960	0.960	0.894

TABLE 5.8: Combined F1 scores for all compute nodes. F1 scores worse than the trivial classifier (decision threshold 1) are greyed out. *RUAD* outperforms all previous approaches, including the previous state-of-the-art (*DENSE_{semi}* and *DENSE_{un}*).

5.5 Conclusions of anomaly detection

This chapter presents an anomaly detection approach for HPC systems (*RUAD*) that outperforms the current state-of-the-art approach based on the dense autoencoders [30]. Improving upon state-of-the-art is achieved by deploying a neural network architecture that considers the temporal dependencies within the data.

The proposed model architecture achieves the highest AUC of 0.77 compared to 0.75, which is the highest AUC achieved by the dense autoencoders (on our dataset).

Another contribution of this work is that the proposed method – unlike the previous work [30, 92, 135, 108] – achieves the best results in an *unsupervised training* case. Unsupervised training is instrumental as it offers a possibility of deploying an anomaly detection model to the cases where (accurately) labelled dataset is unavailable.

The only stipulation for the deployment of *unsupervised* anomaly detection models is that the anomalies are rare – in our work, the anomalies accounted for only 0.035% of the data. The necessity to have a few anomalies in the training set, however, is not a significant limitation as HPC systems are already highly reliable machines with low anomaly rates [54, 121].

Furthermore, the ability to function without the accurately labelled dataset is a big step toward adopting anomaly detection frameworks to the systems where error reporting is not handled precisely (or even not recorded altogether) [107].

To illustrate the capabilities of the approach proposed in this work, we have collected an extensive and accurately labelled dataset describing the first 10 months of operation of the Marconi 100 system in CINECA [141]. The creation of *accurately labelled* dataset was necessary to compare the performance of different models on the data rigorously.

Because of the large scale of the available dataset, we can conclude that for the model proposed in this work, the unsupervised model *outperforms* semi-supervised model *even if accurate anomaly labels are available*.

This is the first experiment of this type and magnitude conducted on a real in-production data centre, both in terms of the number of computing nodes considered and the length of the observation period.

Results discussed in this work are the collection of data from *all* 980+ nodes of Marconi 100; for each node, 4 different deep learning models were trained and evaluated. The results presented in this work thus contain the characteristics from almost 4000 (3940 to be precise) different instances of deep learning models. Based on this experimental work, we can with confidence claim that the proposed model is the first step towards *unsupervised* at scale and production-ready anomaly detection models.

5.5.1 Future work

Introducing an unsupervised anomaly detection framework opens several exciting possibilities for future work. In the cases where anomalies are rare (such as on data presented in this work), and we have temporal information about data (data is time-coherent or ordered into sequences), the presented approach provides a clear upgrade over dense autoencoder models. As such, it has a broad range of potential application areas.

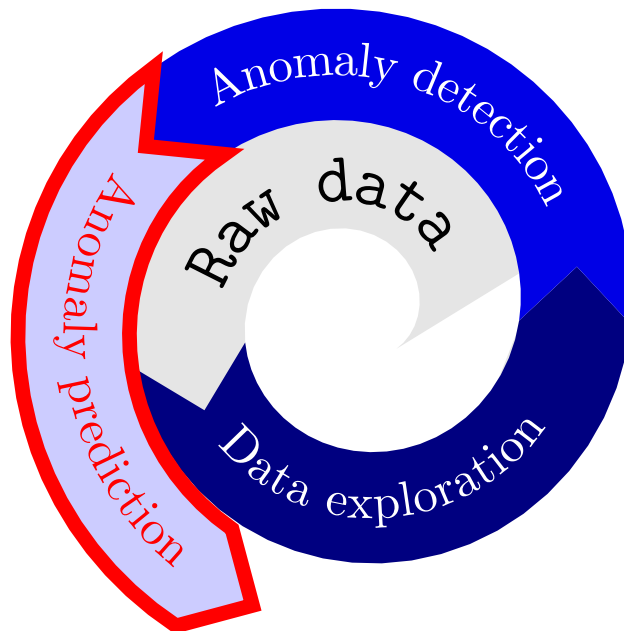
Since it requires no label data, the proposed model could also be used as an anomaly detection tool in cases where system availability is not recorded. In the HPC sphere, this would allow us to analyze the availability of the systems that do not possess the anomaly monitoring infrastructure like Nagios [13].

This approach would also be valuable beyond HPC – for instance, it could analyze the overall availability of an online retail platform by monitoring key performance indicators [11].

Besides the analysis of historical data, the proposed approach also offers a possibility to deploy anomaly detection models to the fields where *high-quality datasets are unavailable* - like online retail [7].

The proposed methodology thus offers exciting possibilities also far outside the field of high-performance computing for which it was initially developed.

Chapter 6



Anomaly prediction – GRAAFE

The work presented in this chapter builds upon the following contributions by Molan et al.:

- The Graph-Massivizer Approach Toward a European Sustainable Data Center Digital Twin [100]
- Graph neural networks for anomaly anticipation in HPC systems [96]
- GRAAFE: GRaph Anomaly Anticipation Framework for Exascale HPC systems [95]. The code associated with the paper is available on a GitLab repository¹.

The anomaly anticipation model for HPC systems has been recognized as a significant innovation within the Graph-Massivizer European research project² and has been featured on the **European Commission’s Innovation Radar** [129].

¹The source code, deployment guidelines, and additional experimental results for GRAAFE can be found at <https://gitlab.com/ecs-lab/GRAAFE>

²<https://graph-massivizer.eu/project/data-center-digital-twin/>

6.1 Introduction to anomaly anticipation

Driven by the need for greater computational performance, today's HPC systems are becoming increasingly more complex. The current generation of petascale systems consists of hundreds of compute nodes, while the HPC field begins to transition to pre-exascale systems consisting of thousands of compute nodes. In addition to large scale, HPC systems are based on heterogeneous design, usually consisting of four AMD or NVIDIA GPUs per compute node [53]. This increased size, combined with cutting-edge technologies, heterogeneous design, and integration densities, increase the system's complexity and cost, directly transposing into more complex and failure-prone management and maintenance of the HPC installations. Machine learning methodologies are introduced to support the HPC system administrators address this complex task.

In HPC systems, anomalies refer to occurrences deviating from regular system operations, jeopardizing its availability, or capacity to perform computational tasks. The transition to exascale and the associated increase in complexity and cost of the HPC systems has increased the importance of proactive management and anticipation (prediction) of anomalous operation of the HPC system or its parts. Jauk et al. [70] provide a comprehensive taxonomy of anomaly anticipation in practice. According to them, anomalies can be node level, referring to the availability of the entire compute node, or component level, such as the availability of a CPU or GPU. According to the taxonomy proposed by the survey, the paper focuses on predicting node-level anomalies. The survey paper [70], however, only focuses on anomaly prediction in HPC systems *based on log information*.

System log information, however, is not always available in the HPC systems. Depending on the implementation, log collection can have a significant performance impact [87]. Compared with log collection, *node telemetry* monitoring has a lower performance impact on the system (CPU utilization) [26]. Due to performance reasons and privacy and security concerns, some HPC centers, like CINECA, do not collect system logs. For such HPC centers, the only available information is node telemetry data [31]. Based on node telemetry data, several *anomaly detection* approaches have been introduced [30, 98]. To the best of our knowledge, however, the approach presented in this work is the first *node-level, telemetry-based anomaly prediction* approach deployed and evaluated on large production HPC systems.

Predicting anomalies is more complex than only detecting deviations from regular behaviour as previous and forthcoming system behaviour should be considered. To cope with this difficulty, we opted not to treat single nodes in isolation from each other (as done in x) but rather exploit the fact that computing nodes are physically and logically tightly coupled. In particular, nodes belonging to the same rack share a similar behaviour [74]; our idea is to exploit these proximity-based correlation to improve the overall accuracy.

In order to address the complexity of developing and deploying the topology-aware machine learning models to large production HPC systems, we introduce GRAAFE: graph anomaly anticipation framework for exascale HPC systems. GRAAFE exploits Graph Neural Networks (GNNs), as they are well-suited to learning tasks where there are complex dependencies that can be represented as graphs [153]. GNNs have never been applied to represent topological information in supercomputers but we claim that they are ideal to handle proximity graphs defined by nodes in the same HPC rack; additionally, since this approach deals with multiple nodes at the same time (entire racks), it is also more computationally efficient. We demonstrate this by conducting a large-scale experiment on the Marconi 100 supercomputer [44] hosted at CINECA. We also bridge the gap toward in-production data analytics by extending the CINECA’s ODA framework with an MLOps framework.

Summing up, the main contributions of the proposed GRAAFE framework are:

1. Development of a GNN to predict real, node-level anomalies that are shown to outperform other state-of-the-art approaches. With a 4-hour look-ahead window the GNN anticipates anomalies with an accuracy of 0.91 AUC; to the best of our knowledge, GRAAFE is the first work to demonstrate the feasibility of anomaly prediction in a production supercomputer.
2. Large-scale training and validation on 980 compute nodes followed by deployment at scale to evaluate its real-world effectiveness.
3. The pipeline can be deployed with only an additional 30% of CPU resources and less than 5% RAM usage increase. Of these overheads, the pipeline execution accounts for less than the 1%, making it scalable.
4. Ensure high reproducibility thanks to open-source code and definition of detailed computing requirements to implement the monitoring system and MLOps framework for Tier-0 supercomputer.

6.2 Related Works

6.2.1 Anomaly Detection & Prediction in HPC

One of the core objectives of ODA for HPC is anomaly detection and prediction, identifying aberrant or atypical patterns or behaviors from monitoring data [90] as soon as possible to minimize the system downtime. These deviations may comprise of any unusual occurrences concerning resource usage, performance variations, or network traffic flow. This work focuses on node-level anomaly analysis using node telemetry data. Several other approaches exist in the literature that either focus on component-level anomalies or on predicting anomalies based on log data [90, 70]. Approaches on log data deployed on a complex in-production large-scale HPC system predict node level anomalies with a future window of only 10 seconds [58].

Works with larger future windows, such as the work of Devesh Tiwari [84], focus on component failure prediction (disk failure specifically). Considerations on performance, security, and privacy prevent the deployment of system log collection in some HPC centers [87]. The only available monitoring data for the HPC systems in those centers, such as Marconi 100 in CINECA, is the node telemetry data [31].

Minimal attempts have been made to create a node-level anomaly anticipation system based on node telemetry data. To the best of our knowledge, the only node telemetry based approach that mentions anomaly anticipation is the work proposed as AdaHPC by Borghesi et al. [30]. The approach, however, fails to provide any estimate about the future window of the anomalies. Based on the taxonomy [70], the presented work is the first that addresses the question of *node-level anomaly prediction based on node telemetry data*.

Outside of the HPC domain, Carvalho et al. [39] propose random forest and artificial neural networks (dense neural networks) as the best-performing approaches for failure prediction based on a systematic literature review. Behera et al. [18] propose a Gradient Boosting tree for failure prediction. These methods inspire baseline per-node methods implemented in this work and are adapted for use as per-node anomaly predictors in HPC compute nodes.

6.2.2 GNNs and HPC

Most anomaly detection methods for HPC operate at a compute-node level, as they disregard the spatial structure of HPC systems – which instead has been suggested to be useful [74]. These structures can be represented as graph, thus suggesting the usage of graph-specialized ML models. Graph neural networks (GNNs) are ML models for graph-structured data [153]. Graph convolutional neural networks (GCNs) constitute a special type of GNN that relies on executing convolution operations. The convolutions combine insights from neighboring nodes to create powerful embeddings used in downstream tasks, such as node labeling or link prediction [101].

GNNs are very good at exploiting node proximity [154]. As supercomputers are organized in racks of neighbouring nodes, GNNs have great potential. For instance, GNNs can be trained using labeled data more efficiently than per-node supervised methods that tend to over-fit on the majority class. This advantage is especially noteworthy when dealing with extremely unbalanced classes typical of anomaly detection in HPC systems [101]. The use of GNN to improve anomaly detection and prediction performance has already partially been studied [41, 52]. For instance, Song et al. [123] use a couple of GNNs to identify abnormal performance fluctuations in cloud environments. The impact of proximity on the behavior of HPC nodes has been explored only in a limited way. Ghiasvand et al. [62] detect faults

using system logs that consider nodes' proximity in terms of hardware architecture, resource allocation and physical location.

6.3 Methodology for anomaly anticipation

In this work, we consider node-level anomalies. Anomaly prediction can be understood as a problem of predicting the transitions between states. We frame it as a binary classification problem [21]; the HPC system is either experiencing an **anomaly (state 1)** or **operating normally (state 0)**. Consequently, we have four possible state transitions:

1. From state 0 to state 0: this is the continuation of normal operation.
2. From state 0 to state 1: the *rising edge* event. This is the occurrence of an anomaly; we are primarily interested in this transition and want to predict it with ML methods.
3. From state 1 to state 1: continuation of the anomalous state.
4. From state 1 to state 0: the *falling edge* event. This is the resolution of the anomalous state and the return to the normal operation.

Transitions 3 and 4 are not the focus of this work as they are not the result of the malfunctioning of the HPC system but are connected to the response of the system administrators. Transition 3 persists until the fault is resolved, leading to transition 4. Thus, our approach are evaluated only on transitions 1 and 2.

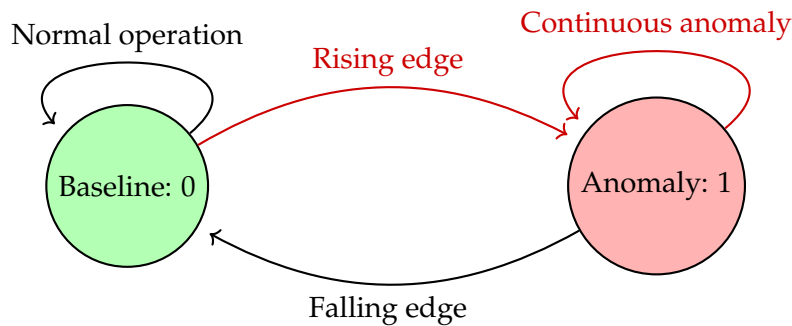


FIGURE 6.1: Finite state machine depicting the transitions between states 0 (normal operation) and state 1 (anomaly).

We embraced Graph convolutional network (GCN) based on:

1. The hypothesis that node physical proximity would benefit the anomaly prediction task.
2. Preliminary experiments demonstrating its potential for improved performance.

We encode the physical layout of compute nodes within racks as graphs. We develop distinct models for each rack; this ensures scalability within large supercomputers similarly to the approach proposed by Molan et al. [98]. The GCN model for each rack is depicted in Figure 6.2; on the left, we can see how racks are represented as graphs: vertices correspond to individual nodes connected vertically.

While conducting the initial study, we did not conduct a comprehensive exploration of hyperparameters but instead relied on prior knowledge and manual fine-tuning. Following an empirical investigation, it was found that the following GNN architecture delivers the best results:

- Graph convolution layer of shape (417×300)
- Graph convolution layer of shape (300×100)
- Graph convolution layer of shape (100×16)
- Dense layer of shape (16×16)
- Dense layer of shape (16×1)

The architecture presented in Figure 6.2, is based on the physical organization of compute nodes in racks. The GCN is trained on the supervised classification task. Labels are obtained by considering a future time window T ; labels with 1 indicate anomalies, and 0 indicate normal samples. For each node and at any point in time, a label of value 1 (anomaly) is assigned if the node encounters any anomaly in the future window T ; otherwise, the label are 0.

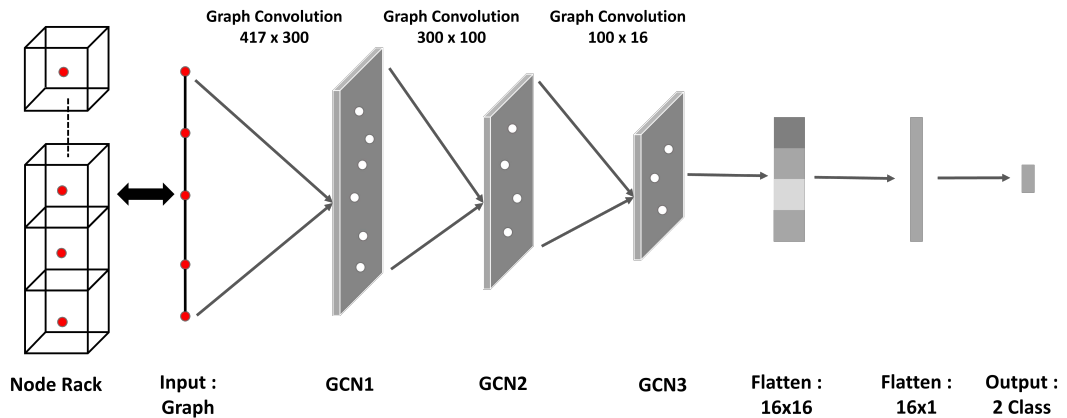


FIGURE 6.2: The structure of the GCN network exploits the organization of compute nodes in a rack.

6.4 Results of anomaly anticipation

Time Window	FW	GNN	DNN	GBT	RF	DT	MC
1h	4	0,91	0,64	0,63	0,61	0,51	0,5
1.5h	6	0,89	0,66	0,64	0,59	0,5	0,5
3h	12	0,84	0,65	0,63	0,59	0,5	0,5
6h	24	0,78	0,62	0,6	0,55	0,5	0,5
8h	32	0,75	0,59	0,58	0,55	0,5	0,5
16h	64	0,66	0,5	0,48	0,49	0,49	0,5
24h	96	0,62	0,55	0,51	0,58	0,51	0,5
48h	192	0,55	0,47	0,48	0,52	0,5	0,5
72h	288	0,53	0,52	0,51	0,52	0,49	0,5

TABLE 6.1: AUC scores of prediction methods. The GNN outperform all other methods across all future windows (FW).

6.4.1 Experimental setting

The Marconi 100 HPC system located in CINECA was utilized to conduct an experimental evaluation. A solitary compute node of Marconi 100, which included 32 IBM POWER9 cores, a RAM capacity of 256 GB, and four NVIDIA V100 GPUs with a memory size of 16 GB, was employed for the training the GNNs. The dataset comprises 31 months wherein two computer racks of the Marconi 100 system were subjected to observation. To establish an efficient model, 80% of the data mentioned above was designated as the train set, while historically, the remaining 20% was allocated for testing purposes.

The proposed approach was tested against all known per-node anomaly anticipation approaches to demonstrate the necessity of the GNN. The label creation and the train/test split were the same across all examined models. Specifically, we have chosen the deep per-node neural network (DNN), the gradient boosting trees (GBT), random forest (RF), and decision trees (DT) as comparative models. The Markov Chain (MC) serves as a trivial predictor baseline, representing a simple approach that consistently predicts the probability of failures as equal to the overall failure rate observed in the dataset, regardless of the input data. The hyperparameters of all the models and the accompanying code for the experimental evaluation are described and included in the Artifact Description. The experimental evaluation is also performed on an open and publicly accessible dataset. This ensures transparency and reproducibility of the experimental setup for further research in this field.

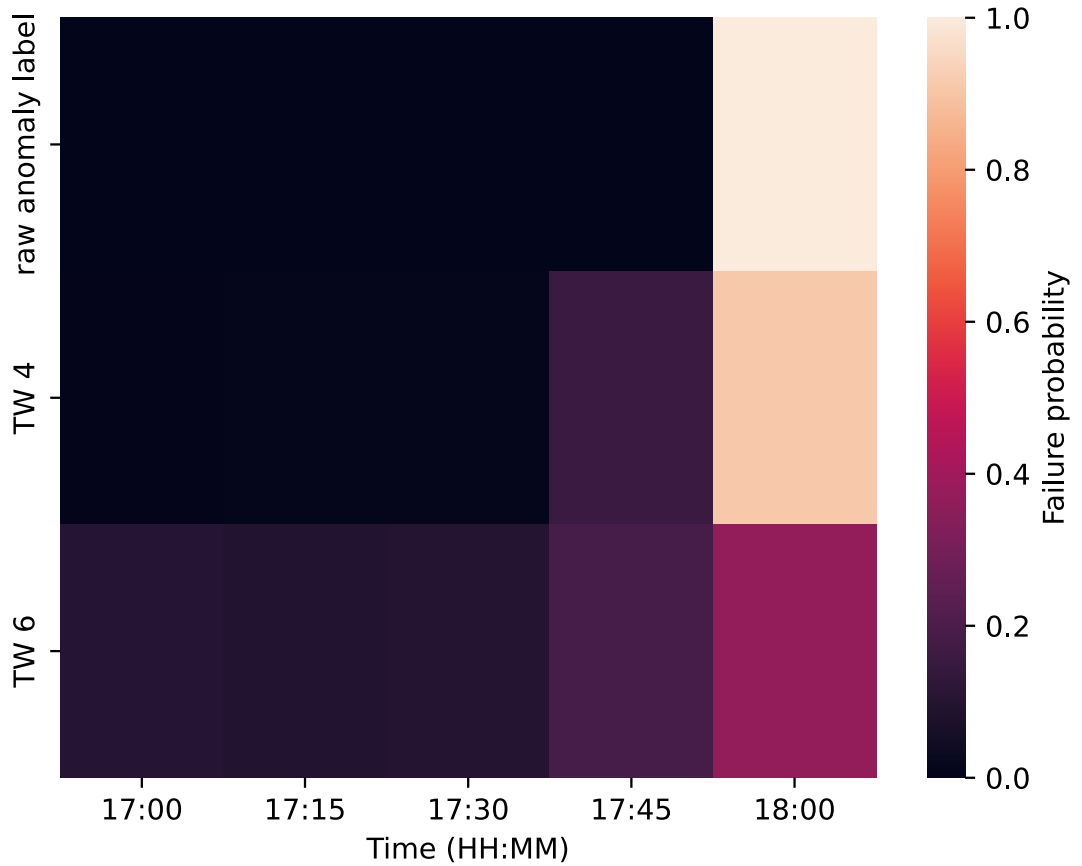


FIGURE 6.3: Period of 5 timestamps or 75 minutes, starting at 17:00 and ending at 18:15.

6.4.2 Anomaly Prediction Model Performance

In line with other works in Operational Data Analytics and, specifically, anomaly detection [98], the area under the ROC curve (AUC) is selected as the primary evaluation metric for the experimental analysis. The predictive models output the probability of the anomaly (instead of just predicting class 0 or 1). This helps system operators. Furthermore, examining the whole ROC curve allows for a more general analysis that does not favor only the performance of the classifiers at a specific threshold. AUC is also chosen as a primary performance metric as it protects from pitfalls shared by other relatively more common metrics (precision, recall, f1) [76].

Across all Future Windows (FW), the GNN significantly outperforms all other approaches that operate on per-node data. Table 6.1 demonstrates higher AUC values and thus indicates better performance in detecting anomalies. The performance of all models decreases with future observation windows showing the increasing difficulty in detecting anomalies further away in the future. The comparative advantage of the GNN model also decreases with larger future windows. At the last observed window 288 (72 hours in the future), the performance differences between various models become almost negligible. All models are slightly better than the

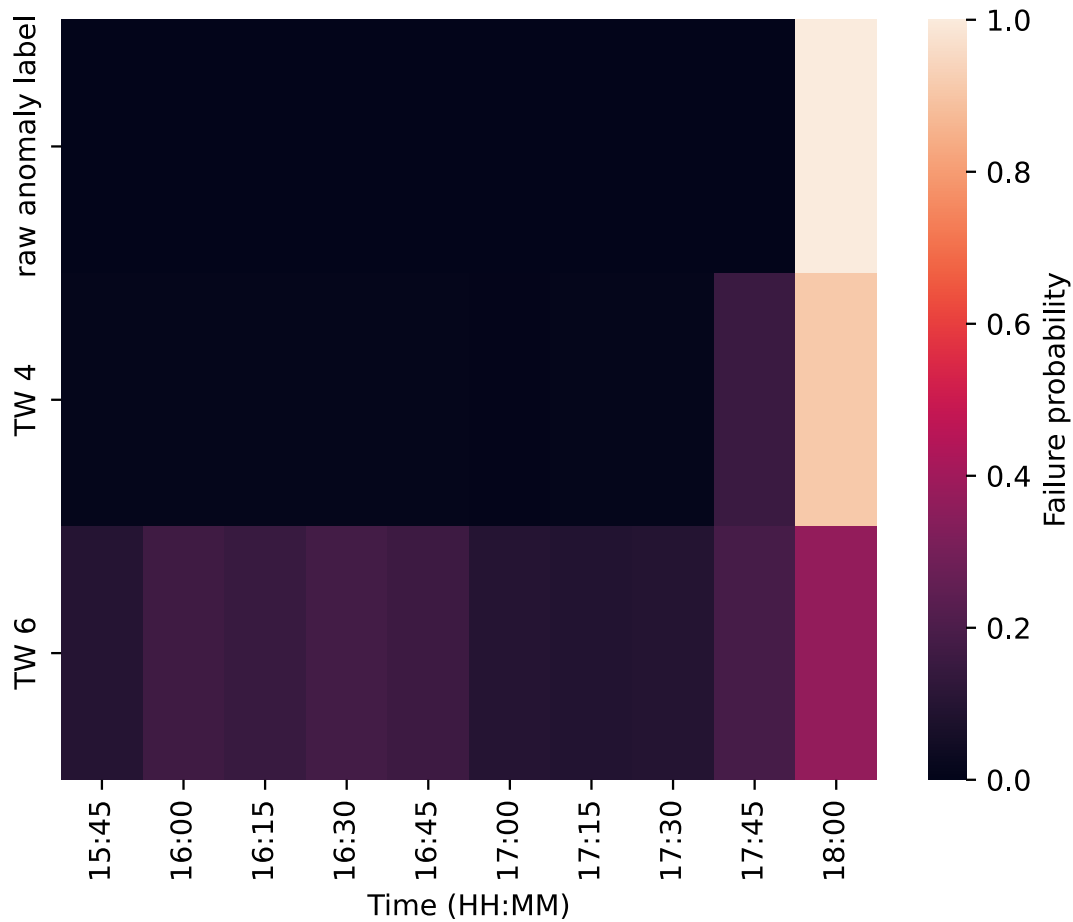


FIGURE 6.4: Period of 10 timestamps or 2.5 hours.

random choice model, which has a 0.5 AUC score.

The performance of the GNN model is strong up to a future window of 64 (16 hours in the future). Then, the performance of all other models (including the per-node NN) dips to around the performance of the random choice. In other words, predicting up to 16 hours in the future is only possible with GNNs. The performance of all other per-node models is within expectation. The best performing is the per-node DNN, followed by GBT, RF, and, lastly, DT. The experimental analysis shows that the GNN model is far superior for predicting anomalies in HPC systems.

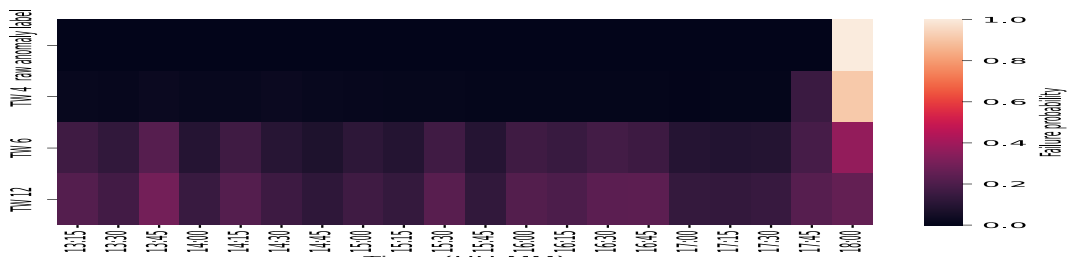


FIGURE 6.5: Period of 20 timestamps or 5 hours.

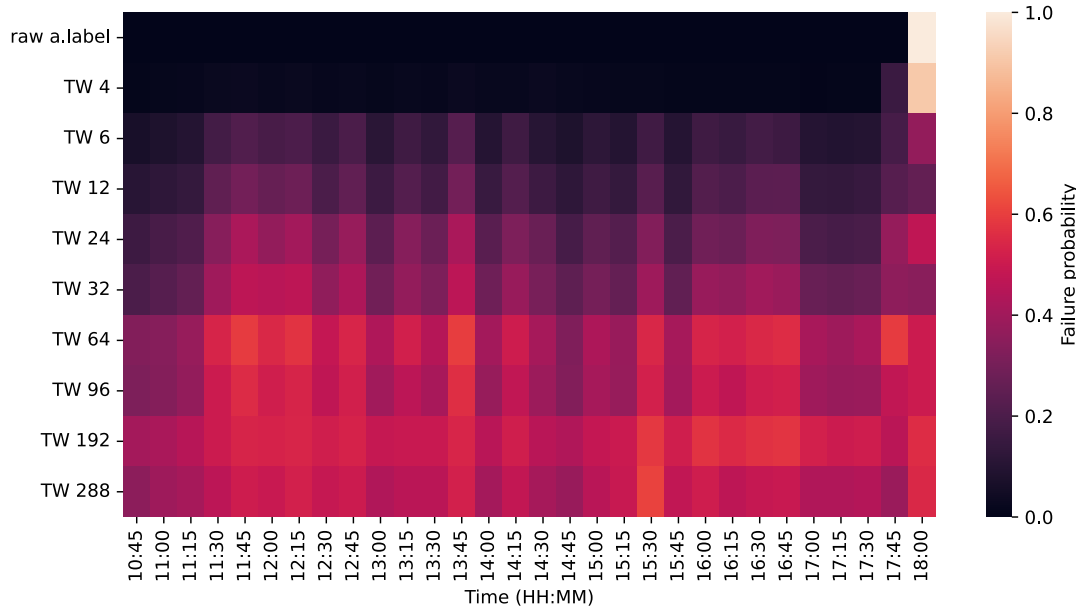


FIGURE 6.6: Period of 30 timestamps or 7.5 hours.

The main points of analysis of the experimental results can be summarised as:

- **Superiority of GNN:** The GNN model exhibits a clear superiority over other models in anomaly detection across all Future Windows (FW). This is evident from the higher area under the curve (AUC) values presented in Table 6.1, a robust indicator of the model's ability to distinguish between normal and abnormal states.
- **Performance Degradation Over Time:** A notable trend observed is the gradual decline in the performance of all models as the future observation window increases. This suggests a fundamental challenge in predicting anomalies further into the future, likely due to the increasing uncertainty and variability in data over extended time frames.
- **Comparative Advantage of GNN Diminishes with Larger Windows:** While the GNN maintains its lead, its comparative advantage diminishes in larger future windows. By the last observed window of 288 (equivalent to 72 hours ahead), the models' performance differences shrink considerably.
- **GNN's Strong Performance in Shorter Time Frames:** The GNN model's strength is particularly pronounced up to a future window of 64 (16 hours ahead). Beyond this point, other models, including the per-node Neural Network (NN), decline to levels comparable to the random choice model.
- **Performance of Other Per-node Models:** Among the other models tested, the per-node Deep Neural Network (DNN) shows the best performance, followed by Gradient Boosting Trees (GBT), Random Forest (RF), and Decision Trees (DT).

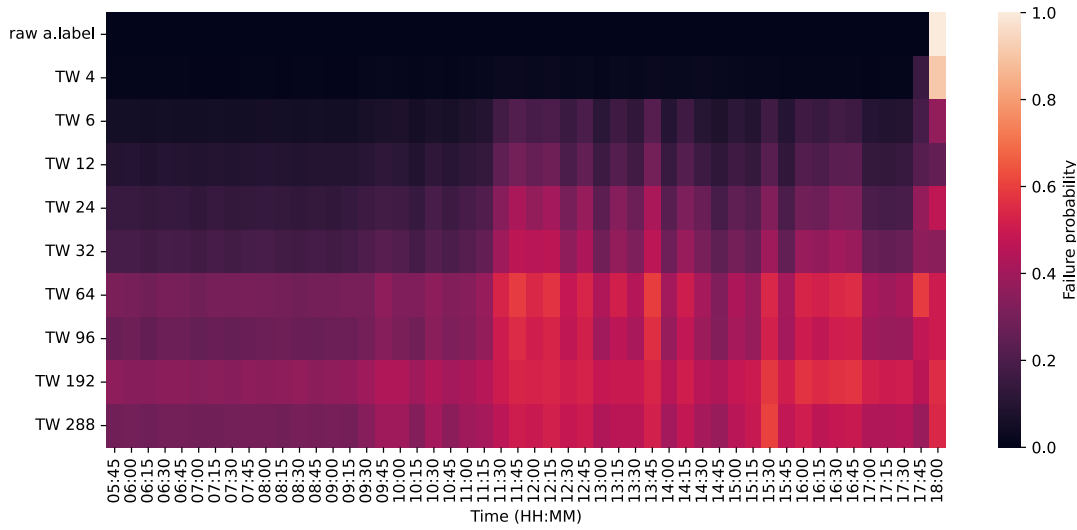


FIGURE 6.7: Period of 50 timestamps or 12.5 hours.

- **Overall Implications:** The experimental results underscore the GNN's effectiveness in the specific application of anomaly detection in HPC systems, especially for short to medium-term predictions. The diminishing returns in longer prediction windows suggest a need for continued research into models that can maintain high accuracy over extended future windows.
- **Future Research Directions:** This analysis suggests potential areas for future research, such as exploring hybrid models that combine the strengths of GNN with other models or developing new techniques to enhance long-term prediction accuracy in HPC systems.

6.4.3 Anomaly prediction model probability calibration

The importance of calibration in the GNN anomaly anticipation model cannot be overstated, as its primary output is predicting the probability of an anomaly occurring in the next time window. Proper calibration is essential to ensure these probability predictions are aligned accurately with the actual frequencies of anomalies observed in the test set. This alignment is critical not only for the model's reliability but also for the trust system administrators place in the model's predictions.

A key measure used for evaluating the calibration of probability predictions in binary classification scenarios, like that of the GNN model, is the **Brier score** [34]. This score assesses the mean squared difference between the predicted probabilities and the actual outcomes. A score of zero in this metric indicates perfect calibration, signifying that the predicted probabilities precisely match the observed frequencies. It's crucial to understand that while the Brier score is a robust measure of calibration, it does not directly reflect the classifier's performance in terms of its accuracy in predicting outcomes.

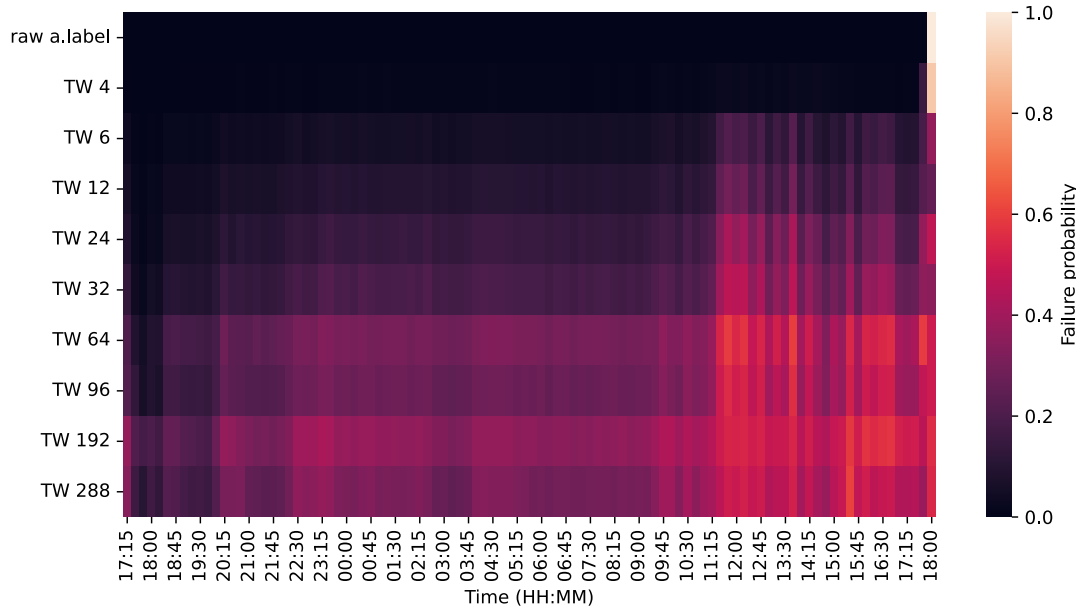


FIGURE 6.8: Period of 100 timestamps or 25 hours.

The calibration results of the GNN model, as indicated in Table 6.2, reveal that the model achieves a Brier score of less than 0.1 across all future windows. This indicates a high level of calibration accuracy. Particularly noteworthy is the model's performance in the shorter future windows of 4 and 6, which correspond to one hour and one and a half hours, respectively. The Brier score is below 0.01, highlighting the model's exceptional calibration in these time frames.

However, it is observed that the calibration score increases with larger future windows, which aligns with the expected trend considering the increasing difficulty of making accurate predictions further into the future. As the prediction window extends, the model encounters greater uncertainty and variability, which naturally poses a challenge to maintaining high calibration accuracy.

The implications of these findings are significant for applying the GNN model in system administration. The high level of calibration accuracy, especially in shorter prediction windows, means that the model's probability outputs can be reliably communicated directly to system administrators. This aspect is further discussed in Section 6.4.4. Despite these positive results, continuous monitoring and evaluation of the model's calibration over time is advisable, particularly as new data emerges or operational contexts evolve. Future research might also explore the potential benefits of additional calibration techniques, especially for enhancing the model's performance in longer future windows.

Time Window	FW	Avg. Brier Score
1h	4	0.007674
1.5h	6	0.009031
3h	12	0.011626
6h	24	0.016599
8h	32	0.019567
16h	64	0.030524
24h	96	0.039851
48h	192	0.062677
72h	288	0.082576

TABLE 6.2: GNN anomaly predictor is well-calibrated for all future windows (FW), as evidenced by the low Brier scores.

6.4.4 Visualization of anomaly anticipation

GNNs trained on different future windows are used concurrently for each node. The AUC scores in Table 6.1 show that the prediction methods with the shortest observation window are the most reliable. The longer observation window methods (e.g., 72 hours) give system administrators an early warning which can then be confirmed via shorter time windows (e.g., 1-hour and 1.5-hour). To illustrate this, we have examined a single raising edge event (transition from state 0 to state 1). This event is taken from a training set of node 240, located in compute rack 12. The anomaly occurred on 7/5/2022 at 18:15. The event with different lengths of the preceding periods is plotted in Figure 6.8. We have plotted the (normalized) probability for class 1 from different GNN classifiers.

Examining the classifier designed for the shortest prediction window, precisely a future window of 4 timesteps, equivalent to predicting an anomaly one hour ahead, we observe that this classifier successfully anticipates the anomaly. Despite this short anticipation window, it generates a distinct anomaly signal before the event’s escalation.

A consistent pattern of anomaly anticipation is observed as the focus shifts to classifiers designed for longer observation windows, spanning 2.5 to 5 hours. These classifiers, trained to detect anomalies over longer windows, also correctly predict the impending anomaly. Notably, the classifier trained for a one-hour advance prediction (future window FW 4) does not signal anomalies in these extended windows. This aligns with expectations since its design and training are tailored for a shorter prediction horizon.

When examining classifiers tasked with even larger time windows, ranging from 7.5 to 25 hours, all exhibit the anticipated increase in anomaly probability as the time

of the actual anomaly approaches. This trend is visually represented in Figure 6.8, where the probability density intensifies as the timeline converges on the anomaly event.

A critical observation across these varying time windows is the emergence of a *rising wave* pattern in the probability predictions as time approaches the anomaly. This pattern is more pronounced and clear-cut in classifiers with shorter prediction windows. The clarity of this pattern correlates with the AUC scores, being more distinct and reliable in classifiers designed for shorter windows. Conversely, as the prediction window widens, the anomaly pattern becomes increasingly noisy and less defined. In classifiers with very large time windows, the anomaly pattern tends to lose its distinctiveness, resulting in a near-constant probability output for an anomaly. This indicates a diminishing efficacy and reliability in the anomaly predictions as the prediction window extends.

These findings highlight a key challenge in anomaly prediction: maintaining accuracy and clarity in the predictive pattern as the prediction window increases. While short-window classifiers offer precise and actionable predictions, the effectiveness of long-window classifiers is hampered by increasing noise and uncertainty. This insight is crucial for the practical application of these classifiers in real-world scenarios, where the choice of prediction window must balance the need for warning against the reliability of the predictions.

In summary, the extended analysis underscores a consistent pattern of rising probability as the time nears an anomaly, with the clarity and reliability of this pattern being inversely proportional to the length of the prediction window. This observation has significant implications for the design and application of anomaly prediction classifiers, especially in their operational environment and the specific requirements of advance warning versus prediction accuracy.

6.4.5 Financial impact of anomaly anticipation

Deployment of the anomaly prediction model in real-life operations consists of estimating the positive aspects and benefits of the deployment against the potential negative aspects and costs. To do this, we have developed a model, which is included in the code repository, that models the benefits of the model against the costs associated with mispredictions. We are combining the model with some assumptions about the operational aspect of a typical supercomputer and the additional results of our predictive models, as described in the additional results in the code repository. We are using the optimal threshold for the classifier and the true positive and false positive rates, as reported in the additional results. The general equation of the cost-benefit model is:

$$B = C_{FN} - C_{FP}$$

where:

B represents the overall benefit,

C_{FN} denotes the cost associated with false negatives,

C_{FP} denotes the cost associated with false positives.

Compared to the case where no predictive model is deployed, the deployment of the GRAAFE framework results in no additional costs of false negatives. For this reason, the cost of false negatives is modeled as zero in our model. The cost of false positives, however, is associated with the time that the system administrators waste analyzing false positive signals. We are modeling it as an unnecessary action cost. Unnecessary action cost is calculated as:

$$C_{UA} = P_{FP} \times T_A \times W_{SA}$$

where:

C_{UA} is the unnecessary action cost,

P_{FP} is the probability of a false positive,

T_A is the analysis duration,

W_{SA} is the hourly pay of the system administrator.

The unnecessary action cost increases for larger future windows as the false positive rate increases and the overall anticipation accuracy decreases.

Hours	Optimistic	Conservative	Pessimistic
0	0.0	0.0	0.0
1	0.7	0.3	0.0
3	0.8	0.6	0.0
6	-	-	0.3
72	0.8	0.8	0.8

TABLE 6.3: Three scenarios are defined based on the probability that the system administrators can prevent an anomaly if given a signal within a future window (in hours).

The benefit of the model development is estimated as the expected value of the deployment benefit:

$$B = P_{DF}(FW) \times P_{PF}(FW) \times C_F$$

where:

B is the benefit,

$P_{DF}(FW)$ is the probability to detect failure, as a function of FW ,

$P_{PF}(FW)$ is the probability to prevent failure, as a function of FW ,

C_F is the cost of failure,

FW is the future window.

The probability of *detecting failure* is a characteristic of the predictive model, and it equals the true positive rate. For the analysis presented in this section, we are using the true positive rate of the per-rack GNN model. The cost of failure is related to the opportunity cost of the HPC system downtime. Our analysis estimates it to be a fraction of the overall cost of the HPC system.

The probability of *preventing failure* is a characteristic of the organizational specifics of each data center and its system administration team. Since we do not have accurate organizational information for the Marconi 100 supercomputer, we have prepared three scenarios demonstrating different system administration teams. Based on the warning period (in hours), the system administrator teams prevent the anomaly with different probabilities as depicted in Table 6.3.

Optimistic scenario depicts a very effective system administration team that can prevent 70% of anomalies with a warning window of one hour and 80% with a warning period of three or more hours.

Conservative scenario depicts a conservative estimation of the system administration team's effectiveness. System administrators can only prevent 30% of the anomalies with a prediction window of one hour.

Pessimistic scenario is an overly pessimistic option where system administrators cannot prevent anomalies if they have a warning period of less than six hours.

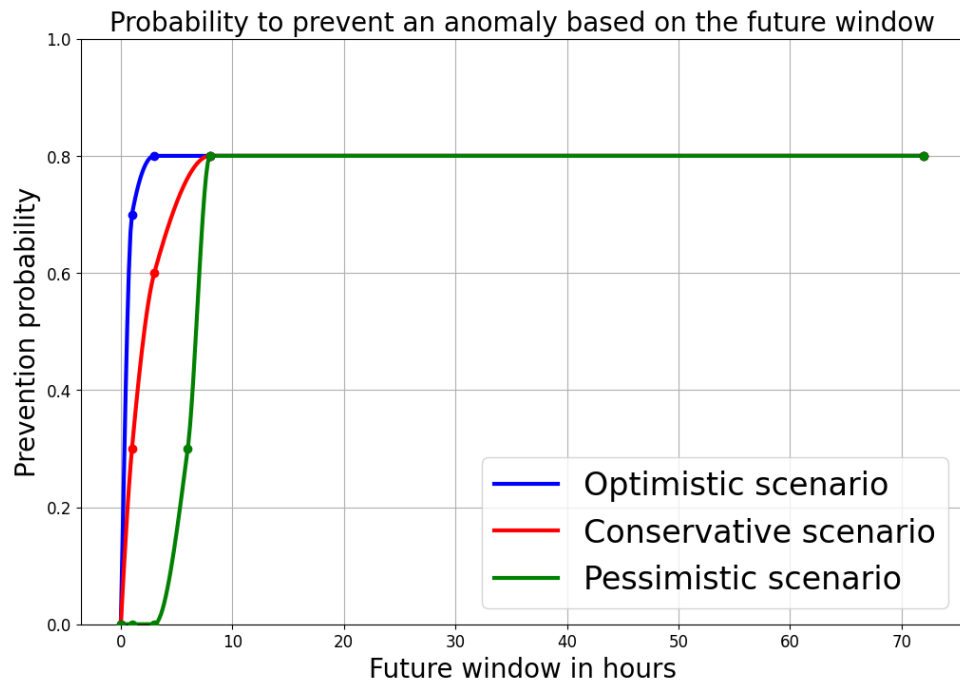
Based on the important points of each scenario, a smooth (double differentiable) function is fitted over them. This gives us the probabilities for every other future window. The function is based on the assumption of being monotonic positive as the prevention probability cannot decrease. Based on three scenarios, we get three different anomaly prevention probability functions as depicted in Figure 6.9a.

Given the anomaly prevention probability function, we can estimate the cost-benefit of deploying the model in each scenario. The cost-benefit, measured in the percentage of the overall system cost, is depicted in Figure 6.9b. In the optimistic scenario, the maximum benefit is achieved for the one-hour future prediction window. For the conservative scenario, the maximum is achieved for the prediction window of 3 hours. Interestingly, for the conservative scenario - the cost-benefit is negative for future windows up to 3 hours. This is because the system administrators cannot prevent the anomalies but would still have to react to the false positive

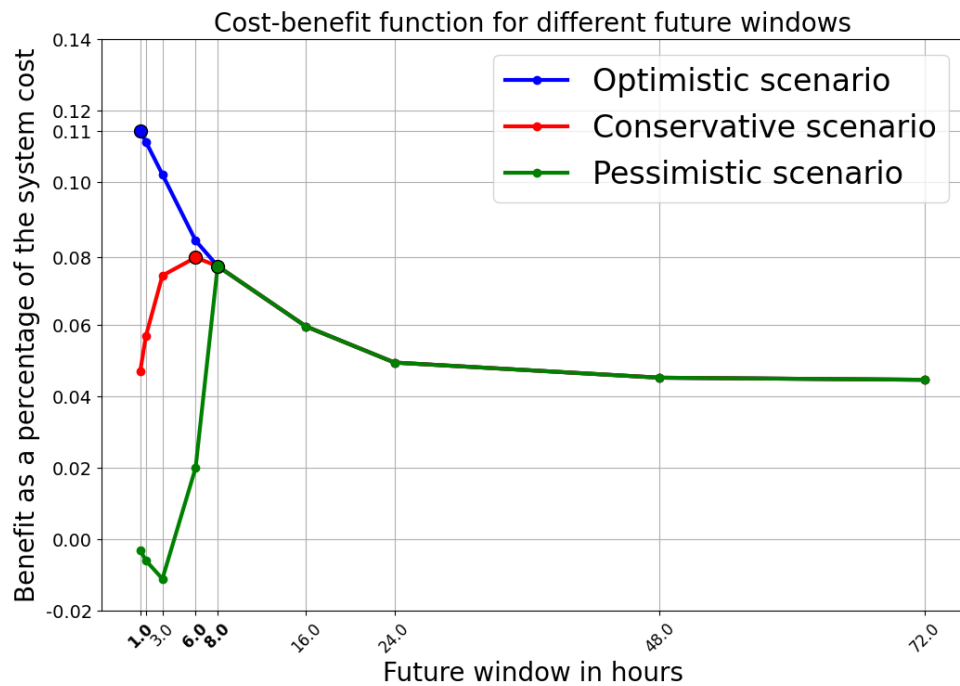
signal (and incur the associated costs). As seen in Figure 6.9, the model’s benefit and the optimal deployment strategy (size of the future window) depend strongly on the organizational characteristics of each data center.

Estimating the organizational characteristics of the individual data center is beyond the scope of this work. For this reason, we have prepared a computational model included in the code repository³ that takes the operational parameters as input and outputs the cost-benefit function for the defined specific scenario. This allows potential adopters of our framework to decide on the optimal deployment strategy based on their organizational reality and needs.

³https://gitlab.com/ecs-lab/GRAAFE/-/blob/main/Cost_benefit.ipynb



(A) The prevention probability function models the probability that the system administrators will be able to prevent the anomaly if given the anomaly signal in a given future window.



(B) The cost-benefit function models the net benefit (benefit minus the cost) of the model adoption for different future windows, expressed as the percentage of the total HPC system cost.

FIGURE 6.9: Depending on the scenario and the associated probability of preventing the anomaly, given a specific warning window, the projected benefit is achieved by deploying the predictive system at different future windows.

6.5 Anomaly prediction as part of a GRAAFE framework

An anomaly prediction system has been developed and deployed as an integral part of the GRAAFE framework. GRAAFE, which stands for *Graph Anomaly Anticipation Framework for Exascale HPC Systems*, is a comprehensive operational data analytics framework that combines data acquisition, anomaly prediction, and visualization. The data acquisition and visualization functionalities are seamlessly integrated with the EXAMON monitoring tool [26].

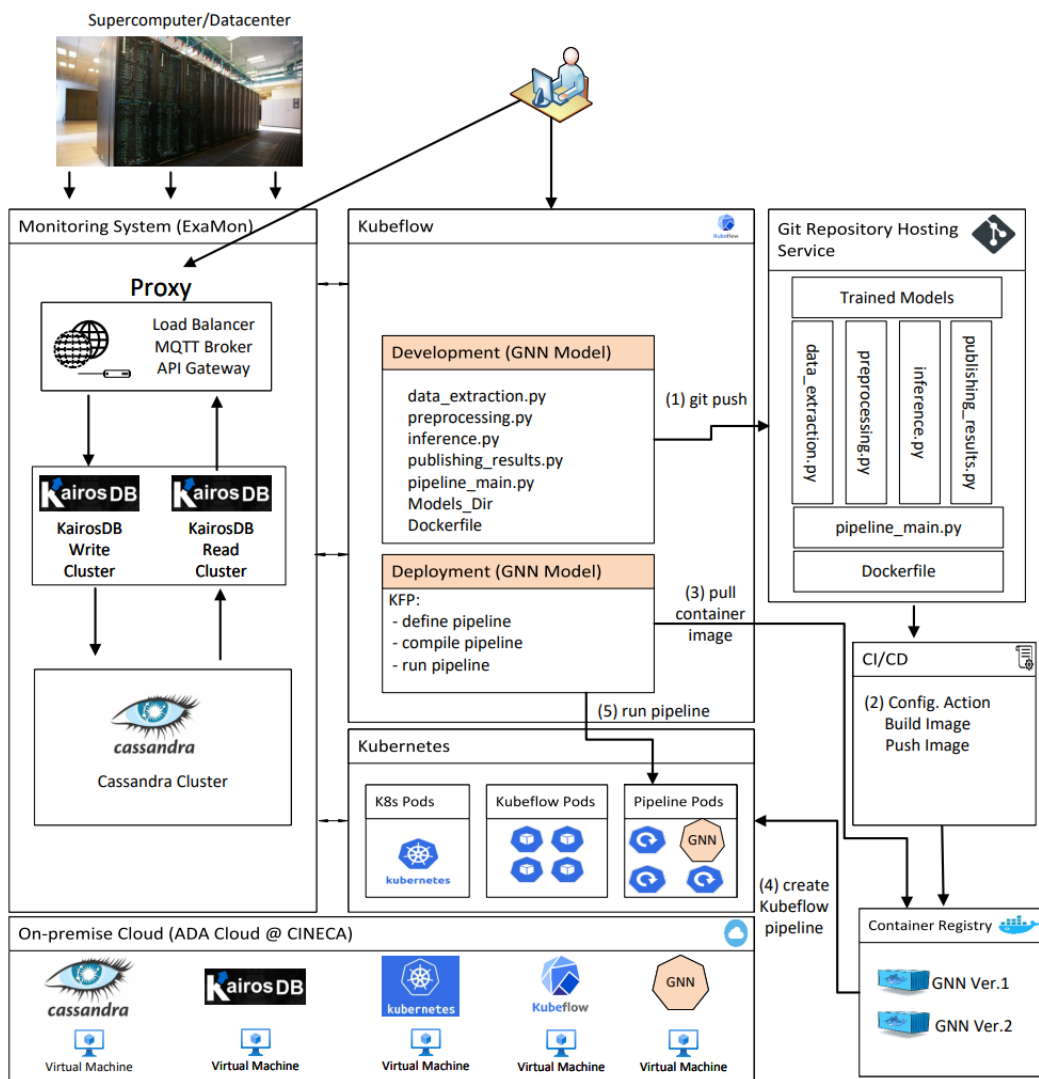


FIGURE 6.10: The software architecture of the GRAAFE framework, built for the deployment of an anomaly prediction model for HPC systems.

The entire data pipeline, depicted in Figure 6.10, is built on Kubernetes/Kubeflow virtualization technology, enabling real-time anomaly prediction for all Marconi100 nodes with updates every 120 seconds. This system achieves a remarkable balance between computational efficiency and model accuracy, requiring only 30%

additional CPU resources and less than 5% more RAM compared to monitoring alone. This efficiency is attributed to the use of graph neural networks for anomaly prediction, which offers an optimal trade-off between performance and accuracy.

The successful deployment of this predictive model within a production-ready anomaly prediction framework is the result of collaborative efforts with my co-authors, culminating in a publication in the *Future Generation Computer Systems* journal [95].

6.6 Conclusion of anomaly anticipation

In this section, we proposed the first anomaly prediction approach for HPC systems explicitly exploiting the underlying proximity structure of computing nodes. The graph structure significantly influences the performance of anomaly anticipation models for supercomputers. Our GNN approach has enabled us to surpass all previously known ML methods in this domain; in particular, we thoroughly demonstrate how we can not only detect failures, as already done in previous work, but predict such failures with several hours of anticipation.

The MLOps framework has been thoroughly evaluated in various deployment configurations to measure the overhead introduced by the Kubernetes pipeline. In its current setup, the impact on the supercomputer itself is negligible, as the MLOps operates on the same cloud infrastructure, located within the HPC facility hosting the supercomputer, where the monitoring infrastructure is also deployed. We have also explored different deployment setups, highlighting how different solutions can be adopted, granting greater flexibility while satisfying real-time requirements. Moreover, the additionally incurred computational overhead with regards to the monitoring infrastructure, which we can assume to be present in most supercomputers nowadays, is minimal.

Chapter 7

Looking into the Future

The most significant innovation in recent years in machine learning—and arguably in the history of machine learning—has been the development of large models with generative capabilities. The most famous and notorious of these models are large language models (LLMs).

The first has been the Generative Pre-trained Transformer (GPT) developed by OpenAI in 2018 and released to the public in 2020 [35]. Other models with generative capabilities include generative image models or text-to-image models, such as Stable Diffusion [117]. Another recent development in large generative models is the emergence of multimodal models—models that combine the understanding of textual data and images. Examples of this include OpenAI’s large multimodal models, such as GPT4 [111] or LLaMa [132] from META.

7.1 Power and Limitations of LLMs

The development of these large models has challenged the previous belief, that only specialized models trained on specific tasks, like text generation or semantic segmentation in images could achieve high performance. It’s understood that large, general models trained on a broad range of tasks can perform well across various applications. However, despite these models’ rapid deployment and development and the significant market interest and investment they have attracted, the integration of large models—or foundational models—has not yet fully extended to other types of data, specifically multivariate time series data.

This type of data, often collected by monitoring systems or IoT devices, still requires custom-made models, such as those discussed in this work. Given the trends and the success of large models, a key question arises: How can we intelligently integrate foundational and large models with custom-developed models for specific applications?

7.1.1 Limitations to Perform Symbolic Reasoning

One limitation of foundational models is their ability to perform symbolic reasoning, such as arithmetic and mathematics, where they operate with numbers rather

than just tokens or strings. Some attempts, like TimeGPT [61], have been made to translate time series prediction tasks into sequence prediction tasks to leverage the capabilities of LLMs. However, these efforts have yielded mixed results and are not yet applicable to the full range of predictive tasks required for monitoring data.

These tasks often need more complex and specialized models beyond the simple predictive tasks that LLMs can handle. For example, advanced data analysis models require comprehensive clustering techniques, while anomaly detection models need sophisticated supervised learning approaches, as discussed in the relevant chapters of this work. Tasks such as time series prediction may require a combination of time series forecasting and graph-based predictions as it is developed and presented in this thesis.

Even if research advances significantly in treating time series as tokens for sequence prediction, more complex tasks, such as those discussed in this work, will likely remain outside the scope of what foundational or LLMs can effectively handle. Most of the reasoning capabilities of leading LLMs, such as GPT-4, come from being trained on large corpora of code data. When asked to perform simple arithmetic operations, these models typically generate small code snippets that execute the calculations rather than relying on the model's intrinsic next-token prediction logic. This behavior, known as "tool use," highlights an important aspect of current research: integrating LLMs into broader software infrastructures, enabling them to retrieve live data from different streams and analyze it using pre-written functions or libraries.

Currently, a significant portion of practical large language model applications involves integrating these models into existing software ecosystems. An essential aspect of this integration is called Retrieval-Augmented Generation (RAG).

Instead of constantly updating LLMs with current data—thus avoiding the risk of hallucinations—mission-critical documents are encoded in a separate database, often a vector database. Based on a query, the most relevant documents are retrieved and used as context for the language model during answer generation. This approach allows businesses to use specialized LLMs without needing expensive re-training or fine-tuning. However, a limitation of current RAG systems is that they mainly focus on textual documents, leaving other data modalities, such as real-time monitoring and time series data, less integrated.

The practical applications of LLMs in today's industry are primarily limited to integration with RAG systems and vector databases. Most documents in these vector databases are already encoded in a way that LLMs or multimodal models can ingest. This typically means that these are textual documents, potentially with images. A significant drawback of these systems is their limited ability to integrate with other data modalities, such as real-time monitoring or time series data. This limitation underscores the need to connect LLMs with specific data sources to tackle particular organizational problems.

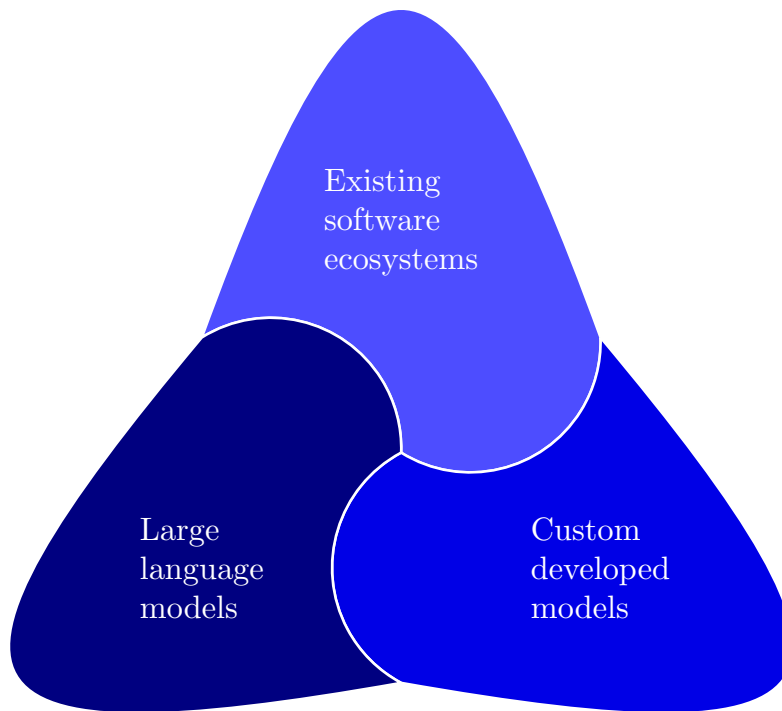


FIGURE 7.1: Structure of the application LLMs

7.1.2 Limitation in Ability to Use Tools

Another critical aspect of integrating LLMs into current business solutions is their ability to use tools. While LLMs are trained to predict the next token in a sequence, they do not inherently possess the capability to perform complex data analysis or mathematical operations.

To enable such capabilities, these models must be integrated with specific functions. The first attempts at such integration involve "tool use", where the language model is instructed to call specific functions. Once capable of calling these functions, the model can integrate with existing libraries, such as those used for data analysis. This, in turn, allows LLMs to perform more complex tasks and offers greater degrees of automation within a business context.

These developments — namely, RAG and tool use — indicate that the future adoption of LLMs in real-world applications will likely make them part of a broader software infrastructure. Specifically, I believe that the future role of LLMs will be as a highly advanced and capable integration layer. Due to their generalizability and flexibility, they can connect various parts of the software ecosystem efficiently.

For example, they could connect with domain knowledge stored as documents, existing code libraries, data analysis tools, and machine learning model repositories.

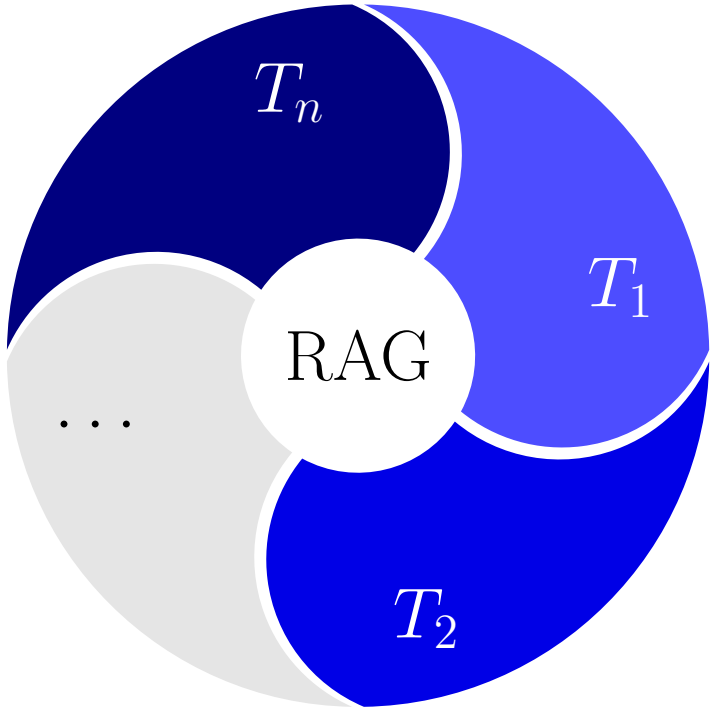


FIGURE 7.2: Adoption of LLMs in real world applications: Large language model (RAG) including broader software infrastructure using tools ($T_1, T_2 \dots, T_n$).

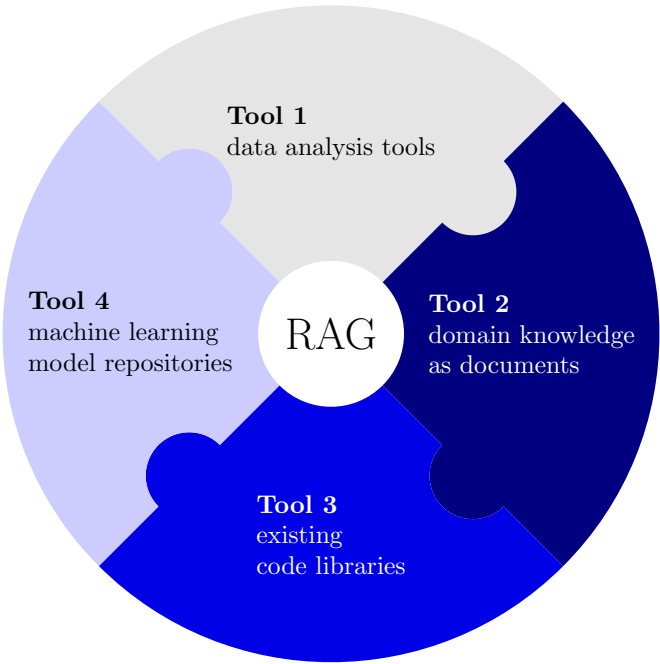


FIGURE 7.3: LLMs and tools

In the context of the work presented in this thesis and within the field of operational data analytics, the role of LLMs will be to integrate and present the developed models in a more useful and relevant way to system administration.

Example 7.1.1 *A system administrator could ask a language model agent:*

What is the probability of node #13 failing due to hardware issues in the next 10 hours?

This answer cannot be generated by the language model alone nor by simply retrieving information from a vector database.

Instead, it requires live telemetry data, which must be analyzed using a trained anomaly prediction model, such as the graph model described in this work, before the results can be communicated to the end user.

More advanced scenarios could further enhance the language model's ability to analyze data from multiple sources autonomously. Even in its current form, this approach would provide invaluable assistance to system administrators, making managing a model framework a more streamlined and efficient experience.

Indeed, I believe that until truly foundational models for time series data emerge—if they ever do—the future of operational data analytics and real-time modeling in the context of AI will involve a combination of LLMs and custom-developed models. This approach leverages the strengths of each.

- Custom Models.
- LLMs.

The predictive power of custom models, such as the work on graph-based anomaly detection presented here, remains unparalleled. It makes sense to use these highly specialized, custom-trained models to achieve the best results.

However, the strength of LLMs lies not in making predictions themselves but in managing and integrating models within different parts of a software ecosystem.

Thus, until, or perhaps even after, foundational time series models are developed, operational data analytics will benefit from

LLMs serving as an integration layer.

This integration will likely require modifications to the traditional three-layer data analytics approach, as Netti and others described [110]. Their model outlines three layers of operational data analytics:

1. data acquisition,
2. data analysis, and
3. data visualization.

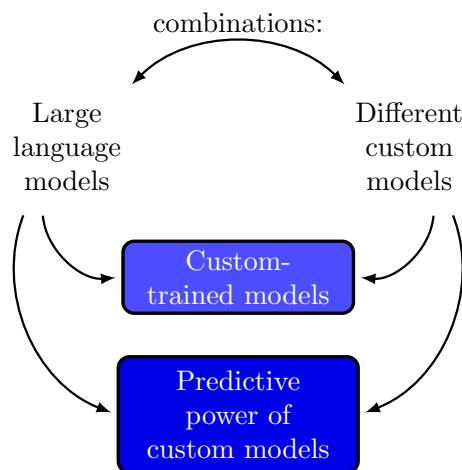


FIGURE 7.4: Future of operation data analytics

We believe an additional layer will emerge between data analysis and visualization, integrating the two.

This new layer is where the large language model agent will reside. The language model agent will manage the model framework in the data analysis layer and potentially modify the data visualization layer on demand. However, it will not replace either of these layers but rather extend their capabilities.

We live in the age of large models and generative AI, which represent tremendous scientific accomplishments. These developments bring new capabilities and enthusiasm to the field of operational data analytics.

However, they do not fundamentally change the core challenges of the field. The state-of-the-art models still follow research directions similar to those before the advent of large foundational models. What has changed is the ease with which we can integrate these models into existing solutions, generate actionable insights for end-users and stakeholders, and manage model toolkits more efficiently.

Chapter 8

Looking beyond HPC systems

The topic of this thesis and the common theme of all the models presented in this work is the use of machine learning applications in the context of operational data analytics and the monitoring and management of high-performance computing systems. While the management of high-performance computing systems is essential as they represent a significant societal and economic resource, the methodologies and the study of real-time monitoring offer utility far beyond the original scope of high-performance computing systems.

8.1 Operational data analytics

Operational data analytics emerged as a discipline aimed at supporting the work of HPC system administrators in automatically analyzing vast quantities of real-time telemetry data. As described in Chapter 4, it first emerged as a combination of data acquisition and visualization systems. Then, driven by the need for more advanced and automatic data analysis, data-driven methodologies, such as the models presented in this work, have been gradually introduced.

The gradual development of operational data analytics in the HPC sector, marked by the introduction of machine learning methodologies, serves as a model for other 4.0 industries. This evolution, as outlined in this work, starts from the most general (data analytics) tasks that require the least high quality of data and progresses to the most specific tasks (anomaly prediction) that demand the highest data quality. *Data Exploration Model DEM* is applicable to any domain where real-time data processing is essential.

Operational data analytics as a paradigm combining the three pillars of data acquisition, data processing, and data visualization applies beyond the domain of Industry 4.0 to every field where *real time* data analytics is essential. Such applications, especially in conjunction with anomaly detection, are mainly present in the finance industry and banking (fraud detection), online retail (downtime detection), and even customer management systems (personalized campaigns).

Beyond the machine learning methodologies themselves, the lesson that applies to these other domains is that the machine learning models cannot exist in a vacuum:

to deliver value to the end user (system administrator, account manager, or system operations manager), machine learning models have to be tightly integrated into a broader framework that records the data, analyses it and then displays the results of the analysis.

In the world of generative AI, the realm of real-time telemetry processing will remain relevant and emerge as another significant topic of AI. Despite significant advances in large multi-modal models, foundational models that generalize across all time-series data from all domains remain elusive [63]. In the machine-learning landscape of the future, real-time data processing will emerge as a component, integrated into a broader framework with generative AI as described in Chapter 7.

8.2 Data analysis

When initially introducing data-driven methodologies for telemetry data, it's crucial to start with exploratory models. These models play a pivotal role in helping data scientists comprehend the data corpus. They are designed to uncover intriguing patterns and subgroups within the dataset, which, in turn, guide further analysis and study. In some cases, they may even pave the way for introducing custom machine-learning models.

8.2.1 Methodologies for Data Explorations

Alongside statistical methodologies for data exploration, which can provide only limited insights into the nature of the dataset, different machine learning methodologies have been proposed. The leading data exploration methodology in the field of unsupervised learning is clustering.

Clustering methodologies aim to define the *distance metric* between the points of the dataset and thus, by extension, define segments (clusters) of the dataset with points close to one another. The intuition behind clustering methodologies is that the defined distance measure, as determined by a specific clustering methodology, corresponds to a similarity between the data points in a dataset.

The identified clusters of data points are close to one another, thus corresponding to the instances (data points) that are similar to one another. Clustering is an invaluable aid to the data scientists as it allows them to, instead of analyzing each data point in the dataset individually, analyze each cluster - the intuition behind this being that all the data points in the same cluster are similar enough that they can share the same conclusions and insights.

The current data landscape, however, is much more complex than the simple tabular dataset anticipated by the clustering algorithms. The modalities of the data are such that they first require a powerful embedding approach to encode them into a latent space that preserves the notion of similarity. This is the approach of the

representation learning and the powerful (and large) embedding models for images and text (large language and large multimodal models).

Another modality of data that is often overlooked, however, is the *trace time-series data*. The intuition behind such data is that it is produced by different generators. Here, instead of comparing different data points (such as two different images or two different text documents), we wish to compare different generators that have produced the traces.

Examples of data traces produced by some hidden generator that cannot be directly observed are telemetry data for different machines on an assembly line, price traces for stocks or commodities, or, as described in this thesis, telemetry traces for nodes in an HPC system.

Compute nodes of an HPC system are a perfect prototype for studying the characteristics of different generators. If we abstract the methodology for data analysis as presented in Chapter 4, different compute nodes of an HPC system can be seen as hidden generators, producing an observable trace of the telemetry data as recorded by the monitoring system. Instead of comparing specific data points, we aim to compare the *generators themselves*. The final output of that methodology are not the clusters of data points but the clusters of compute nodes - clusters of *generators themselves*.

The methodology described in Chapter 4 and demonstrated on a dataset of HPC compute nodes has thus utility and applications far beyond the HPC field itself. It can serve as a prototype for data analysis methodologies that go from the point-wise comparison to the latent *data generator comparison* and become an invaluable part of a data-scientist framework when dealing with complex and heterogeneous datasets.

8.3 Anomaly detection

On the spectrum going from the more general to more specific models, anomaly detection is in the middle, between data exploration and anomaly prediction. While it provides less general observation than data exploration, described in Chapter 4), it is still more general than the anomaly prediction described in Chapter 6. Anomaly prediction requires a specifically defined label - node availability - that is predicted for different future windows. In contrast, anomaly detection simply detects periods of HPC compute node operation that deviate from the standard. While associated with compute node unavailability, these anomalies can also be related to other fluctuations in compute node behavior (like software misconfigurations).

When deploying a specific machine learning model, anomaly detection can be the first model deployed. Unlike supervised models (like anomaly prediction), anomaly detection requires no particular label in the training set. When deployed in this fashion, anomaly detection can be seen as an extension of data exploration.

Instead of identifying different clusters of data points, anomaly detection only recognizes a few interesting (outlying) data points that may require further analysis.

In the cases where the data objective is defined, such as the case of the HPC anomaly detection described in Chapter 5, anomaly detection serves as the first model that can be deployed even if the target label is not present in the dataset (thanks to the methodological innovation of *RUAD* [98]). The annotation of the data, powered by *RUAD* in conjunction with the domain expert, can then be used to create a dataset for more specific supervised models (like anomaly prediction).

8.4 Anomaly prediction

Amongst the modeling technologies presented in this work, anomaly prediction is the most specific as it requires the highest data quality. Still, in turn, it provides the highest quality and the most informative predictions. Anomaly prediction requires both a clear definition of the target label and the presence of the target label in the data set. Additionally, as described in Chapter 6, anomaly prediction also requires additional information in the data set in the form of *graph structure*.

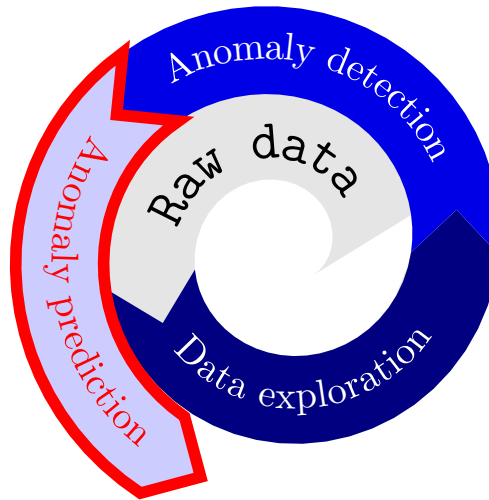


FIGURE 8.1: Position of *Anomaly prediction* in the developed data driven model.

As a part of a more comprehensive model zoo, anomaly prediction seats as the final type of model that is adopted during the introduction of data-driven model methodologies - data exploration and anomaly detection have to be adopted first:

- **Data exploration** helps the System Administrator and other stakeholders to determine which variables in the data require closer observation.
- **Anomaly detection**, in turn, helps improve the data set's quality with semi-automatic annotation, paving the way for anomaly prediction target applications.

- **Anomaly prediction** applications are similar to anomaly detection applications.

All the applications were anomaly detection, like fraud prediction in financial industries, and retail platform availability. All those domains would also benefit from anomaly prediction, but anomaly detection is a necessary step in achieving that.

An important innovation in the anomaly prediction methodology described in this thesis is using multiple data sources to construct a predictive model. The primary data source comprises datasets collected by the data acquisition systems, examined explicitly in the context of CINECA.

While these datasets are also used in anomaly detection models, anomaly prediction distinguishes itself by incorporating additional information through a *graph structure*.

This graph structure not only encodes the physical proximity of the machine room but also represents more abstract information and domain knowledge relevant to the modeling problem. This approach resembles applications where graph structures encode domain knowledge, particularly in lighter processing tasks. Although these tasks may seem different from anomaly prediction in high-performance computing systems, graphs have proven to be effective in capturing the data's shape and the ingested data's sparsity.

Overall, this work demonstrates that the anomaly prediction methodology, and more broadly, the idea of encoding domain knowledge as a graph structure, can serve as a blueprint for addressing complex challenges across various domains.

Chapter 9

Conclusions

The work performed during the PhD and presented in this thesis has been driven by two equal and interconnected principles: *performing fundamental machine learning research* and *solving real-life engineering problems*. I believe both goals are intrinsically connected as paradigm shifts in the real world can only be driven by scientific breakthroughs; any real-world impact not grounded in scientific progress is nothing more than smoke and mirrors, destined to collapse like a Potemkin village under the weight of its over-inflated promises and expectations. The only way to deliver truly impactful results, at any scale or domain, is the synthesis of these two principles: *scientific research and exploration, driven by real-life problems*.

Driven by these two principles, the work presented in this thesis should be evaluated along two axes: *how scientifically productive and innovative* it is and how well these innovations translate into *real-world impact*. The organization of the developed framework and machine learning models, as well as the organization of the thesis itself, follows this principle. The specific target domain of this work is high-performance computing (HPC) systems and the specific challenges that come with adapting machine learning methodologies for their monitoring and management. However, models, findings, and methodologies developed for and motivated by this specific set of requirements have applications far beyond the original domain.

The increasing size and complexity of modern HPC systems necessitate the introduction of advanced data collection, monitoring, and machine learning methodologies that support their management and operations. In literature, this collection of methodologies that go from data collection to data processing and visualization is called operational data analytics (ODA) for HPC systems. The thesis presents and discusses the comprehensive ODA framework comprising multiple models that address some of the most pressing open problems in the field: *open-ended data exploration, unsupervised anomaly detection, and long-term anomaly prediction*.

The ODA framework first establishes the continuum of the machine learning model adoption in the HPC systems, with each part of the framework addressing a specific stage with its unique requirements and previously unanswered questions. Depending on the level of adoption of operational data analytic methodologies, HPC

systems can adopt one, some, or all parts of the framework. The stages of the framework support each other; however, each stage, besides solving its primary objective, enables the adoption of the next one.

The first part of the comprehensive ODA framework is the methodology to perform open-ended *data exploration and analysis*. It motivated by trying to answer the question: Which compute nodes (a small part of the larger HPC system) are the most similar to one another? This seemingly trivial question has been an open problem in the literature. Existing approaches would compare nodes at a specific time (or short period). Comparing the complete history of compute nodes was impossible as it would require significant, multivariate time series clustering. Comparison, however, would, due to noise, yield unusable results. The methodology presented in this thesis brings a fundamentally different perspective to the problem: instead of trying to compare the traces produced by the compute nodes, we compare the autoencoders (type of self-supervised neural network) that best describe the compute nodes. This approach enables us to, for the first time, find similarities in the behavior of different compute nodes along the whole lifetime of a supercomputer.

Data exploration presented in *DEM* is the foundation of the comprehensive ODA framework as it requires no structured or labeled data and can thus be deployed as the first machine learning model adopted by the HPC system. *DEM* provides insights into the operation of the nodes and allows the HPC system administrators (or other relevant stakeholders) to identify relevant metrics that require additional analysis by dedicated machine learning models. One such relevant metric is compute node availability, which was studied by anomaly detection models.

Anomaly detection in HPC systems has been a well-explored, documented, and mature topic. Pioneered by professors Borghesi and Bartolini of the University of Bologna, it has been shown to be able to provide reliable anomaly signals that drastically reduce the response time of the system administrators and thus increase the overall availability of the HPC system. At the beginning of the PhD, this first generation of anomaly detection models was well established and understood. However, the significant drawback of this first generation of anomaly detection models has been the need for semi-supervised training and, consequently, a high-quality dataset. This first-generation anomaly detection model required a training set that contained no anomalous data. No anomalous data, in turn, requires that any system where such anomaly detection has been deployed tracks and documents the anomalies extremely carefully.

This requirement for "clean", high-quality datasets proved to be a big obstacle on the road to practical adoption as it prevents deployment on any system that produces a dataset where anomalies are not precisely annotated. In practice, this includes all but a few experimental and small-scale HPC installations. The open question of anomaly detection was thus: Is it possible to train an anomaly detection model on a dataset that is contaminated with anomalies? RUAD proves that it is

possible. It exploits a well-known phenomenon in machine learning, which is "over-fitting" the majority class. Since anomalous data points (failures) are in a significant minority, the carefully constructed structure of the RUAD model ignores them as it focuses on the majority of data. RUAD not only proves that unsupervised anomaly detection is possible (anomaly detection on the clean dataset), but it outperforms all other anomaly detection models (including all models that work on clean datasets).

RUAD can be deployed on a dataset that contains no labels. It is thus the second stage of the comprehensive ODA framework. Based on the results of the anomaly identification, if performed on an existing dataset, RUAD can, alongside the input from the domain experts, retroactively provide labels to that dataset. Consequently, in turn, paves the way for the last part of the ODA framework: a family of long-term anomaly prediction models.

In contrast to anomaly detection, anomaly prediction in HPC systems has been significantly less explored. It has either been focused on component failures (such as disk, GPU, or CPU failure) or has explored providing a short-term anomaly anticipation signal (with no clear time frame of the anomaly). GRAAFE is the first work that attempts the challenging problem of long-term failure prediction. The classical approach to anomaly prediction has proved futile as the anomalies are rare, and it is thus almost impossible to train a reliable (supervised) prediction model. GRAAFE circumvents these limitations by not only considering node trace data but also considering the physical layout of the compute nodes in a compute room. This additional information is encoded as a graph, which is then processed by graph neural networks. Compared to other machine learning models in HPC systems, GRAAFE is the first generation of model additional domain information and encodes it as a graph. This paradigm shift allows GRAAFE to significantly outperform all known anomaly prediction models and, for the first time, unlocks the ability to perform true long-term predictions—**up to 8 hours**—and reliable compute node failure predictions.

Every scientific innovation presented in this work has been conceived as an answer to a specific open real-world engineering challenge. However, it does not mean that the utility of the presented work ends within the confines of the HPC systems. Anomaly detection and graph methodologies with the idea of harvesting the structure information, have already proved to be a valuable tool in driving innovation in LiDAR processing and LiDAR sensor integrity research. Applications in the automotive and autonomous driving fields, far outside the HPC domain, prove that scientific work that is driven by *concrete questions* can still achieve *universality and widespread applicability*.

We live in the renaissance of AI and machine learning. The advent of new technologies, specifically generative AI, has changed the AI landscape and greatly impacted the economy and general society. It has reinvigorated the interest in the field of AI and spurred additional interest and enthusiasm of the general public. It has

also highlighted the need for an HPC system as the critical infrastructure that has to run efficiently and with the maximum possible availability. As such, the field of ODA and machine learning-powered HPC systems management is more important than ever.

Generative AI poses another, more fundamental question. In the world of ever-more advanced LLMs and autonomous agents, what is the role of human creativity? Some believe that AI will become the dominant creative force, replacing humans in music, creative writing, visual arts, and even scientific discovery. Recent discoveries show that even the *largest* language models still possess no symbolic reasoning skills and that we are still a fundamental scientific breakthrough away from real AI. Until then, such models will remain an exceptionally powerful (but limited) tool, glue, and packaging for models that humans will still develop. Models like GRAAFE and RUAD, as well as the future models developed by my mentors, colleagues, and students, will eventually (hopefully) surpass them.

Until then, scientific discoveries will remain the greatest and purest expression of being uniquely human. We have managed to build machines that can have echoes of human creativity and serve to help and empower us. Still, human creativity is the only known force in the universe that can truly create something new.

Appendix A

Publications and available resources

The work presented in this thesis is a sample of the most relevant research conducted during the PhD research duration. It represents the results that had the most significant scientific and practical value and presents a comprehensive collection of tools to address the challenge of effectively introducing machine learning methodologies into high-performance computing systems.

Alongside the selected works presented in detail in this thesis, additional research results have been obtained either as a derivative of the proposed approaches or its foundational explorations. As they do not form the main structure of the argument, they are collected here. They can be treated as a resource for further reading and may interest the reader who wishes to deepen research in the area connected to the thesis.

Main scientific results:

- Analysing Supercomputer Nodes Behaviour with the Latent Representation of Deep Learning Models (Molan et al. 2022 [94]). Foundation for the argument presented in Chapter 4. It shows that the trained semi-supervised models can be used as a data analysis tool beyond their utility for anomaly detection, identifying non-trivial compute node clusters.
- RUAD: Unsupervised anomaly detection in HPC systems (Molan et al. 2023 [98]). Foundation for the argument presented in Chapter 5. It introduces the first practically usable anomaly detection approach that can be trained entirely unsupervised.
- GRAAFE: GRaph Anomaly Anticipation Framework for Exascale HPC systems (Molan et al. 2024 [95]). Foundation for the argument presented in Chapter 6. It is the first work that uses large-scale training and deployment of large Graph Neural networks in HPC systems. It also shows that GNN opens the doors to previously impossible supervised training on classifiers on extremely imbalanced HPC node telemetry monitoring data.

Supporting the main scientific results are the following works:

- LIDAROC: Realistic LiDAR Cover Contamination Dataset for Enhancing Autonomous Vehicle Perception Reliability (Jati et al. 2024 [68]). This paper describes and demonstrates the phenomenon of environmental contaminants (water, mud, dust) introducing anomalous readings in LiDAR-based perceptive systems. Specifically, it introduces the concept of the critical failure: a high-confidence miss-classification by the downstream processing task. To develop algorithms to detect such anomalies, the paper also introduces the largest open-source LiDAR contaminant dataset.
- TinyLid: a RISC-V accelerated Neural Network For LiDAR Contaminant Classification in Autonomous Vehicle (Jati et al. 2024 [69]). This paper introduces the compute and energy-efficient LiDAR contaminant detection algorithm that is ported to the RISC-V system. Developed contaminant detection achieves latency and energy consumption in an order of magnitude lower than LiDAR (the data source).
- Exploring the Utility of Graph Methods in HPC Thermal Modeling (Guindani et al. 2024 [64]). This paper critically examines the utility of graph representations and graph neural networks for predicting the thermal evolution of HPC systems. It shows that while graphs are powerful representations in the context of HPC thermal modeling, they do not outperform the most optimized per-node models.
- AutoGrAN: Autonomous Vehicle LiDAR Contaminant Detection using Graph Attention Networks (Jati et al. 2024 [67]). This paper introduces the novel architecture of a LiDAR anomaly detection algorithm based on graph representations and graph detection neural networks.
- ExaQuery: Proving Data Structure to Unstructured Telemetry Data in Large-Scale HPC (Khan et al. 2024 [73]). This paper proposes the introduction of ontology (high-level graph representation) to structure unstructured HPC telemetry data and present it as a knowledge graph, supporting graph queries.
- Model for Quantitative Estimation of Functionality Influence on the Final Value of a Software Product (Molan et al. 2023 [91]). This paper introduces the methodology for quantitatively estimating the value of functionalities within the software development process. It is based on the idea of representing the software product as a (computational) graph, with each functionality representing a vertex in this graph. The software product's value can thus be modeled as a function of individual functionality values and the relationships between them.

- M100 ExaData: a data collection campaign on the CINECA's Marconi 100 Tier-0 supercomputer (Borghesi et al. 2023 [31]). This paper is a culmination of a multiple-year data collection and monitoring campaign by the University of Bologna. RUAD is demonstrated as a use case for the utility of the data, while GRAAFE uses its open dataset to validate the results.
- The Graph-Massivizer Approach Toward a European Sustainable Data Center Digital Twin (Molan et al. 2023 [100]). The work has examined the integration of GRAAFE and similar graph-based approaches into a larger framework for general-purpose graph processing called Graph-Massivizer.
- Graph neural networks for anomaly anticipation in HPC systems (Molan et al. 2023 [96]). Preliminary work preceding GRAAFE explored graph neural networks' feasibility and application to anomaly anticipation. Positive initial results then led to developing the comprehensive framework in GRAAFE.
- Machine Learning Methodologies to Support HPC Systems Operations: Anomaly Detection (Molan et al. 2022 [97]). Comprehensive presentation of the Ph.D. topic as well as motivation, supported by the analysis of state-of-the-art research for the research direction within the field of machine learning-powered data analytics in HPC systems.
- Semi-supervised anomaly detection on a Tier-0 HPC system. Preliminary results for the anomaly detection in HPC systems (Molan et al. 2022 [99]). Here, the novel approach of using temporal information for anomaly detection was initially explored. This idea was then developed into RUAD, where the possibility of unsupervised training was explored.
- An explainable model for fault detection in HPC systems (Molan et al. 2021). The first exploration of an explainable model for anomaly detection in the HPC systems. The model, still based on supervised training, however, proved to be less competitive compared to self-supervised approaches. This led to a change of direction in the development of the data analysis techniques and the development of the work presented in [92].
- Anomaly detection and anticipation in high-performance computing systems (Borghesi et al. 2021 [30]). This work represents the basis for all anomaly detection and data analysis approaches the thesis presents. Led by Professor Borghesi, it establishes the self-supervised, per-node models as the optimal approach toward anomaly detection in HPC systems.

Appendix B

Paper preprints

As a reference and for the sake of completion, the three primary papers that form the basis of the work are included in the appendix. The included version of the paper is the version after the revision and accepted for publication. The included versions, however, do not contain the formatting of the publications (by the publisher's copyright). The content of the papers, however, is precisely the same as in the published version. The included papers are (in order of appearance):

- Analysing Supercomputer Nodes Behaviour with the Latent Representation of Deep Learning Models, [94]
- RUAD: unsupervised anomaly detection in HPC systems, [98]
- GRAAFE: GRaph Anomaly Anticipation Framework for Exascale HPC systems, [95]

Analysing Supercomputer Nodes Behaviour with the Latent Representation of Deep Learning Models

Martin Molan¹[0000–0002–6805–2232], Andrea Borghesi¹[0000–0002–2298–2944],
Luca Benini^{1,2}[0000–0001–8068–3806], and Andrea Bartolini¹[0000–0002–1148–2450]

¹ DISI and DEI Department, University of Bologna, Bologna, Italy
{martin.molan2, andrea.borghesi3, luca.benini,
a.bartolini@unibo.it}@unibo.it

² Institut für Integrierte Systeme, ETH, Zürich, Switzerland

Abstract. Anomaly detection systems are vital in ensuring the availability of modern High-Performance Computing (HPC) systems, where many components can fail or behave wrongly. Building a data-driven representation of the computing nodes can help with predictive maintenance and facility management. Luckily, most of the current supercomputers are endowed with monitoring frameworks that can build such representations in conjunction with Deep Learning (DL) models. In this work, we propose a novel semi-supervised DL approach based on autoencoder networks and clustering algorithms (applied to the latent representation) to build a digital twin of the computing nodes of the system. The DL model projects the node features into a lower-dimensional space. Then, clustering is applied to capture and reveal underlying, non-trivial correlations between the features.

The extracted information provides valuable insights for system administrators and managers, such as anomaly detection and node classification based on their behaviour and operative conditions. We validated the approach on 240 nodes from the Marconi 100 system, a Tier-0 supercomputer located in CINECA (Italy), considering a 10-month period.

Keywords: supercomputer monitoring · deep learning · unsupervised learning · autoencoders · predictive maintenance.

1 Introduction

High Performance Computing systems have been steadily rising in size and complexity in the last years, as revealed by the exponential increase of the worldwide supercomputer installation³. HPC systems are typically composed by replicating a large number of components, usually, in the order of thousands of computing nodes, each of them constituted of a collection of smaller functional parts, such as CPUs, RAM, interconnections, storage, etc. Even if similar by design, each

³ <https://www.top500.org/>

computing node is affected by manufacturing variability and variations in the operating conditions. The sheer size and complexity of supercomputers create huge challenges in terms of optimal management of the IT components and their significant energy footprint[1]. The race towards Exascale⁴ continues to make these challenges ever more pressing[3–5].

Overall, it is a daunting task for system administrators and facility managers to optimize supercomputer performance and power consumption, identify anomalous behaviors faulty situations, and guarantee systems operate in optimal conditions. The scale of the problem motivates the development of automated procedures for anomaly detection and faulty node identification in current supercomputers and this need will become even more pressing for future Exascale systems[6]. The fact that most of today’s HPC computing systems are endowed with monitoring infrastructures[7] that gather data from software (SW) and hardware (HW) components can be of great help toward the development of data-driven automated approaches. Historically, system management was performed through hand-crafted scripts and direct intervention of system administrators; most of the data is stored in log files, and anomalies are investigated a posteriori to find the source of reported problems (e.g., when many users recognize the failure and report it to administrators). At the finer granularity, each core of the processing element is equipped with performance counters which can monitor several micro-architectural events (i.e., cache misses, stalls, throughput) and physical means (i.e., temperature, power consumption, and clock frequency). Processing units as well as the motherboard, the power distribution units, the onboard voltage regulators, the PCIe devices, and the fans are equipped with hardware (HW) sensors and counters. Similarly, software components can provide useful information as well, ranging from the details about jobs submitted by users (e.g., information gathered by job dispatchers such as SLURM[8] or PBS[9]) to software tools performing health-check of various subsystems[10] and I/O monitoring[11].

As the amount of data is overwhelming for human operators, automated processes could be highly beneficial in improving the data center usage to ease the burden of human operators and lower the response time to failures. In this context, Artificial Intelligence (AI) can provide significant benefits, as it allows to exploit the available big data effectively and to create decision support tools for HPC system administrators and facility managers[12, 13]. In the past, many works from the literature and the practice demonstrated the possibility to extract useful information using data collected from HPC computing nodes and employing supervised Deep Learning (DL) models[14–16] and semi-supervised ones[17–19]. These methods have been applied to detect nodes’ availability, defined as operation without anomalies. Availability and the corresponding error rate (1 minus availability rate) is a key metric of the node’s performance, and a target for optimization of the HPC system operation [20]. Due to its importance, we focus on the availability rate in the experimental part of this paper.

⁴ The supercomputer peak performance is expected to reach the ExaFlops (10^{18}) scale in 2023[2].

Borghesi et al. in [18] show that semi-supervised anomaly detection models trained on individual nodes data outperform a single model trained on multi-node data. This suggests that the semi-supervised model can learn differences between nodes even if the nodes share the same design and composition. Theoretically, the learned model encapsulates the node’s characteristics, however to the best of our knowledge, no one has ever evaluated the feasibility of using the disparities between trained DL models to evaluate the differences between the behavior of the corresponding nodes. In this work, we answer this question by introducing a novel approach that focuses on the latent representation of the trained DL models (in particular on the coefficients, the weights of the latent layer); the approach can identify clusters that deviate from the overall (node) population’s average availability, relying on the DL model parameters.

We focused on a Tier0 supercomputer composed of 985 nodes for which we trained a series of per-node semi-supervised DL models based on autoencoders (AE), as proposed by authors in [19], the state-of-the-art for semi-supervised anomaly and fault detection in HPC systems. We focus on semi-supervised methods as the availability of labels cannot be taken for granted in a supercomputer due to the non-negligible cost of annotating the vast wealth of monitored data. We explored different approaches to extract features from the weights and biases of the latent layer of the AE model. The key idea is to apply a geometric transformation to the weight matrix underlying the latent layer of the trained AEs; we opted to explore a variety of transformations; namely, we compute: (1) the vector of singular values, (2) the singular vector corresponding to the largest singular value, (3) the map of the representative vector (with and without bias), (4) the weights matrix similarity in L1, L2, and absolute L2 norm, (5) the affine (augmented matrix) similarity in L1, L2, and absolute L2 norm. The empirical evaluation demonstrates that the vector of singular values identifies interesting clusters among the different methods to extract salient features from the latent representation.

We propose to use the deviation from population average availability to evaluate the goodness of the clustering results. The vector of singular values, extracted from the weights matrix of the latent layer of the trained autoencoder, identifies two clusters with average overall availability lower than 89% (compared to 96% population average). The proposed method’s ability to identify these clusters is significant as the autoencoders have no access to the availability label during training.

2 Related work

Since anomalies in HPC systems are rare events, the problem of anomaly detection cannot be treated as a classical supervised learning problem [17, 21]; the majority of works that treat it in a fully supervised fashion have been tested using synthetic[14, 22] or injected anomalies[15]. Instead of learning the properties of both relevant classes, the standard approach is to learn just the properties of the system’s normal operation - anything deviating from this normal operation

is then recognized as an anomaly. Machine learning models are trained only on normal data to learn the characteristics of the normal operation. This training of ML models on normal data is called *semi-supervised* training [18].

The state-of-the-art for anomaly detection on the HPC system is to train a particular class of neural networks – called autoencoders – in a semi-supervised way [19]. Autoencoders are a specific type of neural networks that are trained to reproduce an input signal while simultaneously learning the most efficient latent representation of the data [23]. The latent representation of the data has a lower dimension than the original data; this lower dimension of the latent layer naturally leads to the idea of using autoencoders as pre-processing step before applying clustering techniques [24–26], as most of the clustering algorithms have worse performance in high-dimensional spaces [27]. The autoencoders are first trained on the whole dataset when using autoencoders as a dimension-reduction step before clustering. Then the dataset is projected (by the encoder part of the network) into a lower-dimensional latent layer [24].

Current approaches that combine clustering and autoencoder neural networks use a single trained autoencoder to encode each instance into a latent space. The state-of-the-art for HPC anomaly detection, however, is to train multiple models (a different model for each node in the system) [19]. The fact that the models trained on individual nodes outperform the model trained on combined data of all nodes [19, 17, 18] suggests that there are significant differences between the behavior of the compute nodes and, consequently, the corresponding trained models. Thus, this paper’s contribution is to explore the possibility of leveraging the fact that we are training multiple AE models to explore the relationship between the nodes themselves. Specifically, we explore the possibility to extract features from the trained neural networks to perform the clustering of the *whole operation history* of the compute nodes.

3 Methodology

In this section we present the architecture of the proposed approach. We start by providing the probabilistic perspective underlying the foundations of our approach in Sec. 3.1. We then describe in more detail the general architecture (Sec. 3.2) and then provide the more detailed description of the method in Sec. 3.3 and Sec. 3.4.

3.1 Probabilistic Background

The idea of extracting information and comparing trained neural networks extends the standard methodology of statistical modeling where two (or more) populations (or generally a collection of instances) are compared by contrasting parameters of fitted distributions. Comparing the parameters of fitted functions is the key idea underlying the proposed approach. Let us consider as an example the common statistical problem of comparing two populations of individuals – specifically, we want to compare a specific random variable X in two distinct

populations (e.g., height in two different countries). The first point of comparison in such cases is to calculate *empirical mean* $\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$ and *empirical variance* $\frac{1}{n-1} \sum_{i=1}^N (x_i - \bar{x})^2$. Two populations can be compared by looking at the empirical mean and variance of the random variables of interest (observed variables present inside each population).

The mathematical foundation of comparing mean and variance between two populations is directly in line with the idea of this paper. If we are observing two large populations, we know (from the central limit theorem [28]) that the sum of the variables will tend towards a Gaussian distribution. Two parameters determine Gaussian distribution: expected value μ and variance σ^2 [28]; to fit the Gaussian distribution to the data (population), we thus have to estimate these two parameters. If we fit the distribution via the Maximum Likelihood Estimation (MLE) method, [29], we see that the best estimator for the expected value is *empirical mean* and for variance, the best estimator is *empirical variance*. From the probability theory, we know that the difference of two random Gaussian variables is Gaussian variable with mean that is the difference of means and variance that is the sum of variances [28]. Comparing population mean and population variance is thus actually equivalent to comparing the *Gaussian distributions* fitted to the data.

Another perspective from which to examine the problem of comparing populations is that we fit a *function* to the data (this function being the Gaussian distribution). For some problems - like High Performance Computer (HPC) system monitoring - autoencoders (type of neural networks) achieve state-of-the-art results [19, 30]. As autoencoders are the class of functions that best describe this specific class of problems (behavior of compute node in an HPC system), we examine if we can compare the *compute nodes* by comparing the parameters of the fitted autoencoders.

3.2 General overview of the approach

Figure 1 reports the block diagram of the proposed methodology. We can identify the following steps:

1. On each node, a separate autoencoder model is trained. Semi-supervised training of per-node autoencoder models is adopted from the state-of-the-art paper [19].
2. After models are trained on each node, features are extracted (as described in Section 3.4) from the deep learning models.
3. Based on these extracted features, the similarity between nodes is calculated. Calculation of similarity can be done as the autoencoder projects the input features into a *latent* representation where only the most salient correlations between the input variables are preserved. The similarity measure is calculated by comparing the representation maps - specifically, the parameters of the latent layer.
4. This similarity measure is then used in hierarchical clustering.

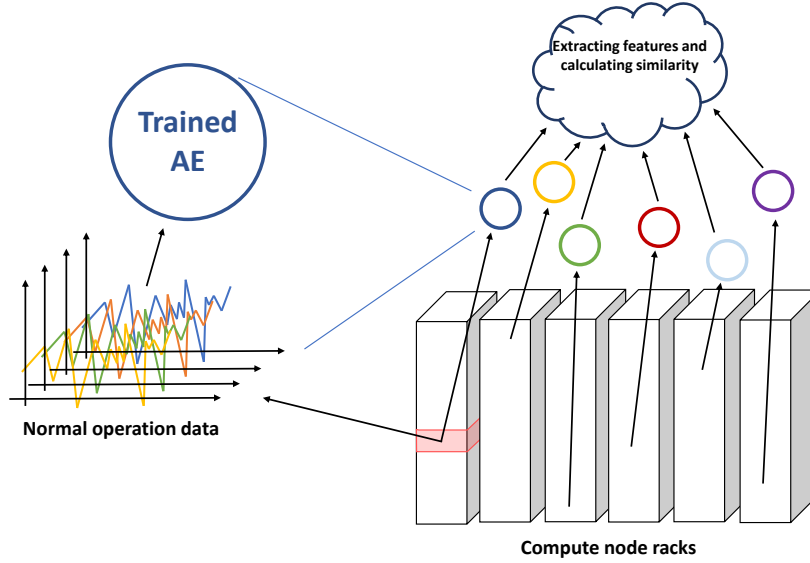


Fig. 1: Data flow schema. On each of the nodes (red in the picture), organized into racks, we train a separate autoencoder model (circles). From these trained models we extract features that are then used in the clustering of nodes.

3.3 Autoencoder models

Dense autoencoders are a type of deep neural network, which can be characterized by different topology; those used in this work have a distinct hourglass shape, a choice motivated by the results obtained by previous works in the state-of-the-art⁵. The most relevant information of the network is encoded in the latent layer. In this particular type of autoencoder, the latent layer is the layer in the middle of the network and contains the fewest neurons. It is preceded by the encoder and succeeded by the decoder, each composed of one or multiple layers. The encoder and decoder layers used in this work have a symmetrical architecture, which, generally speaking, is not strictly required. The fundamental role of the network is to efficiently encode the information from the input in a compressed representation in the latent layer. Training of the autoencoder is driven by the reproduction error produced by the decoder; reproduction error, which is the difference between the real input and the reconstructed signal, is minimized during training. The architecture of the network used in this work is presented in Figure 2. It is adapted from the work by Borghesi et al. [19] where it has been shown to produce state-of-the-art results in detecting anomalies on an HPC system.

The set of autoencoders - as in original work [19] - are individually trained on each node in a semi-supervised setting. Semi-supervised training means that the

⁵ They are also referred to as *contractive autoencoders*

data for training is filtered of all anomalies and that only the normal instances are used in training the model.

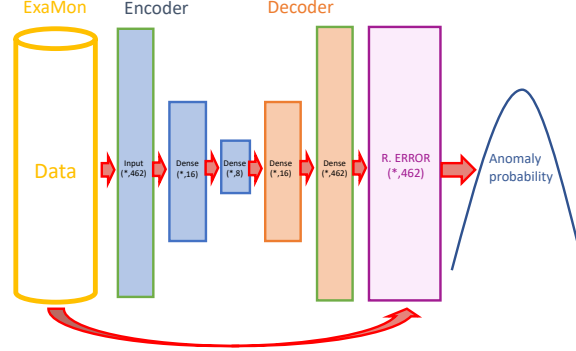


Fig. 2: Architecture of the state-of-the-art model, proposed by [19]. In this paper, relevant information is extracted from the latent layer Dense (*,8). Data is collected for the ExaMon monitoring system [31].

3.4 Feature extraction

Due to the architecture of the neural network used in this work - as discussed in Section 3.3 - we extract the relevant features from each node (each one with its data set); these features are embedded in the weights of the latent layer. The latent layer is described by the weights matrix W and the bias vector \vec{b} . The activation of the latent layer is given by $\vec{a}' = f(W\vec{a} + \vec{b})$ where f is a nonlinear activation function. In the next subsections, we will describe different encoding approaches of the latent layer information, which will then be used to extract features.

Singular value decomposition Singular value decomposition represents matrix M as $M = USV^*$ where S is a diagonal matrix containing singular values [32]. In this work, we used singular value decomposition on W , and we extracted the *vector* of singular values (abbreviated to singular values in the future) and a singular *vector* corresponding to the largest singular value (abbreviated singular vector).

Representative vector A vector of ones $\vec{1}$ is used as a representative vector as it corresponds to the activation of all neurons in a latent layer. It can serve as a proxy for the transformation of the (linear part) of the latent layer. For each node, we have thus calculated the product of $W\vec{1}$ (abbreviated vector of ones) and $W\vec{1} + \vec{b}$ (abbreviated vector of ones plus bias).

Matrix measures In this work, we leveraged the L1 and L2 norms induced in the matrix space (induced by p norms for vectors) [33]. Based on these norms we propose two ways to calculate distance between two matrices: $distance = ||A - B||_p$ and absolute distance $abs_distance = |||A - B||_p|$ where p is 1 or 2. Since the L1 measure is already symmetric, we do not separate a case with absolute distance. We introduce the absolute value as we want our distance measure to be symmetric.

We calculate the distance between nodes as a distance between the weights matrices of autoencoders trained on them. Additionally, since the linear part of the neural network is an affine transform, we introduce an augmented matrix A :

$$A = \left(\begin{array}{c|c} W & \vec{b} \\ \hline 0 \dots 0 & 1 \end{array} \right).$$

This matrix A captures the affine transform since $\vec{a}' = W\vec{a} + \vec{b}$ is equivalent to

$$\begin{pmatrix} \vec{a}' \\ 1 \end{pmatrix} = A \begin{pmatrix} \vec{a} \\ 1 \end{pmatrix}.$$

Another way to calculate the distance between nodes is to calculate the distance between an affine transform that is determined by the (affine $W\vec{a} + \vec{b}$) part of the latent layer of the corresponding autoencoder.

3.5 Clustering

The calculated distance between clusters is an input for clustering. In this work, we use agglomerative hierarchical clustering: each instance (in our case node) starts as its cluster. At every step of the iteration, the two closest clusters are connected. The connection between clusters is the closest distance between two instances in corresponding clusters. The combining of clusters is repeated until we reach the predetermined number of clusters.

3.6 Evaluating clustering

There are several possible measures to evaluate the goodness of the clustering (e.g., Silhouette score) [34]. These scores, however, are not applicable in the scenario explored by this work. We evaluate different possible feature extraction methods from the trained autoencoders; these different feature extraction approaches produce different feature spaces. Thus we cannot compare the clustering score (like Silhouette score) *between different spaces*. For this reason, we evaluate the relevance of our clustering approaches by evaluating how “interesting” the created clusters are.

The interest of clusters is reflected by how well they separate a specific variable. Since clustering is an unsupervised method, it is reasonable to assume that not all clusters will separate the same variable (such clustering would produce distinctly *uninteresting* clusters). However, we expect that there would be at

least one cluster where the distribution of the target variable would be significantly different than it is in the whole dataset. In this work, the target variable is system availability. In other words, clusters will separate computing nodes based on the autoencoder model’s latent layer encoding in groups having similar availability, thus similar failure rate. We stress that from a practical point of view, this means that an autoencoder model for each node is trained only on ”normal” operation samples and contains the information on the likelihood of the node to be available (or not to fail). Clusters of nodes sharing the same failure’s likelihood can be used to rationalize the maintenance procedure.

In the whole dataset, the system is available 0.96179% of the time. The most interesting cluster is thus the one where the average availability of a cluster will be as far away from this population average. The best clustering method is the one producing the most interesting cluster.

3.7 Random sampling baseline

The relevance of the produced clusters determines the relevance of feature extraction and, consequently, of clustering approaches. Specifically, we observe how well the clusters separate a target variable (in the case of this work, the node’s availability). To claim the relevance of the clustering approaches, we compare them to random sampling. We compare how well the target variable is separated by random sampling to how well clustering methods separate it. We are particularly interested in clustering methods that produce clusters and separations that do not (are very unlikely to occur) in random separation.

This paper implemented random clustering by producing a random matrix (of the same size and range as extracted features) that is then passed to clustering algorithms. The produced clusters are thus equivalent to random sampling without replacement. The generation of random clusters is repeated several (in this work 10) times. For each cluster, the distribution of the target variable is calculated; this distribution is then compared to distributions given by clustering methods. In the results (section 4), the range of randomly generated distributions is presented as a box with whiskers plot. Distributions outside the range of random distributions represent interesting patterns uncovered by the clustering method.

4 Results

This section presents the results of the experimental analysis conducted on a tier-0 supercomputer, Marconi100, hosted at CINECA, the largest Italian computing center. The results were conducted on a statistically significant fraction of the supercomputing nodes (more than two hundred) and cover a 10-months time span of production activity of the system.

4.1 Experimental setting

As explained in the methodology section 3, an individual model was trained on each of the 241 randomly selected nodes of Marconi100. Models were trained semi-supervised, meaning that only normal operation data was used for training. The whole dataset consists of 10 months of operational data collected on Marconi100. The first eight months of the data were used as a training set and the *last two* as a test set. Autoencoder models were trained on the train set. The cluster analysis was performed only on the test set.

The dataset used in this work consists of a combination of information recorded by Nagios (the system administrators tool used to visually check the health status of the computing nodes) and the Examon monitoring systems; the data encompasses the first ten months of operation of the M100 system. The features collected in the dataset are listed in table 1. In order to align different sampling rates of different reporting services, 15 minute aggregates of data points were created. 15 minute interval was chosen as it is the native sampling frequency of the Nagios monitoring service (where our labels come from). Four values were calculated for each 15 minute period and each feature: minimum, maximum, average, and variance.

Source	Features
Hardware monitoring	ambient temp., dimm[0-15] temp., fan[0-7] speed, fan disk power, GPU[0-3] core temp. , GPU[0-3] mem temp. , gv100card[0-3], core[0-3] temp. , p[0-1] io power, p[0-1] mem power, p[0-1] power, p[0-1] vdd temp. , part max used, ps[0-1] input power, ps[0-1] input voltage, ps[0-1] output current, ps[0-1] output voltage, total power
System monitoring	CPU system, bytes out, CPU idle, proc. run, mem. total, pkts. out, bytes in, boot time, CPU steal, mem. cached, stamp, CPU speed, mem. free, CPU num., swap total, CPU user, proc. total, pkts. in, mem. buffers, CPU idle, CPU nice, mem. shared, PCIe, CPU wio, swap free

Table 1: An anomaly detection model is created only on hardware and application monitoring features. More granular information regarding individual jobs is not collected to ensure the privacy of the HPC system users.

Features extracted from trained autoencoders are passed to hierarchical clustering. Hierarchical clustering has been chosen as it only requires the pairwise distance between the instances without making any assumptions about the space induced by the distance measure. The number of clusters is set to 20 for all experiments. A number of clusters is not a tuned parameter; 20 clusters represents roughly 10% of all nodes and is a randomly chosen number.

4.2 Trained autoencoder

The trained autoencoder is adopted from the current state-of-the-art semi-supervised approach for anomaly detection [19]. The structure of the autoencoder is presented in Figure 3. The autoencoder used as a binary classifier (form the normalized reconstruction error) on the test set achieves the AUC (area under the receiver-operator characteristic curve) of 0.7602.

Normal operation (the data where the autoencoder is trained) is determined by the label (system availability) provided by the monitoring systems.

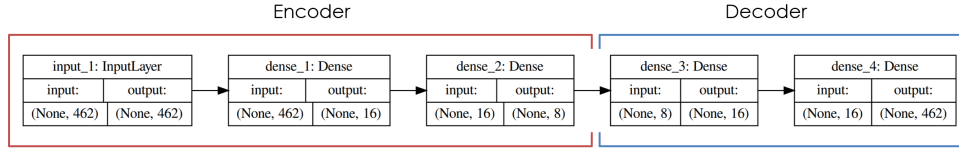


Fig. 3: Architecture of the autoencoder network, adopted from Borghesi et al. [19]

4.3 Cluster analysis: normal operation percentage

The proposed approach aims to identify interesting clusters of nodes that behave similarly. The similarity in behavior is also reflected in the fact that a cluster will have similar values for at least one relevant feature. In this section, we evaluate the similarity in average availability rate - in other words, we are interested in seeing if the clustering methods can identify clusters with particularly low availability (high failure rate). The average failure rate amongst 241 identified nodes (in the test set) is 0.96179. We wish to identify clusters with significantly lower availability rate.

In Table 2, the minimum average availability rates in a cluster, unidentified by a specific feature extraction approach, are reported. The table shows that the vector of singular values combined by the euclidean distance metric identifies a cluster with the minimum average availability. This availability is also lower than the random method's minimum availability (ever achieved).

In Figure 4 and Figure 5 average availability per node is plotted (red dots). Results of random sampling without replacement are presented as a box plot.

Distance measure:	Avg ava. in min. cluster:	Num. of nodes in min. cluster:
Sing. vector (Euc.)	0.9286	6
Vector of sing. values (Euc.)	0.8809	7
$W\vec{1} + \vec{b}$ (Euc.)	0.9126	8
$W\vec{1}$ (Euc.)	0.9367	5
W (absolute L2)	0.9191	7
A (absolute L2)	0.9276	5
W (L2)	0.9239	7
A (L2)	0.9124	10
W (L1)	0.9303	8
A (L1)	0.9303	8
Random sampling	0.9021	Not applicable

Table 2: Minimum average availability within clusters identified by different feature extraction methods. Vector of singular values identifies a cluster with the lowest average availability (highest anomaly rate). This is the most interesting method as it separates the target variable (node availability) the best. None of the proposed methods identify a cluster with a single node.

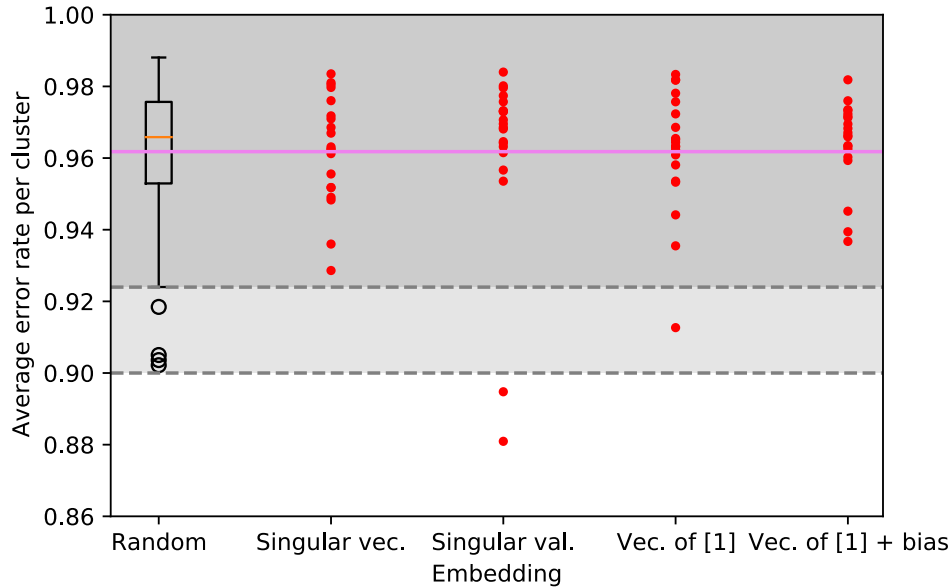


Fig. 4: Average error rate per cluster. Representation of nodes with a vector of singular values identifies two clusters with significantly higher anomaly rate than the whole population.

The average error rate across all nodes (0.96179) is marked with a violet dotted

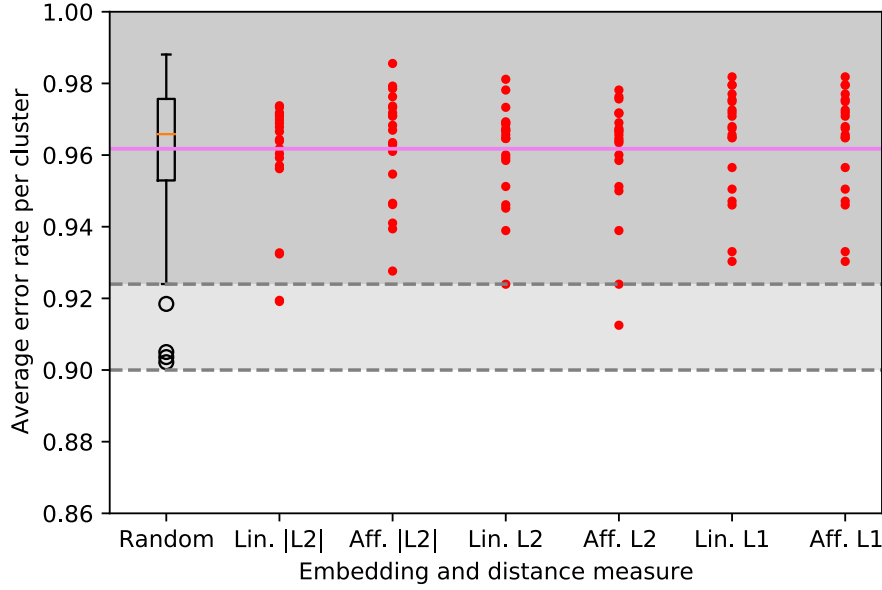


Fig. 5: Average error rate per cluster. Matrix-based feature extraction performs worse than the vector methods.

line. Area of values, observed in a random process, are marked with gray. Values *never* observed by the random process are left white.

Analyzing Figures 4 and 5 we observe that only the vector of singular values produced cluster with averages never observed in random samples.

The clustering method based on a vector of singular values combined with euclidean distance identifies two clusters with particularly low average availability. Such low average availability has also never occurred in a random selection of clusters. Low average availability means that hierarchical clustering based on singular value decomposition of weights matrix produces non-trivial clusters that are extremely unlikely to be matched by a random selection of clusters.

Identifying interesting clusters regarding availability is a non-trivial result as a neural network has no access to that label during training.

This promising result suggests that the created clusters share similar availability, and thus clusters can be created based on autoencoder semi-supervised models latent layer information. This cluster can then be used during the system’s lifetime to create canaries to focus the maintenance over nodes belonging to the same cluster of the canary node.

5 Conclusions

This work opens the possibility of extracting additional information from the state-of-the-art approach towards anomaly detection in the HPC setting. Besides

using per-node autoencoder models for anomaly detection [19, 17, 18], it is also possible to construct informative clusters from the parameters of the trained neural networks themselves.

We demonstrate the usefulness of the identified clusters on a concrete example: identifying clusters with the abnormal failure rate. This result is significant as the neural networks, from where the features are extracted, have no *access to that label during training*. Still, our approach can identify two clusters of nodes with lower availability (higher failure rate) than the population average.

We stress the fact that with this approach, clusters can be created based on a model trained on the first month of operations and then applied for the remaining lifetime of the system to focus maintenance to the nodes belonging to the same cluster containing the node which has experienced failures. System administrators focus their regular inspections only on canary nodes, each representative of one cluster.

6 Acknowledgments

This research was partly supported by the EuroHPC EU PILOT project (g.a. 101034126), the EuroHPC EU Regale project (g.a. 956560), EU H2020-ICT-11-2018-2019 IoTwins project (g.a. 857191), and EU Pilot for exascale EuroHPC EUPEX (g. a. 101033975). We also thank CINECA for the collaboration and access to their machines and Francesco Beneventi for maintaining Examon.

References

1. M. Garcia-Gasulla, B. J. Wylie, Performance optimisation and productivity for eu hpc centres of excellence (and european parallel application developers preparing for exascale): Best practice for efficient and scalable application performance, in: Platform for Advanced Scientific Computing (PASC) Conference, no. FZJ-2022-00887, Jülich Supercomputing Center, 2021.
2. P. Kogge, D. R. Resnick, Yearly update: exascale projections for 2013., 2013. doi:10.2172/1104707.
3. T. C. Germann, Co-design in the exascale computing project (2021).
4. J. Gao, F. Zheng, F. Qi, Y. Ding, H. Li, H. Lu, W. He, H. Wei, L. Jin, X. Liu, et al., Sunway supercomputer architecture towards exascale computing: analysis and practice, Science China Information Sciences 64 (4) (2021) 1–21.
5. O. Terzo, J. Martinovič, HPC, Big Data, and AI Convergence Towards Exascale: Challenge and Vision, CRC Press, 2022.
6. X. Yang, Z. Wang, J. Xue, Y. Zhou, The reliability wall for exascale supercomputing, IEEE Transactions on Computers 61 (6) (2012) 767–779.
7. K. S. Stefanov, S. Pawar, A. Ranjan, S. Wandhekar, V. V. Voevodin, A review of supercomputer performance monitoring systems, Supercomputing Frontiers and Innovations 8 (3) (2021) 62–81.
8. M. A. Jette, A. B. Yoo, M. Grondona, Slurm: Simple linux utility for resource management, in: In LCNS: Proceedings of JSSPP, Springer-Verlag, 2002.
9. A. P. Works, Pbs professional®14.2 plugins (hooks) guide, <https://pbsworks.com/pdfs/PBSHooks14.2.pdf> (2017).

10. T. Dang, N. Nguyen, Y. Chen, Hiperview: real-time monitoring of dynamic behaviors of high-performance computing centers, *The Journal of Supercomputing* 77 (10) (2021) 11807–11826.
11. B. Yang, X. Ji, X. Ma, X. Wang, T. Zhang, X. Zhu, N. El-Sayed, H. Lan, Y. Yang, J. Zhai, et al., End-to-end {I/O} monitoring on a leading supercomputer, in: 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), 2019, pp. 379–394.
12. A. Bartolini, F. e. a. Beneventi, Paving the way toward energy-aware and automated datacentre, in: *Proceedings of the 48th International Conference on Parallel Processing: Workshops*, 2019, pp. 1–8.
13. N. Wu, Y. Xie, A survey of machine learning for computer architecture and systems, *ACM Computing Surveys (CSUR)* 55 (3) (2022) 1–39.
14. O. Tuncer, E. Ates, Y. e. a. et Zhang, Online diagnosis of performance variation in hpc systems using machine learning, *IEEE Transactions on Parallel and Distributed Systems* (9 2018).
15. A. Netti, Z. Kiziltan, O. Babaoglu, A. Sîrbu, A. Bartolini, A. Borghesi, A machine learning approach to online fault classification in hpc systems, *Future Generation Computer Systems* (2019).
16. A. Bose, H. Yang, W. H. Hsu, D. Andresen, Hpcgc: A predictive framework on high performance computing cluster log data using graph convolutional networks, in: 2021 IEEE International Conference on Big Data (Big Data), IEEE, 2021, pp. 4113–4118.
17. A. Borghesi, A. Bartolini, et al., Anomaly detection using autoencoders in hpc systems, in: *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.
18. A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems, *Engineering Applications of Artificial Intelligence* 85 (2019) 634–644.
19. A. Borghesi, M. Molan, M. Milano, A. Bartolini, Anomaly detection and anticipation in high performance computing systems, *IEEE Transactions on Parallel and Distributed Systems* 33 (4) (2022) 739–750. doi:10.1109/TPDS.2021.3082802.
20. A. Netti, W. Shin, M. Ott, T. Wilde, N. Bates, A conceptual framework for hpc operational data analytics, in: 2021 IEEE International Conference on Cluster Computing (CLUSTER), 2021, pp. 596–603. doi:10.1109/Cluster48925.2021.00086.
21. B. Aksar, Y. Zhang, E. Ates, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, M. Egele, A. K. Coskun, Proctor: A semi-supervised performance anomaly diagnosis framework for production hpc systems, in: *International Conference on High Performance Computing*, Springer, 2021, pp. 195–214.
22. B. Aksar, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, M. Egele, A. K. Coskun, E2ewatch: An end-to-end anomaly diagnosis framework for production hpc systems, in: *European Conference on Parallel Processing*, Springer, 2021, pp. 70–85.
23. D. Bank, N. Koenigstein, R. Giryes, Autoencoders, *CoRR* abs/2003.05991 (2020). arXiv:2003.05991.
URL <https://arxiv.org/abs/2003.05991>
24. C. Song, F. Liu, Y. Huang, L. Wang, T. Tan, Auto-encoder based data clustering, in: J. Ruiz-Shulcloper, G. Sanniti di Baja (Eds.), *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 117–124.
25. X. Li, Z. Chen, L. K. Poon, N. L. Zhang, Learning latent superstructures in variational autoencoders for deep multidimensional clustering, *arXiv preprint arXiv:1803.05206* (2018).

26. W. Wang, D. Yang, F. Chen, Y. Pang, S. Huang, Y. Ge, Clustering with orthogonal autoencoder, *IEEE Access* 7 (2019) 62421–62432.
27. A. Alam, M. Muqeem, S. Ahmad, Comprehensive review on clustering techniques and its application on high dimensional data, *International Journal of Computer Science & Network Security* 21 (6) (2021) 237–244.
28. B. Davis, D. McDonald, An elementary proof of the local central limit theorem, *Journal of Theoretical Probability* 8 (3) (1995) 693–702.
29. L. L. Cam, Maximum likelihood: An introduction, *International Statistical Review / Revue Internationale de Statistique* 58 (2) (1990) 153–171.
URL <http://www.jstor.org/stable/1403464>
30. A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems, *Engineering Applications of Artificial Intelligence* 85 (2019) 634–644.
doi:<https://doi.org/10.1016/j.engappai.2019.07.008>.
URL <https://www.sciencedirect.com/science/article/pii/S0952197619301721>
31. A. Bartolini, F. Beneventi, A. Borghesi, D. Cesarini, A. Libri, L. Benini, C. Cavazoni, Paving the way toward energy-aware and automated datacentre, in: *Proceedings of the 48th International Conference on Parallel Processing: Workshops, ICPP 2019, Association for Computing Machinery, New York, NY, USA, 2019*.
doi:10.1145/3339186.3339215.
URL <https://doi.org/10.1145/3339186.3339215>
32. M. E. Wall, A. Rechtsteiner, L. M. Rocha, Singular value decomposition and principal component analysis, in: *A practical approach to microarray data analysis*, Springer, 2003, pp. 91–109.
33. G. Belitskii, et al., *Matrix norms and their applications*, Vol. 36, Birkhäuser, 2013.
34. F. Murtagh, P. Contreras, Algorithms for hierarchical clustering: an overview, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2 (1) (2012) 86–97.

RUAD: unsupervised anomaly detection in HPC systems

Martin Molan^a, Andrea Borghesi^a, Daniele Cesarini^b, Luca Benini^{a,c}, Andrea Bartolini^a

^a*DISI and DEI Department, University of Bologna, Bologna, Italy*

^b*CINECA consorzio interuniversitario, Bologna, Italy*

^c*Institut für Integrierte Systeme, ETH, Zürich, Switzerland*

Abstract

The increasing complexity of modern high-performance computing (HPC) systems necessitates the introduction of automated and data-driven methodologies to support system administrators' effort toward increasing the system's availability. Anomaly detection is an integral part of improving the availability as it eases the system administrator's burden and reduces the time between an anomaly and its resolution. However, current state-of-the-art (SoA) approaches to anomaly detection are supervised and semi-supervised, so they require a human-labelled dataset with anomalies - this is often impractical to collect in production HPC systems. Unsupervised anomaly detection approaches based on clustering, aimed at alleviating the need for accurate anomaly data, have so far shown poor performance.

In this work, we overcome these limitations by proposing RUAD, a novel Recurrent Unsupervised Anomaly Detection model. RUAD achieves better results than the current semi-supervised and unsupervised SoA approaches. This is achieved by considering temporal dependencies in the data and including long-short term memory cells in the model architecture. The proposed approach is assessed on a complete ten-month history of a Tier-0 system (Marconi100 from CINECA with 980 nodes). RUAD achieves an area under the curve (AUC) of 0.763 in semi-supervised training and an AUC of 0.767 in unsupervised training, which improves upon the SoA approach that achieves an AUC of 0.747 in semi-supervised training and an AUC of 0.734 in unsupervised training. It also vastly outperforms the current SoA unsupervised anomaly detection approach based on clustering, achieving the AUC of 0.548.

1. Introduction

Recent trends in the development of high-performance computing (HPC) systems (such as heterogeneous architecture and higher-power integration density) have increased the complexity of their management and maintenance [1]. A typical contemporary HPC system consists of thousands of interconnected nodes; each node usually contains multiple different accelerators such as graphical processors, FPGAs, and tensor cores [2]. Monitoring the health of all those subsystems is an increasingly daunting task for system administrators. To simplify this monitoring task and reduce the time between anomaly insurgency and response by the administrators, automatic anomaly detection systems have been introduced in recent years [3].

Anomalies that result in downtime or unavailability of the system are expensive events. Their cost is primarily associated with the time when the HPC system cannot accept new compute jobs. Since HPC systems are costly and have a limited service lifespan [4], it is in the interest of the system's operator to reduce unavailability times. Anomaly detection helps in this regard as it can significantly reduce the time between the fault and the response by the system administrator, compared to manual reporting of faulty nodes [5].

Modern supercomputers are endowed with monitoring systems that give the system administrators a holistic view of the system [3]. Data collected by these monitoring systems and historical data describing system availability are the basis for Ma-

chine Learning anomaly detection approaches [6, 7, 8, 9, 10], which build data-driven models of the supercomputer and its computing nodes. In this work, we focus on CINECA Tier0 HPC system (Marconi100 [11, 12] ranked 9th in Jun. 2020 Top500 list [13]), which employs a holistic monitoring system called EXAMON [14].

Production HPC systems are reliable machines that generally have very few downtime events - for instance, in Marconi100 at CINECA, timestamps corresponding to faulty events represent, on average, only 0.035% of all data. However, although anomalies are rare events, they still significantly impact the system's overall availability - during the observation period, there was at least one active anomaly (unavailable node) 14.4% of the time. State-of-the-art (SoA) methods for anomaly detection on HPC systems are based on supervised and semi-supervised approaches from the Deep Learning (DL) field [5]; for this reason, these methods require a training set with accurately annotated periods of downtime (or anomalies). In turn, this requires the monitoring infrastructure to track downtime events; in some instances, this can be done with specific software tools (e.g., Nagios [15]), but properly configuring these tools is a complex and time-consuming task for system administrators.

So far, the challenges of anomaly detection on HPC systems have been approached by deploying anomaly reporting tools by training the models in a supervised or semi-supervised fashion

[5, 16, 17, 8]. The need for an accurately labelled training set is the main limitation of current approaches as it is expensive, in terms of time and effort of the system administrators, to be applied in practice. Downtime tracking also has to be able to record failures with the same granularity as the other monitoring services. Some methods in production HPC systems only record downtime events by date [1, 2, 3]. In most production HPC systems, accurate anomaly detection is thus not readily achievable. For this reason, the majority of the methods from the literature were tested on historical or synthetic data or in supercomputers where faults were injected in a carefully controlled fashion [18]. Another limitation for the curation of an accurately labeled anomaly dataset is the short lifetime of most HPC systems. In the HPC sector, a given computing node and system technology have a lifetime of between three and five years. Short lifetime means, in practice, that the vendor has no time to create a dataset for training an anomaly detection model before the system is deployed to the customer site.

A completely unsupervised anomaly detection approach could be deployed on a new node or even on an entirely new HPC system. It would then learn online and without any interaction with the system administrators. Additionally, such a system would be easier to deploy as it would require no additional framework to report and record anomalous events (in addition to the monitoring infrastructure needed to build the data-driven model of the target supercomputer - a type of infrastructure which is becoming more and more widespread in current HPC facilities [3]).

Unsupervised anomaly detection approaches for HPC systems exist such as [19, 20, 21]. They either work on log or sensor data. Approaches based on log data [19, 21], while useful, can only offer a post-mortem and restricted view of the supercomputer state. The SoA for anomaly detection on sensor data [20] is based on clustering, which requires a degree of manual analysis from system administrators and offers poor performance compared to semi-supervised methods. The semi-supervised methods [5, 6, 22], based on the dense autoencoders, which are trained to reproduce their input, could be trained in an unsupervised fashion. However, none of the presented works has explored this possibility. According to the SoA, the models would perform worse as the dense autoencoder is also capable of learning the characteristics of the anomalies [5, 6, 22].

The primary motivation for this work is to propose a novel approach that relies *only on the fact that the anomalies are rare events* and works at least equally well when trained in an *unsupervised manner* as it does when trained in *semi-supervised manner* - this has not been the case in the current SoA. In this work, we propose an *unsupervised* approach: RUAD (Recurrent Unsupervised Anomaly Detection) that works on sensor data and *outperforms* all other approaches, including the current SoA semi-supervised approach [5] and SoA unsupervised approach [20]. RUAD achieves that by taking into account temporal dependencies in the data. We achieve that by using Long Short-Term Memory (LSTM) cells in the proposed neural network model structure, which explicitly take into consideration the temporal dimension of observed phenomena. We also show

that the RUAD model, comprising of LSTM layers, is capable of learning the characteristic of the normal operation even if the anomalous data is present in the test set - the RUAD model is thus able to be trained in an *unsupervised manner*. RUAD targets single HPC computing nodes: we have different anomaly detection models for each computing node. The motivation behind this is *scalability*: in this way, each node can be used to train its own model with minimal overhead - moreover, this strategy would work in larger supercomputers as well, as if the number of nodes increases, we just have to add new detection models.

1.1. Contributions of the paper

To recap, in this paper, we propose an anomaly detection framework that can handle complex system monitoring data, scale to large-scale HPC systems, and be trained even if no labelled dataset is available. The key contributions presented in this paper are:

- We propose a completely *unsupervised* anomaly detection approach (RUAD) that exploits the fact that the anomalies are rare and explicitly considers the *temporal dependencies* in the data by using LSTM cells in an autoencoder network. The resulting Deep Learning model *outperforms* the previous state-of-the-art semi-supervised approach [5], based on time-unaware autoencoder networks. On the dataset presented and analysed in this paper (collected from the Marconi100 supercomputer), the previous approach achieves an Area-Under-the-Curve (ACU) test set score of 0.7470. In contrast, our unsupervised approach achieves the best test set AUC score of 0.7672. To the best of our knowledge, this work is the first time such an approach has been applied to the field of HPC system monitoring and anomaly detection.
- We have conducted a *very large-scale experimental evaluation* of our methods. We have trained four different deep learning models for each of the 980+ nodes of Marconi100. To the best of our knowledge, this is the largest scale experiment relating to anomaly detection in HPC systems, both in terms of the number of considered nodes and length of time. Previous works only evaluate the models on a subset of nodes with a short observation time ([5] paper, for instance, only analyzed 20 nodes of the HPC system over two months). Per-node training of models also demonstrates the feasibility of *per node* models for large HPC systems. The training time for the individual model was under 30 minutes on a single NVIDIA Volta V100 GPU.

1.2. Structure of the paper

We present the current state-of-the-art and position our paper in Section 2. The machine learning approaches used for anomaly detection, including our novel approach, are described in section 3. The experimental setting for empirical validation of our results is detailed in Section 4.1 and our results are discussed in the rest of Section 4. Finally, Section 5 offers some concluding remarks.

2. Related Works

The drive to detect events or instances that deviate from the norm (i.e. *operational anomalies*) is present across many industrial applications. One of the earliest applications of anomaly detection models was credit card fraud detection in the financial industry [23, 24]. Recently, anomaly detection (and associated predictive maintenance) has become relevant in manufacturing industries [25, 26], internet of things (IoT) [27, 28, 29], energy sector [30], medical diagnostics [31, 32], IT security [33], and even in complex physics experiments [34].

Typically, anomalies in an HPC system refer to periods of (and leading to) suboptimal operating modes, faults that lead to failed or incorrectly completed jobs, or node and other components hardware failures. While HPC systems have several possible failure mitigation strategies [35] and fault tolerance strategies [36], anomalies of this type still significantly reduce the amount of compute time available to users [37]. The transition towards Exascale and the increasing heterogeneity of hardware components will only exacerbate the issues stemming from failures, and anomalous conditions that already plague HPC machines [1, 3, 38]. A DARPA study estimates that the failures in future exascale HPC systems could occur as frequently as once every 35-39 minutes [39], thus significantly impacting the supercomputing availability and system administrator load.

However, when looking at specific components and not at the entire HPC system (e.g., considering a single computing node), faults remain very rare events, thus falling under the area of anomaly detection, which can be seen as an extreme case of supervised learning on unbalanced classes [40]. Because data regarding normal operation far exceeds data regarding anomalies, classical supervised learning approaches tend to overfit the normal data and give a sub-optimal performance on the anomalous data [41]. In order to mitigate the problem of unbalanced classes, the anomaly detection problem is typically approached from two angles. Approaches found in the State-of-Art (SoA) that address the class imbalance either modify the data [42] or use specialized techniques that work well on anomaly detection problems [5]. Data manipulation approaches address the dataset imbalance either by decreasing the data belonging to normal operation (*under sampling the majority class*) or by oversampling or even generating anomalous data (*over sampling minority class*) [42]. Data manipulation for anomaly detection in HPC systems has not yet been thoroughly studied. Conversely, most existing approaches rely on synthetic data generation, e.g., injection of anomalies in real (non-production) supercomputers or HPC simulators [5].

Another research avenue exploits the abundance of normal data from HPC systems using a different learning strategy, namely semi-supervised ML models. Instead of learning on a dataset containing multiple classes – and consequently learning the characteristics of all classes – semi-supervised models are trained only on the normal data. Hence, they are trained to learn the characteristics of the of the normal class (the majority class in the dataset). Anomalies are then recognized as anything that does not correspond to the learned characteristic of the normal class [40, 6, 43, 22, 44].

Regarding the type of data used to develop and deploy anomaly detection systems, we can identify two macro-classes: system monitoring data collected by holistic monitoring systems (i.e. Examon [14]) and log data. This data is then annotated with information about the system or node-level availability, thus creating a label associated with the data points. The label encodes whether the system is operating normally or experiencing an anomaly. Since it is expensive and time-consuming to obtain labelled system monitoring data, a labelled dataset for supervised learning can be obtained by “injecting” anomalies into the HPC system (like [18]). Labels are important for both supervised, semi-supervised and unsupervised approaches. In the first case, they are used to compute the loss, in the second case to identify the training dataset and validation, and in the third case, only for validation. This data can then be used in a supervised learning task directly or after processing new features (feature construction). Examples of this approach are [45, 17, 46] where authors use supervised ML approaches to classify the performance variations and job-level faults in HPC systems. For fault detection, [8, 18] propose a supervised approach based on Random Forest (an ensemble method based on decision trees) to classify faults in an HPC system. All mentioned approaches use synthetic anomalies injected into the HPC system to train a supervised classification model. Approaches [5] and [16] are among the few that leverage *real* anomalies collected from production HPC systems (as opposed to injected anomalies). In this paper, we are interested in real anomalies, and thus, we will not include methods using synthetic/simulated data or injected anomalies in our quantitative comparisons.

All mentioned approaches do not take into account temporal dependencies of data (models are not trained on time series but on *tabular data containing no temporal information*). System monitoring data approach [47] is the first to take into account temporal dependencies in data by calculating statistical features on temporal dimension (aggregation, sliding window statistics, lag features). Most approaches that deal with *time series anomaly detection* do so on system log data. Labelled anomalies are either analyzed with log parsers [48] or detected with deep learning methods. Deep learning methods for anomaly detection are based on LSTM neural networks as they are a proven approach in other text processing fields.

Compared to labelled training sets, much less work has been done on unlabelled datasets - despite this case being much more common in practice. So far, all research on unlabelled datasets has focused on system log data. [19] propose a *k*-means based unsupervised learning approach that does not take into account temporal dynamics of the log data. A clustering-based approach on sensor data is proposed by [20]. This approach will serve as one of the baselines in the experimental section (as it is the only unsupervised approach on the sensor and not on log data). An approach [21] works on time series data in an unsupervised manner. It uses the LSTM-based autoencoder and is trained on the existing log data dataset. The proposed anomaly detector achieves the AUC (area under the receiver-operator characteristic curve) of 0.59. Although it works on a drastically different type of dataset (log data as opposed to system monitoring data),

it is the closest existing work to the scope of the research presented in this paper. As we show later in the paper, we can achieve much better results than the one reported for the log data models [21] by deploying an unsupervised anomaly detection approach on system monitoring data on a per-node basis. Table 1 summarizes the most relevant approaches described in this section, focusing on the training set and temporal dependencies.

	Tabular data	Time series
Supervised	[49, 9]	[47, 48, 10]
Semi-supervised	[5, 6, 43, 22]	
Unsupervised	[19, 20]	[21]

Table 1: Summary of anomaly detection approaches on HPC systems

The novelty of this paper is, in relation to the existing works, threefold:

- it introduces an *unsupervised time-series based* anomaly detection model named RUAD;
- it proposes a deep learning architecture that captures *time dependency*;
- the approach is evaluated on a *large scale production* dataset with *real anomalies* – this is the largest scale evaluation ever conducted on this kind of problem, to the best of our knowledge.

3. Methodology

In this section, we describe the proposed approach for unsupervised anomaly detection. We do not directly introduce the proposed method (the LSTM autoencoder deep network) as we want to show how it is a significant extension to the current state-of-the-art; thus, we start by introducing three baseline methods, i) exponential smoothing (serving as the most basic method for comparison), ii) unsupervised clustering and iii) the dense autoencoder used in [5]. We then describe our approach in detail and highlight its key strengths (the unsupervised training regime and the explicit inclusion of the temporal dimension).

3.1. Node anomaly labeling

We aim to recognize the severe malfunctioning of a node that prevents it from executing regular compute jobs. This malfunctioning does not necessarily coincide with removing a node for the production, as reported by Nagios. In our discussions with system administrators of CINECA, we have concluded that the best proxy for node availability is the most critical state, as reported by Nagios. For this reason, we have created a *new label* called *node anomaly* that has a value 1 if any subsystem reported by Nagios reports a critical state. From these events (reported anomalies), we then filter out known false positive events based on reporting tests or configurations in Jira [50].

Jira logs are supplied by CINECA. The labels used in our previous work [5] do not apply to M100 as they were extensively used to denote nodes being removed from production for testing and calibration. In this work, we are examining the early period of the HPC machine life-cycle, when several rounds of re-configuration were performed, thus partially disrupting the normal production flow of the system. Comparing the two labelling strategies in table 2, we can see that the overlap between the two is minimal. Additionally, there are far fewer anomalies as reported by the *node anomaly* mainly because the M100 went through substantial testing periods in the first ten months of operation where nodes are marked as removed from production while still functioning normally. In the remainder of the paper, class 0 or class 1 will *always* refer to the value of *node anomaly* being 0 or 1 respectively. Normal data is all data where *node anomaly* has value 0 and *anomalies* are instances where *node anomaly* has value 1.

	Node anomaly	
	0	1
Removed from production: False	12 139 560	4 280
Removed from production: True	15 783	12

Table 2: Comparison between *removed from production* and *node availability*. The anomalies studied in this work (node availability) significantly differ (and are more reliable) from anomalies studied in previous works. The new labels also mark much fewer events as anomalous.

3.2. Reconstruction error and result evaluation

The problem of anomaly detection can be formally stated as a problem of training the model M that estimates the probability P that a sequence of vectors of length W ending at time t_0 represents an anomaly at time t_0 :

$$M : \vec{x}_{t_0-W+1}, \dots, \vec{x}_{t_0} \rightarrow P(\vec{x}_{t_0} \text{ is an anomaly}). \quad (1)$$

Vector \vec{x}_t collects all feature values at time t ; the features are the sensor measurements collected from the computing nodes. W is the size of the past window that the model M takes as input. If the model does not take past values into account - like the dense model implemented as a baseline [5] - and the window size W is 1, the problem can be simplified as estimating:

$$M : \vec{x}_{t_0} \rightarrow P(\vec{x}_{t_0} \text{ is an anomaly}). \quad (2)$$

In the case of autoencoders, model M is composed of two parts: autoencoder A (a neural network) and the anomaly score, which is computed using the reconstruction error of the autoencoder. The reconstruction error is calculated by comparing the output of autoencoder model A and the real value vector \vec{x}_{t_0} . The task of model A is to reconstruct the last element of its input sequence:

$$A : \vec{x}_{t_0-W+1}, \dots, \vec{x}_{t_0} \rightarrow \hat{\vec{x}}_{t_0}. \quad (3)$$

Vector $\hat{\vec{x}}_{t_0}$ the reconstruction of vector \vec{x}_{t_0} . As in Eq. 2, window size W can be 1. The model A outputs normalized data. The reconstruction error is calculated as the sum of the absolute

difference between the output of model A and the normalized input value for each feature: $Error(t_0) = \sum_i^N |\hat{x}_i - x_i|$ where N is the number of features and \hat{x}_{t_0} is the output of the model A . The error is then normalized by dividing it by the maximum error on the training set: $Normalized\ error(t_0) = \frac{Error(t_0)}{\max(Error(t))}$. We estimate the probability for class 1 (anomaly) as

$$P(\vec{x}_{t_0} \text{ is an anomaly}) = \begin{cases} 1, & \text{if : } Normalized\ error \geq 1, \\ Normalized\ error, & \text{otherwise} \end{cases} \quad (4)$$

Based on probability $P(\vec{x}_{t_0} \text{ is an anomaly})$, the classifier makes the prediction whether the sequence $\vec{x}_{t_0-w}, \dots, \vec{x}_{t_0}$ belongs to class 1 (anomaly) or class 0 (normal operation). This prediction depends on a threshold T , which is a tunable parameter:

$$Class(\vec{x}_{t_0}) = \begin{cases} 1, & \text{if : } P(\vec{x}_{t_0} \text{ is an anomaly}) \geq T, \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

To avoid selecting a specific threshold T , we introduce the Receiver-Operator Characteristic curve (ROC curve) as a performance metric. It allows us to evaluate the performance of the classification approach for all possible decision thresholds [51]. The receiver-operator characteristic curve plots the true-positive rate in relation to the false-positive rate. The random decision represents a linear relationship between the two – for a classifier to make sense, the ROC curve needs to be above the diagonal line. For each specific point on the curve, the better classifier is the one whose ROC curve is above the other. The overall performance of the classifier can be quantitatively computed as the Area Under the ROC Curve (AUC); a classifier making random decisions has the AUC equal to 0.5. AUC scores below 0.5 designate classifiers that are worse than random choice. The best possible AUC score is 1, which is achieved by a classifier that would achieve a true-positive rate equal to 1 while having a false-positive rate equal to 0 (broadly speaking, this is only achievable on trivial datasets or very simple learning tasks).

3.3. Trivial baseline: exponential smoothing

Exponential smoothing is implemented as a trivial baseline comparison. It is a simple and computationally inexpensive method that detects rapid changes (jumps) in values. If the anomalies were simply rapid changes in values with no correlation between features, a simple exponential smoothing method would be able to discriminate them. Therefore, we chose exponential smoothing as a first baseline as it is computationally inexpensive and requires no training set. Additionally, if exponential smoothing performs poorly, this underlines that we are indeed solving a non-trivial anomaly detection problem, for which more powerful models are needed.

For the baseline, we choose to implement exponential smoothing per feature independently. Exponential smoothing for feature i at time t is calculated as:

$$\hat{x}_i = \alpha x_t^i + (1 - \alpha) \hat{x}_{t-1}^i, \forall i \in F \quad (6)$$

where \hat{x}_t^i is an estimate of x^i at time t and α is a parameter of the method. We do this for all features in set F . The estimate at the beginning of the observation is equal to the actual value at time t_0 : $\hat{x}_{t_0}^i = x_{t_0}^i$.

3.4. Unsupervised baseline: clustering

A possible approach to unsupervised anomaly detection is to use standard unsupervised machine learning techniques such as k-means clustering proposed by [20]. The clusters are determined on the train set; each new instance belonging to the test set is associated with one of the pre-trained clusters. We opted for this particular unsupervised technique for the comparison as it is the only unsupervised method found in the literature (to the best of our knowledge) which uses sensor data and not logs – and thus, we guarantee a fair comparison. It has to be noted, however, that clustering, while belonging to the field of unsupervised machine learning *cannot detect anomalies* in an unsupervised manner – for each of the clusters determined on the train set, the probability for the anomaly has to be calculated. This probability can only be calculated using the labels.

In this work, the clustering approach inspired by [20] is implemented to prove the validity of the obtained results. We have used K-means clustering [19] like it has been proposed in [20]. We have trained the clusters on the train set. Based on the silhouette score¹ on the train set, we have determined the optimal number of clusters for each node². The percentage of instances that belong to class 1 is calculated for each of the determined clusters. We use this percentage of anomalous instances as the anomaly probability for each instance assigned to a specific cluster. The train and test set split is the same as in all other evaluated methods.

3.5. Semi-supervised baseline: dense autoencoder

The competitive baseline method is based on the current state-of-the-art dense autoencoder model proposed by [5]. Autoencoders are types of neural networks (NN) trained to reproduce their input. The network is split into two (most often symmetric) parts: encoder and decoder. The role of the encoder is to compress the input into a more condensed representation. This representation is called the *latent layer*. To prevent the network from learning a simple identity function, we choose the latent layer to be smaller than the original input size (number of input features) [6]. The role of the decoder is to reconstruct the original input using the latent representation.

Dense autoencoders are a common choice for anomaly detection since we can restrict their expressive power by acting on the size of the latent layer. Compressing the latent dimension forces the encoder to extract the most salient characteristics from the input data; unless the input data is highly redundant, the autoencoder cannot correctly learn to recreate its

¹the Silhouette score is a measure of performance for a clustering method. It measures how similar an instance is to others in its own cluster compared to instances from the other clusters [52]. It is calculated as $S_{score} = \frac{b-a}{\max(a,b)}$ where a is the mean inter-cluster distance, and b is the mean nearest cluster distance for each sample.

²Optimal number of clusters is the number of clusters that produces the highest silhouette score on the train set.

input after a certain latent size reduction. In the current state-of-the-art for anomaly detection in production supercomputers ([5]) the dense autoencoder is used in a semi-supervised fashion, meaning that the network is trained using only data points corresponding to the normal operation of the supercomputer nodes (Class 0). Semi-supervised training is doable as the normal points are the vast majority and thus are readily available; however, this requires having labelled data or at least a certainty that the HPC system was operating in normal conditions for a sufficiently long period of time. Once the autoencoder has been trained using only normal data, it will be able to recognize similar but previously unseen points. Conversely, it will struggle to reconstruct new points which do not follow the learned normal behaviour, that is, the anomalies we are looking for; hence, the reconstruction error will be higher. The structure of the autoencoder model is presented in Figure 1a. The dense autoencoder does not take into account the temporal dynamics of the data – its input and target output are the same vector:

$$SoA : \vec{x}_{t_0} \rightarrow \vec{x}_{t_0}. \quad (7)$$

3.6. Recurrent unsupervised anomaly detection: RUAD

Moving beyond the state-of-the-art model, we propose a different approach, RUAD. It takes as input a sequence of vectors and then tries to reconstruct only the last vector in the sequence:

$$RUAD : \vec{x}_{t_0-W+1}, \dots, \vec{x}_{t_0} \rightarrow \vec{x}_{t_0}. \quad (8)$$

The input sequence length is a tunable parameter that specifies the size of the observation window W . The idea of the proposed approach is similar to the dense autoencoder in principle, but with a couple of significant extensions: 1) we are encoding an input sequence into a more efficient representation (latent layer) and 2) we train the autoencoder in an unsupervised fashion (thus removing the requirement of labelled data). The key insight in the first innovation is that while the data describing supercomputing nodes is composed of multi-variate time series, the state-of-the-art does not explicitly consider the temporal dimension – the dense autoencoder has no notion of time nor of *sequence* of data points. To overcome this limitation, our approach works by encoding the sequence of values *leading up to the anomaly*. The encoder network is composed of Long Short-Term Memory (LSTM) layers, which have been often proved to be well suited to the context where the temporal dimension is relevant [53]. An LSTM layer consists of recurrent cells that have an input from the previous timestamp and from the long-term memory.

To address the scale of current pre-exascale and future exascale HPC systems that will consist of thousands of nodes [3], we want a scalable anomaly detection approach. The most scalable approach currently for anomaly detection on a whole supercomputer is a node-specific approach as each compute node can train its own model. Still, we want to achieve this by minimally impacting the regular operation of the HPC system. This is why it is important for the proposed solution to have a small overhead. Additionally, since we want to train a per-node

model, we want the method to be data-efficient. To address these requirements, we choose not to make the decoder symmetric to the encoder. The proposed approach is thus comprised of a Dense decoder and an LSTM encoder. LSTM encoder output is passed into a dense decoder trained by reproducing the final vector in an input sequence. The decoder network is thus composed of fully connected dense layers. The architecture of the proposed approach is compared to the state-of-the-art approach in Figure 1.

The reduced complexity of training allows us to train a separate model for each compute node. As shown previously ([54]), node-specific models provide better results than a single model trained on all data. We decided to adopt this scheme (one model per node) after a preliminary empirical analysis showed no significant accuracy loss while the training time was vastly reduced (by approximately 50%); this is very important in our case as we trained one DL model for each of the nodes of Marconi 100 (980+), definitely a non-negligible computational effort.

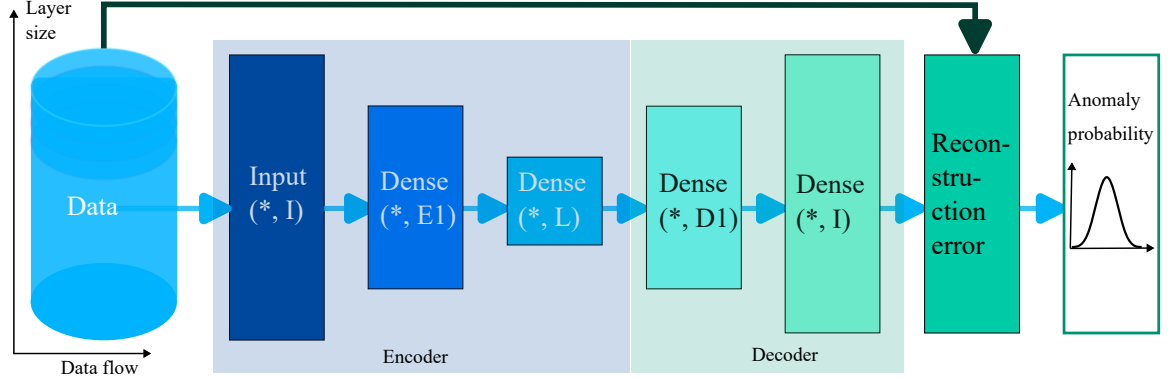
3.7. Data pre-processing

As introduced in Section 3.6 our proposed methodology consists of training a model for each node. Thus, the data from each node is first split into training and test sets. The training set contains 80% of data, and the test set contains the last 20% of data (roughly the last two months of data). It is important to stress that we have chosen to have two not overlapping datasets for the training and test. This avoids the cross-transferring of information when dealing with sequencing. Moreover, the causality of the testing is preserved. (No in-the-future data are used to train a model). This makes the results valid for in-practice usage.

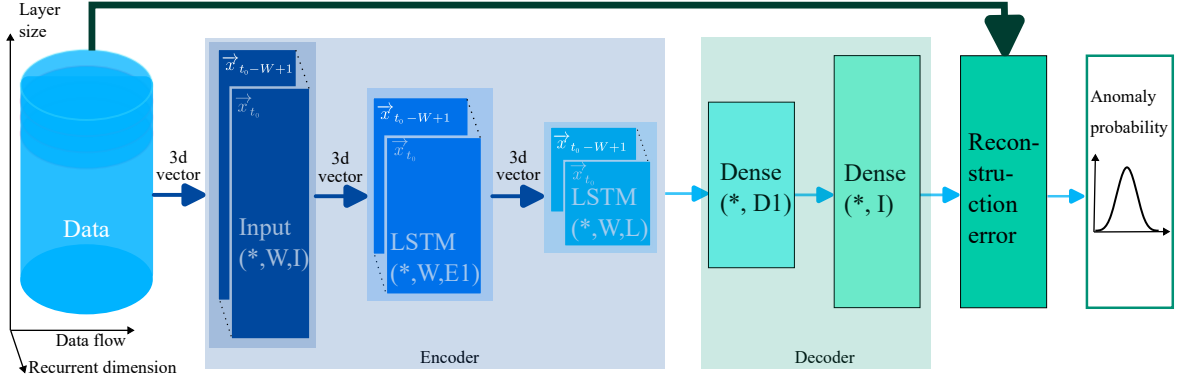
For semi-supervised training, the training set is filtered by removing anomalous events (anomalous events are identified by the *node anomaly* label as described in Section 3.1). We name this filter the semi-supervised filter, as depicted in Figure 2. For unsupervised learning, the training set is not filtered. For both the cases (unsupervised and semi-supervised learning), labels are used to evaluate the results. After filtering, a scaler is fitted to training data. A scaler is a transformer that scales the data to the $[0, 1]$ interval. In the experimental part, a min/max scaler is used on each feature [55]. After fitting to the training data, the scaler is applied to the test data - for rescaling the test set, min and max values of the training set are used (as it is standard practice in DL methods). After scaling, both training and test sets are filtered out to ensure time consistency: the data is split into sequences without missing chunks (missing chunks are the result of the semi-supervised filter). The sequences that are smaller than W are dropped. Finally, sequences are transformed into batches of sequences with length W . Figure 2 describes the whole data pre-processing pipeline.

3.8. Summary of evaluated methods

We compare our proposed approach RUAD against established semi-supervised and unsupervised baselines. Summary of pre-processing filters is presented in Table 3. The semi-supervised filter is applied to all semi-supervised approaches. A



(a) Structure of baseline model - the dense autoencoder.



(b) Structure of the proposed RUAD model consisting of the LSTM encoder and dense decoder.

Figure 1: The proposed approach replaces the encoder of the baseline model (1a) with the LSTM autoencoder (1b). The last layer of LSTM encoder returns a vector (not a temporal sequence) which is then passed to the fully connected decoder. W is the window size, I is the size of the input data, L is the size of the latent layer and $E1$ and $D1$ are sizes of encoder and decoder layer respectively. Chosen parameters for L , W , $E1$ and $D1$ are listed in Section 4.3.

time consistency filter is applied to methods that explicitly consider the temporal dimension of the data: Exponential smoothing and RUAD. RUAD and the current SoA anomaly detection approach based on dense autoencoders ([5]) is evaluated in both semi-supervised and unsupervised version.

Model	Filters		Name
	Semi-supervised	Time consistency	
Trivial baseline: exponential smoothing	NO	YES	<i>EXP</i>
Unsupervised baseline: clustering	NO	NO	<i>CLU</i>
DENSE autoencoder baseline semi-supervised	YES	NO	<i>DENSE_{Esemi}</i>
DENSE autoencoder baseline unsupervised	NO	NO	<i>DENSE_{Eun}</i>
RUAD semi-supervised	YES	YES	<i>RUAD_{semi}</i>
RUAD unsupervised	NO	YES	<i>RUAD</i>

Table 3: Short names and training strategies for examined methods. *DENSE_{Esemi}* is the current SoA [5].

We wish to highlight that, unlike the unsupervised learning baseline [20], our proposed method RUAD requires no additional action after the training of the model. The approach RUAD, proposed in this work, works on an *unlabeled dataset* and requires no additional *post training analysis*. A summary of approaches relating to training set requirements is presented in Table 4.

Method	Training set required	Post-training
<i>EXP</i>	Unlabeled dataset	No action required
<i>CLU</i> [20]	Unlabeled dataset	Assigning anomaly probability to clusters
<i>DENSE_{Esemi}</i> [5]	Labeled dataset	No action required
<i>RUAD</i>	Unlabeled dataset	No action required

Table 4: Comparison of implemented approaches relating to the training set requirements.

4. Experimental results

4.1. Experimental setting

The focus of the experimental part of this work is Marconi 100 (M100) HPC system, located in the CINECA supercomputing centre. It is a tier-0 HPC system that consists of 980 compute nodes organized into three rows of racks. Each compute node has 32 core CPU, 256 GB of RAM and 4 Nvidia V100 GPUs. In this work, nodes of the HPC system will be considered independent. This is also in line with the current SoA works [18, 6, 5] where anomaly detection is performed per node. Future works will investigate inter-node dependencies in the anomaly detection task.

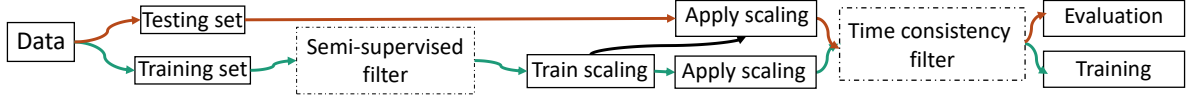


Figure 2: Data processing schema. Data flow is represented by green (training set) and orange (testing set). Scaling is trained on training set and applied on testing set to avoid contaminating the testing set. Semi-supervised and time consistency filters are optional and applied only when required by the modeling approach as indicated in Table 3

The monitoring system in an HPC setting typically consists of hardware monitoring sensors, job status and availability monitoring, and server room sensors. In the case of M100, hardware monitoring is performed by Examon[14], and system availability is provided by system administrators[15]. This raw information provided by Nagios, however, contains many false-positive anomalies. For this reason, we have constructed a new anomaly label called *node anomaly* described in Section 3.1.

For each of the 980 nodes of M100, a separate dataset was created. Dataset details are explained in Section 4.2. *DENSE* and *RUAD* models were trained and evaluated on the node-specific training and test sets for each node. The training set consisted of the first eight months of system operation, and the test set comprised the remaining two months. Such testing split ensures a fair evaluation of the model as described in Section 4.2. For the baseline, the exponential smoothing operation (defined in equation (6)) was applied only over the test set (as the approach requires no training). For each node, the scaler (for min and max scaling) was trained on training data and applied to test data. All results discussed in this section are *combined results from all 980 nodes of M100*.

The dense autoencoder and the *RUAD* model were trained in two different regimes: semi-supervised and unsupervised. For the semi-supervised training, the semi-supervised filter was applied that removed all data points corresponding to anomalies. In the unsupervised case, no such filtering was performed. It can hence be noticed one of the key advantages of the unsupervised approach: *no data pre-processing needs to be done and no preliminary knowledge about the computing nodes condition is required*.

For all three approaches (exponential smoothing, dense autoencoder and the *RUAD*), the probability for an anomaly (class 1) was estimated from reconstruction error as explained in Section 3.2. The probabilities from the test sets of all nodes from a single modelling approach (e.g. *RUAD* with observation window of length $W = 40$) were collected together to plot the Receiver Operator Characteristic (ROC) curve that is a characteristic for the modelling approach across all nodes. For clustering baseline and exponential smoothing (worst performing baselines), the ROC curve is compared against a dummy classifier which randomly chooses the class.

4.2. Dataset

The dataset used in this work consists of a combination of information recorded by Nagios (the system administration tool used to visually check the health status of the computing nodes) and the Examon monitoring systems; the data encompasses the

Source	Features
Hardware monitoring	ambient temp., dimm[0-15] temp., fan[0-7] speed, fan disk power, GPU[0-3] core temp. , GPU[0-3] mem temp. , gv100card[0-3], core[0-3] temp. , p[0-1] io power, p[0-1] mem power, p[0-1] power, p[0-1] vdd temp. , part max used, ps[0-1] input power, ps[0-1] input voltage, ps[0-1] output current, ps[0-1] output voltage, total power
System monitoring	CPU system, bytes out, CPU idle, proc. run, mem. total, pkts. out, bytes in, boot time, CPU steal, mem. cached, stamp, CPU speed, mem. free, CPU num., swap total, CPU user, proc. total, pkts. in, mem. buffers, CPU idle, CPU nice, mem. shared, PCIe, CPU wio, swap free

Table 5: An anomaly detection model is created only on hardware and application monitoring features. More granular information regarding individual jobs is not collected to ensure the privacy of the HPC system users.

first ten months of operation of the M100 system. The procedure for obtaining a *node anomaly label* is described in Section 3.1. The features collected in the dataset are listed in table 5. The data covers 980 compute nodes and five login nodes. Login nodes have the same hardware as the compute nodes but are reserved primarily for job submission and accounting. Thus we removed them from our analysis. The data is collected by the University of Bologna with approval from CINECA³.

In order to align different sampling rates of different reporting services (each of the sensors used has a different sampling frequency), 15 minute aggregates of data points were created. 15 minute interval was chosen as it is the native sampling frequency of the Nagios monitoring service (where our labels come from). Four values were calculated for each 15 minute period and each feature: minimum, maximum, average, and variance.

³CINECA is a public university consortium and the main supercomputing centre in Italy[56].

4.3. Hyperparameters

Hyper-parameters for all methods discussed in this paper were determined based on initial exploration on the set of 50 nodes. Chosen parameters performed best on the test from the initial exploration nodes (they achieved the highest AUC score on the test set). Results from the initial exploration set are excluded from the results discussed further in the chapter. Tuned hyperparameters include the structure of the neural nets (number and size of layers) and the smoothing factor of the exponential smoothing:

- Exponential smoothing: smoothing factor $\alpha = 0.1$
- Clustering: hyper-parameter (number of clusters) is trained on a train set for each node independently.
- Dense autoencoder: Structure of the network consists of 5 layers of shapes: $(*,462), (*,16), (*,8), (*,16), (*,462)$.
- RUAD (LSTM encoder, dense decoder): Structure of the network consists of 5 layers of shapes: $(*,W,462), (*,W,16), (*,W,8), (*,16), (*,462)$. W is the length of the observation window. Chosen window lengths W were: 5, 10, 20, 40.

4.4. Exponential smoothing

As mentioned in the methodology, exponential smoothing (EXP) is implemented to demonstrate that the anomalies we observe are not simply unexpected spikes in the data signal. Furthermore, exponential smoothing is applied to each feature independently of other features. As shown in Figure 3, exponential smoothing performs even worse than a dummy classifier (random choice). Poor performance of exponential smoothing shows that the anomalies we are searching for are more complex than simple jumps in values for a feature.

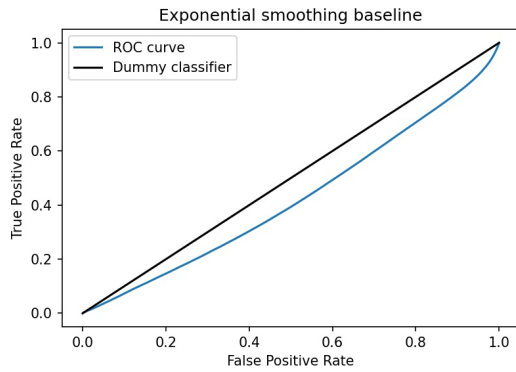


Figure 3: Combined ROC curve from all 980 nodes of M100 for the exponential smoothing baseline. Exponential smoothing performs even worse than the dummy classifier - anomaly detection based on exponential smoothing is completely unusable.

4.5. Clustering

The simple clustering baseline performs better than the exponential smoothing baseline and better than the dummy classifier, as seen in Figure 4. However, as we will illustrate in the following sections, it performs worse than any other autoencoder method. This demonstrates that the problem we are addressing (anomaly detection on an HPC system) requires more advanced methodologies like semi-supervised and unsupervised autoencoders.

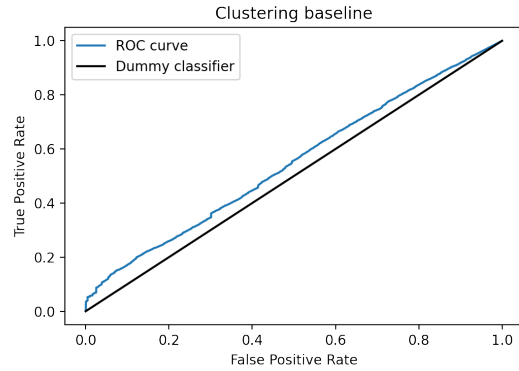


Figure 4: Combined ROC curve from all 980 nodes of M100 for the simple clustering baseline. This baseline performs only marginally better than the dummy classifier.

4.6. Dense autoencoder

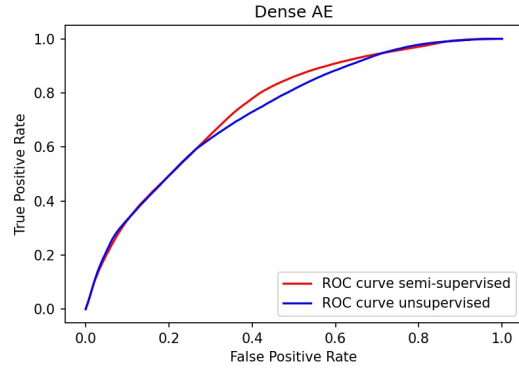


Figure 5: Combined ROC curve from all 980 nodes of M100 for the Dense autoencoder model. In the area interesting for practical application - True Positive Rate between 0.6 and 0.9 - semi-supervised approach outperforms unsupervised approach.

We consider now the dense autoencoder. We train a different network for each computing node of Marconi 100. The optimal network topology was determined during a preliminary exploration done on the sub-sample of the nodes of the system and following the guidelines provided by Borghesi et

al.[54]. In line with the existing work[5], the semi-supervised learning approach $DENSE_{semi}$ slightly outperforms the unsupervised learning approach $DENSE_{un}$ as seen in Figure 5. The better performance in the semi-supervised case is due to the nature of the autoencoder learning model - its capability to reconstruct its input. For example, suppose the autoencoder is fed with anomalous input during the training phase, as in the unsupervised case. In that case, anomalous examples in the training data constitute a type of “noise” that renders the autoencoder partially capable of reconstructing the anomalous examples in the test set.

4.7. RUAD

This section examines the experimental results obtained with the RUAD model (unsupervised LSTM autoencoder). The most important parameter is the length of the input sequence W that is passed to the model. This parameter encodes our expectation of the length of the dependencies within the data. Since each data point represents 15 minutes of node operation, the actual period we observe consists of $W \times 15 \text{ min}$. In this set of experiments, we selected the following time window sizes: 5 (75 minutes), 10 (2h30), 20 (5h), 40 (10h). These period lengths were obtained after a preliminary empirical evaluation; moreover, these time frames are in line with the typical duration of HPC workloads, which tend to span between dozens of minutes to a few hours[57]. We have trained the model in both semi-supervised $RUAD_{semi}$ and unsupervised $RUAD$ fashion for each selected window length. Results across all the nodes are collected in Figure 6.

4.8. Comparison of all approaches

The main metric for evaluating model performance is the area under the ROC curve (AUC). This metric estimates the classifiers’ overall performance without the need to set a discrimination threshold [51]. The closer the AUC value is to 1, the better the classifier performs. AUC scores for implemented methods are collected in table 6. From the lower table in table 6 (rows correspond to different training regimes and columns to window size for $RUAD$ network) and upper table in 6 (rows correspond to the performance of different implemented baselines), we see that the proposed approach outperforms the existing baselines. The highest AUC achieved by the previous baselines is 0.7470 (achieved by the $DENSE_{semi}$). This is outperformed by $RUAD$ for *all window sizes*. The best performance of $RUAD$ is achieved by selecting the windows size 10 where it achieves an AUC of 7.672. This result clearly shows that some temporal dynamics contribute to the appearance of anomalies.

The final consideration is the impact of observation window length W on the performance of the RUAD model. One might expect that considering longer time sequences would bring benefits, as more information is provided to the model to recreate the time series. This is, however, not the case (as seen in table 6) as the $RUAD$ achieves the best performance of 0.7672 with window size 10. The performance then reduces sharply with window size 40, only achieving an AUC of 0.7473. Several factors might explain this phenomenon. For instance, in

tens of hours, the workload on a given node might change drastically. Considering longer time series might thus force the RUAD model to concentrate on multiple workloads, hindering its learning task. Finally, an issue stems from the fact that there are gaps (periods of missing measurements) in the collected data (a very likely problem in many real-world scenarios). Longer sequences mean that more data has to be cut from the training set to ensure time-consistent sequences; this is because we are not applying gap-filling techniques at the moment⁴, thus, sub-sequences missing some points need to be removed from the data set. Combining these two factors contributes to the model’s decline in performance with longer observation periods.

Considering all discussed factors, the optimal approach is to use the proposed model architecture with window size $W = 10$ (i.e. 2 hours and 30 minutes), trained in an *unsupervised* manner. This configuration outperforms semi-supervised $RUAD_{semi}$ as well as the dense autoencoder. As mentioned in the related work (Section 2), labelled datasets are expensive to obtain in the HPC setting. Good unsupervised performance is why this result is promising - it shows us that if the anomalies represented a small fraction of all data, we could train an anomaly detection model even on an unlabeled dataset (in an unsupervised manner). Such a model not only achieves the state-of-the-art performance but *outperforms* semi-supervised approaches. The best AUC, achieved by the previous SoA $DENSE_{semi}$, is 0.7470. The best AUC score achieved by $RUAD$ is 0.7672. Moreover, unsupervised training makes this anomaly detection model more applicable to a typical HPC (or even datacentre) system.

Method	Combined ROC score
<i>EXP</i>	0.4276
<i>CLU</i>	0.5478
$DENSE_{semi}$	0.7470
$DENSE_{un}$	0.7344

Method	Combined ROC score			
Sequence length	5	10	20	40
$RUAD_{semi}$	0.7632	0.7582	0.7602	0.7446
$RUAD$	0.7651	0.7672	0.7655	0.7473

Table 6: According to expectations, the semi-supervised dense autoencoder outperforms the unsupervised dense one (highlighted by the higher AUC score). $RUAD$ and $RUAD_{semi}$ outperform all previous baselines. In contrast to the dense autoencoders, the proposed approach $RUAD$ performs best in *unsupervised manner*.

5. Conclusions

The paper presents an anomaly detection approach for HPC systems (RUAD) that outperforms the current state-of-the-art approach based on the dense autoencoders [5]. Improving upon

⁴We decided not to consider such techniques for the moment, as we wanted to focus on the modelling approach and gap-filling methods tend to require additional assumptions and to introduce noise in the data.

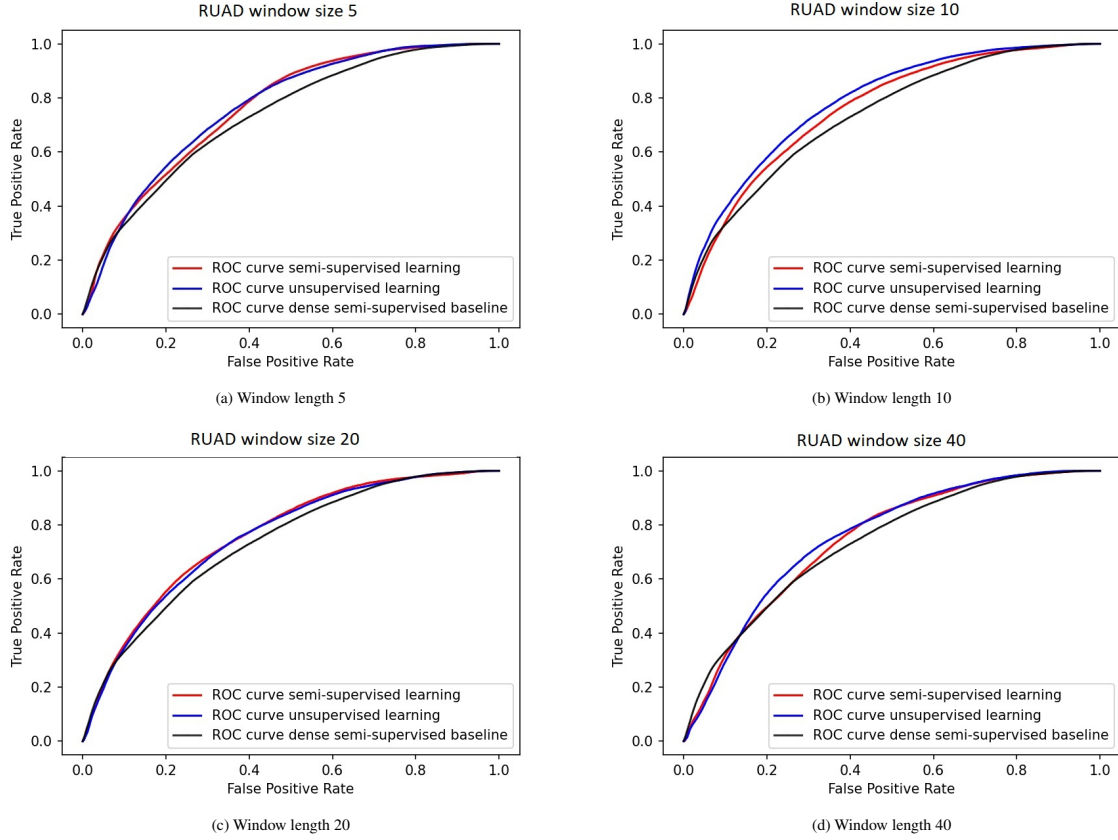


Figure 6: Combined results from all 980 nodes of M100. Comparison of different window lengths for the RUAD model. For all window lengths, performances of semi-supervised and unsupervised approaches are similar. Performance of the proposed model (red and blue line) is compared to the state-of-the-art baseline semi-supervised autoencoder proposed by Borghesi et al.[5].

state-of-the-art is achieved by deploying a neural network architecture that considers the temporal dependencies within the data. The proposed model architecture achieves the highest AUC of 0.77 compared to 0.75, which is the highest AUC achieved by the dense autoencoders (on our dataset).

Another contribution of this paper is that the proposed method – unlike the previous work [5, 16, 17, 8] – achieves the best results in an *unsupervised training* case. Unsupervised training is instrumental as it offers a possibility of deploying an anomaly detection model to the cases where (accurately) labelled dataset is unavailable. The only stipulation for the deployment of *unsupervised* anomaly detection models is that the anomalies are rare – in our work, the anomalies accounted for only 0.035% of the data. The necessity to have a few anomalies in the training set, however, is not a significant limitation as HPC systems are already highly reliable machines with low anomaly rates [58, 1].

To illustrate the capabilities of the approach proposed in this work, we have collected an extensive and accurately labelled dataset describing the first 10 months of operation of the Mar-

coni100 system in CINECA [56]. The creation of *accurately labelled* dataset was necessary to compare the performance of different models on the data rigorously. Because of the high quality and large scale of the available dataset, we can conclude that for the model proposed in the paper, the unsupervised model *outperforms* semi-supervised model *even if accurate anomaly labels are available*. This is the first experiment of this type and magnitude conducted on a real, in-production datacentre (both in terms of the number of computing nodes considered and the length of the observation period).

In future works, we will further explore the problem of anomaly detection in HPC systems, in particular, discovering the root causes of the anomalies - e.g., *why* a computing node is entering a failure state? We also have plans to further extend and refine the collected dataset and make it available to the public, in accordance with the facility owners and regulations about users' personal data (albeit not considered in this work, information about the users submitting the jobs to the HPC system can indeed be collected). Moreover, in this work, we focused on node-level anomalies; this was done to be comparable with the

state-of-the-art and for scalability purposes; in the future, we will explore the possibility of detecting systemic anomalies as well, i.e., anomalies involving multiple nodes at the same time. In this direction, the natural follow-up to the present work is to build hierarchical approaches which generate anomaly signals based on the composition of the signals generated by the node-specific detection models.

6. Acknowledgments

This research was partly supported by the EuroHPC EU PILOT project (g.a. 101034126), the EuroHPC EU Regale project (g.a. 956560), EU H2020-ICT-11-2018-2019 IoTwins project (g.a. 857191), and EU Pilot for exascale EuroHPC EUPEX (g.a. 101033975). We also thank CINECA for the collaboration and access to their machines and Francesco Beneventi for maintaining Examon.

References

- [1] W. Shin, V. Oles, A. M. Karimi, J. A. Ellis, F. Wang, Revealing power, energy and thermal dynamics of a 200pf pre-exascale supercomputer, in: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '21*, Association for Computing Machinery, New York, NY, USA, 2021, pp. 1–14. doi:10.1145/3458817.3476188. URL <https://doi.org/10.1145/3458817.3476188>
- [2] D. Milojicic, P. Faraboschi, N. Dube, D. Roweth, Future of hpc: Diversifying heterogeneity, in: *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 276–281. doi:10.23919/DATE51398.2021.9474063.
- [3] A. Netti, W. Shin, M. Ott, T. Wilde, N. Bates, A conceptual framework for hpc operational data analytics, in: *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 2021, pp. 596–603. doi:10.1109/Cluster48925.2021.00086.
- [4] L. A. Parnell, D. W. Demetriou, V. Kamath, E. Y. Zhang, Trends in high performance computing: Exascale systems and facilities beyond the first wave, in: *2019 18th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITHERM)*, 2019, pp. 167–176. doi:10.1109/ITHERM.2019.8757229.
- [5] A. Borghesi, M. Molan, M. Milano, A. Bartolini, Anomaly detection and anticipation in high performance computing systems, *IEEE Transactions on Parallel and Distributed Systems* 33 (4) (2022) 739–750. doi:10.1109/TPDS.2021.3082802.
- [6] A. Borghesi, A. Bartolini, et al., Anomaly detection using autoencoders in hpc systems, in: *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 24–32.
- [7] A. Borghesi, M. Milano, L. Benini, Frequency assignment in high performance computing systems, in: *International Conference of the Italian Association for Artificial Intelligence*, Springer, 2019, pp. 151–164.
- [8] A. Netti, Z. Kiziltan, O. Babaoglu, A. Sirbu, A. Bartolini, A. Borghesi, A machine learning approach to online fault classification in hpc systems, *Future Generation Computer Systems* (2019).
- [9] A. Netti, Z. Kiziltan, O. Babaoglu, A. Sirbu, A. Bartolini, A. Borghesi, Online fault classification in hpc systems through machine learning, in: *European Conference on Parallel Processing*, Springer, 2019, pp. 3–16.
- [10] M. Du, F. Li, G. Zheng, V. Srikumar, Deeplog: Anomaly detection and diagnosis from system logs through deep learning, in: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17*, Association for Computing Machinery, New York, NY, USA, 2017, p. 1285–1298. doi:10.1145/3133956.3134015. URL <https://doi.org/10.1145/3133956.3134015>
- [11] F. Iannone, G. Bracco, C. Cavazzoni, et al., Marconi-fusion: The new high performance computing facility for european nuclear fusion modelling, *Fusion Engineering and Design* 129 (2018) 354–358.
- [12] N. Beske, Ug3.2: Marconi100 userguide, accessed: 2020-08-17 (2020). URL <https://wiki.u-gov.it/confluence/pages/viewpage.action?pageId=336727645>
- [13] Top500list, <https://www.top500.org/lists/top500/2020/06/> (2020).
- [14] A. Bartolini, F. Beneventi, A. Borghesi, D. Cesarini, A. Libri, L. Benini, C. Cavazzoni, Paving the way toward energy-aware and automated data-centre, in: *Proceedings of the 48th International Conference on Parallel Processing: Workshops, ICPP 2019*, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1–8. doi:10.1145/3339186.3339215. URL <https://doi.org/10.1145/3339186.3339215>
- [15] W. Barth, Nagios: System and network monitoring, No Starch Press, 2008.
- [16] M. Molan, A. Borghesi, F. Beneventi, M. Guarrasi, A. Bartolini, An explainable model for fault detection in hpc systems, in: H. Jagode, H. Anzt, H. Laief, P. Luszczyk (Eds.), *High Performance Computing*, Springer International Publishing, Cham, 2021, pp. 378–391.
- [17] O. Tuncer, E. Ates, Y. e. a. et Zhang, Online diagnosis of performance variation in hpc systems using machine learning, *IEEE Transactions on Parallel and Distributed Systems* (9 2018).
- [18] A. Netti, Z. Kiziltan, et al., Finj: A fault injection tool for hpc systems, in: *European Conference on Parallel Processing*, Springer, 2018, pp. 800–812.
- [19] M. Dani, H. Doreau, S. Alt, K-means application for anomaly detection and log classification in hpc, in: *Lecture Notes in Computer Science book series (LNAI, volume 10351)*, 2017, pp. 201–210. doi:10.1007/978-3-319-60045-1_23.
- [20] A. Morrow, E. Baseman, S. Blanchard, Ranking anomalous high performance computing sensor data using unsupervised clustering, in: *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2016, pp. 629–632. doi:10.1109/CSCI.2016.0124.
- [21] S. Bursic, A. D’Amelio, V. Cuculo, Anomaly detection from log files using unsupervised deep learning (09 2019).
- [22] A. Borghesi, A. Libri, et al., Online anomaly detection in hpc systems, in: *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems, IEEE*, 2019, pp. 229–233.
- [23] G. Moschini, R. Houssou, J. Bovay, S. Robert-Nicoud, Anomaly and fraud detection in credit card transactions using the arima model (2020). arXiv:2009.07578.
- [24] M. Ahmed, A. N. Mahmood, M. R. Islam, A survey of anomaly detection techniques in financial domain, *Future Generation Computer Systems* 55 (2016) 278–288. doi:https://doi.org/10.1016/j.future.2015.01.001. URL <https://www.sciencedirect.com/science/article/pii/S0167739X15000023>
- [25] K. B. Lee, S. Cheon, C. O. Kim, A convolutional neural network for fault classification and diagnosis in semiconductor manufacturing processes, *IEEE Transactions on Semiconductor Manufacturing* 30 (2) (2017) 135–142.
- [26] L. Rosa, T. Cruz, M. B. de Freitas, P. Quitério, J. Henriques, F. Caldeira, E. Monteiro, P. Simões, Intrusion and anomaly detection for the next-generation of industrial automation and control systems, *Future Generation Computer Systems* 119 (2021) 50–67. doi:https://doi.org/10.1016/j.future.2021.01.033. URL <https://www.sciencedirect.com/science/article/pii/S0167739X21000431>
- [27] I. Martins, J. S. Resende, P. R. Sousa, S. Silva, L. Antunes, J. Gama, Host-based ids: A review and open issues of an anomaly detection system in iot, *Future Generation Computer Systems* 133 (2022) 95–113. doi:https://doi.org/10.1016/j.future.2022.03.001. URL <https://www.sciencedirect.com/science/article/pii/S0167739X22000760>
- [28] F. Cauteruccio, L. Cinelli, E. Corradini, G. Terracina, D. Ursino, L. Virgili, C. Savaglio, A. Liotta, G. Fortino, A framework for anomaly detection and classification in multiple iot scenarios, *Future Generation Computer Systems* 114 (2021) 322–335. doi:https://doi.org/10.1016/j.future.2020.08.010. URL <https://www.sciencedirect.com/science/article/pii/S0167739X19335253>
- [29] R. Xu, Y. Cheng, Z. Liu, Y. Xie, Y. Yang, Improved long short-term memory based anomaly detection with concept drift adaptive method for supporting iot services, *Future Generation Computer Systems* 112 (2020) 228–242. doi:https://doi.org/10.1016/j.future.2020.05.035.

- URL <https://www.sciencedirect.com/science/article/pii/S0167739X20302235>
- [30] S. Fu, S. Zhong, L. Lin, M. Zhao, A re-optimized deep auto-encoder for gas turbine unsupervised anomaly detection, *Engineering Applications of Artificial Intelligence* 101 (2021) 104199. doi:<https://doi.org/10.1016/j.engappai.2021.104199>. URL <https://www.sciencedirect.com/science/article/pii/S0952197621000464>
 - [31] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, N. V. Chawla, A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data, *CoRR* abs/1811.08055 (2018). arXiv:1811.08055.
 - [32] P. V. Astillo, D. G. Duguma, H. Park, J. Kim, B. Kim, I. You, Federated intelligence of anomaly detection agent in iotmd-enabled diabetes management control system, *Future Generation Computer Systems* 128 (2022) 395–405. doi:<https://doi.org/10.1016/j.future.2021.10.023>. URL <https://www.sciencedirect.com/science/article/pii/S0167739X21004192>
 - [33] T. Salman, D. Bhamare, A. Erbad, R. Jain, M. Samaka, Machine learning for anomaly detection and categorization in multi-cloud environments, 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud) (2017). arXiv:1812.05443, doi:10.1109/CSCloud.2017.15.
 - [34] M. Molan, Pre-processing for Anomaly Detection on Linear Accelerator. CERN openlab online summer intern project presentations (Sep 2020).
 - [35] M. Gamell, K. Teranishi, et al., Modeling and simulating multiple failure masking enabled by local recovery for stencil-based applications at extreme scales, *IEEE Transactions on Parallel and Distributed Systems* 28 (10) (2017).
 - [36] E. Meneses, X. Ni, et al., Using migratable objects to enhance fault tolerance schemes in supercomputers, *IEEE Transactions on Parallel and Distributed Systems* 26 (7) (2015) 2061–2074. doi:10.1109/TPDS.2014.2342228.
 - [37] I. Boixaderas, D. Zivanovic, et al., Cost-aware prediction of uncorrected dram errors in the field, in: 2020 SC20: International Conference for HPC, Networking, Storage and Analysis (SC), IEEE Comp. Soc., Los Alamitos, CA, USA, 2020, pp. 1–15.
 - [38] G. Iuhasz, D. Petcu, Monitoring of exascale data processing, in: 2019 IEEE International Conference on Advanced Scientific Computing (ICASC), 2019, pp. 1–5. doi:10.1109/ICASC48083.2019.8946279.
 - [39] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, et al., Exascale computing study: Technology challenges in achieving exascale systems, Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep 15 (2008).
 - [40] G. Pang, C. Shen, L. Cao, A. V. D. Hengel, Deep learning for anomaly detection: A review, *ACM Comput. Surv.* (mar 2020). doi:10.1145/3439950.
 - [41] G. Pang, C. Shen, L. Cao, A. V. D. Hengel, Deep learning for anomaly detection: A review, *ACM Comput. Surv.* 54 (2) (Mar. 2021). doi:10.1145/3439950. URL <https://doi.org/10.1145/3439950>
 - [42] G. Lemaître, F. Nogueira, C. K. Aridas, Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning, *Journal of Machine Learning Research* 18 (17) (2017) 1–5.
 - [43] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems, *Engineering Applications of Artificial Intelligence* 85 (2019) 634–644.
 - [44] P. Wu, C. A. Harris, G. Salavasidis, A. Lorenzo-Lopez, I. Kamarudzman, A. B. Phillips, G. Thomas, E. Anderlini, Unsupervised anomaly detection for underwater gliders using generative adversarial networks, *Engineering Applications of Artificial Intelligence* 104 (2021) 104379. doi:<https://doi.org/10.1016/j.engappai.2021.104379>. URL <https://www.sciencedirect.com/science/article/pii/S095219762100227X>
 - [45] O. Tuncer, E. Ates, et al., Diagnosing performance variations in hpc applications using machine learning, in: *International Supercomputing Conference*, Springer, 2017, pp. 355–373.
 - [46] B. Aksar, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, M. Egele, A. K. Coskun, E2ewatch: An end-to-end anomaly diagnosis framework for production hpc systems, in: *European Conference on Parallel Processing*, Springer, 2021, pp. 70–85.
 - [47] B. Aksar, Y. Zhang, E. Ates, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, M. Egele, A. K. Coskun, Proctor: A semi-supervised performance anomaly diagnosis framework for production hpc systems, in: B. L. Chamberlain, A.-L. Varbanescu, H. Ltaief, P. Luszczek (Eds.), *High Performance Computing*, Springer International Publishing, Cham, 2021, pp. 195–214.
 - [48] E. Baseman, S. Blanchard, N. DeBardeleben, A. Bonnie, A. Morrow, Interpretable anomaly detection for monitoring of high performance computing systems, in: *Outlier Definition, Detection, and Description on Demand Workshop at ACM SIGKDD*. San Francisco (Aug 2016), 2016, pp. 1–27.
 - [49] B. Aksar, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, M. Egele, A. K. Coskun, E2ewatch: An end-to-end anomaly diagnosis framework for production hpc systems, in: L. Sousa, N. Roma, P. Tomás (Eds.), *EuroPar 2021: Parallel Processing*, Springer International Publishing, Cham, 2021, pp. 70–85.
 - [50] Wikipedia, Jira (software) — Wikipedia, the free encyclopedia, <http://en.wikipedia.org/w/index.php?title=Jira\%20%28software%29&oldid=1052315603>, [Online; accessed 04-December-2021] (2021).
 - [51] Receiver operating characteristic (Nov 2021). URL https://en.wikipedia.org/wiki/Receiver_operating_characteristic
 - [52] K. R. Shahapure, C. Nicholas, Cluster quality analysis using silhouette score, in: 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA), 2020, pp. 747–748. doi:10.1109/DSAA49011.2020.00096.
 - [53] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, M. Weyrich, A survey on long short-term memory networks for time series prediction, *Procedia CIRP* 99 (2021) 650–655.
 - [54] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems, *Engineering Applications of Artificial Intelligence* 85 (2019) 634–644. doi:<https://doi.org/10.1016/j.engappai.2019.07.008>. URL <https://www.sciencedirect.com/science/article/pii/S0952197619301721>
 - [55] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research* 12 (2011) 2825–2830.
 - [56] Wikipedia, CINECA — Wikipedia, the free encyclopedia, <http://en.wikipedia.org/w/index.php?title=CINECA&oldid=954269846>, [Online; accessed 04-December-2021] (2021).
 - [57] M. C. Calzarossa, L. Massari, D. Tessera, Workload characterization: A survey revisited, *ACM Computing Surveys (CSUR)* 48 (3) (2016) 1–43.
 - [58] J. Dongarra, Report on the fujitsu fugaku system, University of Tennessee-Knoxville Innovative Computing Laboratory, Tech. Rep. ICLUT-20-06 (2020).

GRAAFE: GRaph Anomaly Anticipation Framework for Exascale HPC systems

Martin Molan^a, Mohsen Seyedkazemi Ardebili^a, Junaid Ahmed Khan^a, Francesco Beneventi^a, Daniele Cesarini^b, Andrea Borghesi^a, Andrea Bartolini^a

^aDISI and DEI Department, University of Bologna, Bologna, Italy

^bCINECA consorzio interuniversitario, Bologna, Italy

Abstract

The main limitation of applying predictive tools to large-scale supercomputers is the complexity of deploying Artificial Intelligence (AI) services in production and modeling heterogeneous data sources while preserving topological information in compact models. This paper proposes GRAAFE, a framework for continuously predicting compute node failures in the Marconi100 supercomputer. The framework consists of (i) an anomaly prediction model based on graph neural networks (GNNs) that leverage nodes' physical layout in the compute room and (ii) the computationally efficient integration into the Marconi100's ExaMon holistic monitoring system with Kubeflow, an MLOps Kubernetes framework which enables continuous deployment of AI pipelines. The GRAAFE GNN model achieves an area under the curve (AUC) from 0.91 to 0.78, surpassing state-of-the-art (SoA), achieving AUC between 0.64 and 0.5. GRAAFE sustains the anomaly prediction for all the Marconi100 nodes every 120s, requiring an additional 30% CPU resources and less than 5% more RAM w.r.t. monitoring only.

Keywords: Anomaly prediction, High-Performance Systems, Graph Neural Networks, MLOps

1. Introduction

Driven by the need for greater computational performance, today's high-performance computing (HPC) systems are becoming increasingly more complex. The current generation of petascale systems consists of hundreds of compute nodes, while the exascale systems consist of thousands of compute nodes. In addition to large scale, HPC systems are based on heterogeneous design, usually consisting of multiple AMD or NVIDIA GPUs per compute node [1]. This increased size, combined with cutting-edge technologies, heterogeneous design, and integration densities, increases the system's complexity and cost, directly transposing into more complex and failure-prone management and maintenance of the HPC installations. Machine learning-powered Operational Data Analytics (ODA) is introduced to support the HPC system administrators in addressing this complex task.

Proactive management and anticipation (prediction) of anomalous operation of the HPC system or its parts grows in importance with the system's size and cost. In HPC systems, anomalies refer to occurrences deviating from regular system operations that jeopardize its availability (capacity to perform computational tasks). In line with the recent works on HPC anomaly anticipation [2, 3], anomalies in this work are synonymous with the unavailability of the compute node to accept compute jobs; these are periods where a compute node is offline, removed from production, experiencing severe configuration issues or is otherwise unavailable to the job scheduler. According to Jauk et al. [4], anomalies can be at the node level (in this context, availability of the whole compute node) or component level (e.g., CPU [5], GPU availability [6]). In the aforementioned survey paper [4], the authors survey state-of-

the-art (SoA) methods for anomaly prediction, identifying two major approaches, one based on telemetry data and the other on log information. From the survey, the authors conclude that node-level anomaly prediction is less studied, and all the approaches discussed are *based on log information*. All the proposed approaches rely on the availability of datasets of production systems with annotated data. According to a recent survey on ODA frameworks deployed in production [7], only half of the centers use a log-monitoring capable ODA framework. Indeed, log collection has a significant performance impact (CPU utilization) [8] than *node telemetry*. In addition, log data shows higher privacy and security concerns than telemetry data, which consists only of numerical time-series data. If we restrict to telemetry data and node-level failures, only *anomaly detection* approaches have been studied [2, 3]. With this regard, the approaches in the SoA focus on detecting anomalies reported as the NAGIOS node's drained state. So far, related works have proposed models based on classifiers, autoencoders, and recurrent models, but all are trained per node.

Predicting anomalies is more complex than only detecting deviations from regular behavior, as previous and forthcoming system behavior should be considered. To cope with this difficulty, we opted not to treat single nodes in isolation from each other (as done in current solutions [2]) but rather to exploit the fact that computing nodes are physically and logically tightly coupled. In particular, nodes belonging to the same rack share a similar behaviour [9]; our idea is to exploit this proximity-based correlation to improve the overall accuracy. Moreover, we target the rising edge anomalies or transitions from normal operation to the anomaly. The persistence of the anomaly and its resolution depend solely on the response time of the system

administrators and are thus not a machine learning task.

In order to address the complexity of developing and deploying the topology-aware machine learning models to large production HPC systems, we introduce GRAAFE: graph anomaly anticipation framework for exascale HPC systems.

GRAAFE exploits Graph Neural Networks (GNNs), as they are well-suited to learning tasks where there are complex dependencies that can be represented as graphs[10]. GNNs have never been applied to represent topological information in supercomputers, but we show that they can handle proximity graphs defined by nodes in the same HPC rack; additionally, since this approach deals with multiple nodes at the same time (entire racks), it requires less models inference (only one per rack).

When it comes to bringing these models into production, current ODA frameworks lack a unified and standard framework. Some prototype implementations have been proposed to distribute model inference processing into the compute resources [11] or to leverage spark run-time [12] but are no longer adopted. In other fields, data analytics pipelines are used in production by means of MLOps frameworks applying DevOps principles and practices, such as continuous integration, delivery, and deployment, to the ML process for faster experimentation and model development. With GRAAFE, we extend the current ODA framework (namely ExaMon[13]) used in the CINECA [14] datacenter with Kubeflow, a widely used MLOps framework for Kubernetes.

Summing up, the paper’s main contributions are (1) the Development of a GNN to predict real, node-level anomalies that are shown to outperform other state-of-the-art approaches. With a 4-hour look-ahead window, the GNN anticipates anomalies with an accuracy of 0.91 AUC (Area-Under-the-Curve); to the best of our knowledge, GRAAFE is the first work to demonstrate the feasibility of anomaly prediction in a production supercomputer. (2) Large-scale training and validation (980 compute nodes) followed by deployment at scale to evaluate its real-world effectiveness. (3) Extension of an existing ODA at CINECA’s facility with the GNN via MLOps.

The pipeline can be deployed with only an additional 30% of CPU resources and less than 5% RAM usage increase. Of these overheads, the pipeline execution accounts for less than the 1%, making it scalable. (4) Ensure high reproducibility thanks to open-source code and the definition of detailed computing requirements to implement the monitoring system and MLOps framework for Tier-0 supercomputers. The source code, detailed guidelines for the deployment of GRAAFE as well as additional experimental results are available at <https://gitlab.com/ecs-lab/GRAAFE>.

The remainder of the manuscript is organized as it follows: Sec. 2 surveys the related literature; then, we have the core of the proposed approach, divided into Sec. 3 which covers the anomaly prediction problem and Sec. 3.3 that describes the deployment of the MLOps framework; Sec. 4 evaluates the performance of the method and Sec. 5 concludes the paper.

2. Related Works

We group the related works into three blocks: a) predictive maintenance in HPC, b) GNNs, and c) monitoring frameworks and MLOps for supercomputers.

2.1. Anomaly Detection & Prediction in HPC

One of the core objectives of ODA for HPC is anomaly detection and prediction, identifying aberrant or atypical patterns or behaviors from monitoring data [15] as soon as possible to minimize the system downtime. These deviations may comprise any unusual occurrences concerning resource usage, performance variations, or network traffic flow. This paper focuses on node-level anomaly analysis using node telemetry data. Several other approaches exist in the literature that either focus on component-level anomalies or on predicting anomalies based on log data [15, 4]. Approaches on log data deployed on a complex in-production large-scale HPC system predict node level anomalies with a future window of only 10 seconds [16]. Works with larger future windows, such as the work of Devesh Tiwari [17], or Yu Liu [18] focus on component failure prediction (disk failure specifically). Liu et al. [18] propose a disk failure prediction approach, training a generator adversarial network (GAN) for anomaly detection based on SMART disk monitoring data; the GAN is trained on normal data. The proposed approach achieves high accuracy for both the startup period and the lifetime use of the discs.

Considerations on performance, security, and privacy prevent the deployment of system log collection in some HPC centers [8]. The only available monitoring data for the HPC systems in those centers, such as M100 in CINECA, is the node telemetry data [19]. Minimal attempts have been made to create a node-level anomaly anticipation system based on node telemetry data. To the best of our knowledge, the only node telemetry-based approach that mentions anomaly anticipation is the work proposed by Borghesi et al. [2]. A self-supervised autoencoder network generates the anomaly anticipation signal. The autoencoder, trained to detect the anomalies, sometimes also generates an anomaly signal that anticipates the anomalies. The approach, however, fails to provide any estimate about the future window of the anomalies. Based on the taxonomy [4], the work presented in this paper is the first work that addresses the question of *node-level anomaly prediction based on node telemetry data*.

Based on the systematic literature review of predictive maintenance methodologies across all disciplines, Carvalho et al. [20] propose Random forest and artificial neural networks (dense neural networks) as the best-performing approaches for failure prediction. Behera et al. [21] propose a gradient-boosting tree for failure prediction applied to the problem of predictive maintenance of turbofan engines. Similarly, Zhang et al. proposed boosted decision trees and deep neural networks for anomaly detection in wide-area networks [22]. While Zhang et al. propose an anomaly detection and not an anomaly prediction method, since the approach is supervised, a similar method can be extended to the problem of anomaly prediction. These methods inspire baseline per-node methods implemented in this

work and are adapted for use as per-node anomaly predictors in HPC compute nodes.

2.2. Graph neural networks (GNN)

Most anomaly detection methods for HPC operate at a compute-node level, as they disregard the spatial structure of HPC systems, which has been used to improve the performance of predictive models [9]. These structures can be represented as graphs, thus suggesting the usage of graph-specialized ML models. GNNs are ML models for graph-structured data [10]. Graph Convolutional NNs (GCNs) constitute a special type of GNN that relies on executing convolution operations. The convolutions combine insights from neighboring nodes to create powerful embeddings used in downstream tasks, like labeling or link prediction[23].

GNNs are very good at exploiting node proximity[24]. As supercomputers are organized in racks of neighboring nodes, GNNs have great potential. For instance, GNNs can be trained using labeled data more efficiently than per-node supervised methods that tend to over-fit on the majority class. This advantage is especially noteworthy when dealing with extremely unbalanced classes typical of anomaly detection in HPC systems [23]. The use of GNN to improve anomaly detection and prediction performance has already partially been studied[25, 26]. For instance, Song et al.[27] use a couple of GNNs to identify abnormal performance fluctuations in cloud environments. The impact of proximity on the behavior of HPC nodes has been explored only in a limited way. Ghiasvand et al.[28] detect faults using system logs that consider nodes' proximity in terms of hardware architecture, resource allocation, and physical location.

The utility of graphs has been explored for anomaly detection in multivariate time series. Song et al. propose a GNN-based approach for anomaly detection. The approach observes the prediction error of the multivariate time series. Since this approach requires the calculation of observation error, it can only be used for anomaly detection, not prediction [29]. Consequently, it cannot be used as a baseline of comparison in this work.

2.3. Monitoring and ML ops frameworks in HPC

Operational data analytics (ODA) consists of two elements: 1) monitoring systems and 2) data analysis applications [30]. The monitoring systems in ODA are responsible for collecting and analyzing data from various sources within the HPC system. This data is then analyzed to provide actionable insights to system administrators and other stakeholders.

The basis for the application layer of the ODA is the data collection layer. Specifically for the HPC, several monitoring frameworks have been proposed: ExaMon, Ganglia, Nagios, and Prometheus [31]. The common denominator of all these approaches is that they need to have minimal impact on the overall system performance, provide data with low latency, and are capable of being scaled to large HPC systems. Building on the monitoring frameworks, there have been some attempts to integrate the monitoring and the application layers[31]. The latest

such approach is Wintermute [11], an ODA system developed for supercomputers. Wintermute comprises a custom-built distributed framework for "edge-telemetry data processing on the HPC compute node". Wintermute supports online access to the data and OpenCV ML kernels. While its distributed and online approach can reduce potential overhead in centralized processing, it is limited to OpenCV ML operations. It does not support continuous development and integration of the machine learning models.

The IT industry has addressed the drawbacks of data application platforms by introducing MLOps platforms. MLOps stands for "Machine Learning Operations". It is a set of practices and tools that combine software (SW) development (DevOps) and ML to manage the entire lifecycle of ML models. It focuses on creating reliable and scalable ML pipelines that can easily be integrated into existing systems while ensuring these models continuously improve. MLOps platforms are characterized by the following: 1) automation of repetitive tasks such as data preprocessing, model training, testing, and deployment; 2) a scalable design; 3) a modular deployment and easy integration with existing systems; 4) version control for code and appropriate documentation to ensure the reproducibility of results; 5) real-time monitoring of model performance; 6) collaborative development enabled by shared environments [32].

3. GRAAFE Architecture

The framework for exascale HPC systems (GRAAFE) comprises three main parts: The monitoring subsystem, the MLOps subsystem, and the anomaly anticipation/prediction model.

3.1. Monitoring system

The CINECA datacenter features a holistic monitoring framework, ExaMon, which aggregates a wide set of telemetry data [31] collected via a set of plugins (one for each monitored component) that read the sensor and communicate to the ExaMonDB via MQTT messages. ExaMonDB uses Cassandra and KairosDB technologies. The different monitored components are at the system level, the job scheduler, the cooling and power provisioning equipment, while at the compute node level, Nagios and Ganglia for in-band telemetry and IPMI for out-of-band telemetry. This work focuses on the Nagios state as anomaly label [3], and IPMI and Ganglia for node-level telemetry. The complete list of collected metrics is described by Borghesi et al. [19]. The Intelligent Platform Management Interface (IPMI) provides remote telemetry access to the built-in sensors for each node and its associated components, such as voltage regulators and fans. The ExaMon monitoring system collects sensor data with the IPMI interface with 20-second sampling rate. It stores these data in its internal KairosDB database as time traces and is remotely accessible through REST APIs [31].

3.2. Graph neural network model

To maintain the node physical proximity information in the ML anomaly prediction task, in GRAAFE, we propose to use a

Graph Convolutional Network (GCN) based on 1) the hypothesis that node physical proximity would benefit the anomaly prediction task and 2) preliminary experiments demonstrating its potential for improved performance. We encode the physical layout of compute nodes within racks as graphs. The connectivity matrix defines the specific structure of the input graph data. GCN can thus be thought of as a function that maps input data X (representing vectors corresponding to nodes) and connectivity matrix N to y , which is a vector of labels corresponding to each node in a graph: $GCN(X, N) \rightarrow y$.

In this work, we have explored two types of graph structures based on the physical proximity of the compute nodes in the computing room: rack-level and room-level models. The room-level model depicts the compute nodes as vertexes in a graph that are connected to their neighbors in X , Y , and Z dimensions (each node has a maximum of six neighbors). The weight of the connection is inversely proportional to their distance. The rack-level model achieved far superior performance and consisted of a line graph where each compute node was connected to the node above and below. The structure of the rack-level model is depicted in Fig. 1. While conducting the experimental evaluation, we did not conduct a comprehensive exploration of hyperparameters but instead relied on prior knowledge and manual fine-tuning. The optimal hyperparameters and the connectivity matrices for each model are described in the code repository.

The GCN models are trained on the supervised classification task. Labels are obtained by considering a future time window T ; label 1 indicates anomalies, and 0 indicates normal samples. For each node and at any point in time, a label of value 1 (anomaly) is assigned if the node encounters any anomaly in the future window T ; otherwise, the label will be 0.

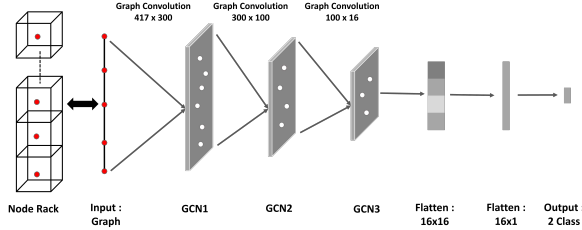


Figure 1: The structure of the GCN network exploits the organization of compute nodes in a rack.

3.3. Machine Learning Operations

The architecture of our novel high-performance computing monitoring and MLOps framework, as illustrated in Figure 2, is structured into three distinct layers: the data acquisition layer, represented by the monitoring systems; the data processing layer featuring a machine learning model, and a version control layer incorporating the git repository and the container registry. This three-layer structure follows the conceptual framework for operational data analytics in HPC, introduced by Netti et al. [30]. This paper focuses on the processing layer, which is a unique implementation built upon the Examon

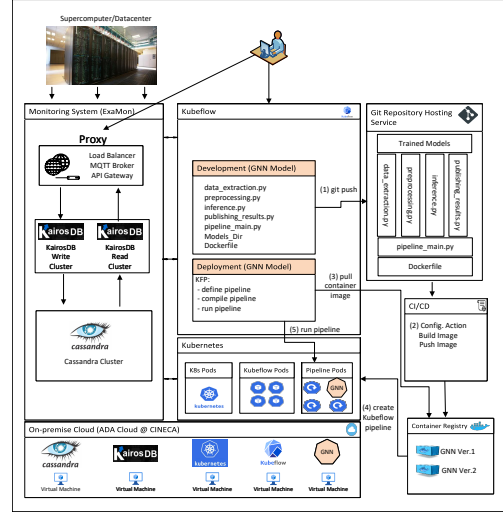


Figure 2: HPC monitoring and MLOps framework.

FW	rack GNN	room GNN	DNN	GB	RF	DT	MC
4	0,91	0,59	0,64	0,63	0,61	0,51	0,5
6	0,89	0,58	0,66	0,64	0,59	0,5	0,5
12	0,84	0,47	0,65	0,63	0,59	0,5	0,5
24	0,78	0,58	0,62	0,6	0,55	0,5	0,5
32	0,75	0,55	0,59	0,58	0,55	0,5	0,5
64	0,66	0,42	0,5	0,48	0,49	0,49	0,5
96	0,62	0,58	0,55	0,51	0,58	0,51	0,5
192	0,55	0,52	0,47	0,48	0,52	0,5	0,5
288	0,53	0,50	0,52	0,51	0,52	0,49	0,5

Table 1: The rack-level GNN outperforms (achieves higher AUC score) all other methods across all future windows (FW).

[31] monitoring and data acquisition system. The processing layer is further divided into three sublayers: the on-premises cloud layer (CINECA cloud), the container orchestration (Kubernetes) layer, and the MLOps (Kubeflow) layer.

The data processing pipeline, where the machine learning models are deployed, consists of a set of Python scripts responsible for (1) extracting data from the monitoring system, (2) preprocessing and transforming data to a suitable for the model, (3) making predictions based on input samples (inference), and (4) publishing the results to the monitoring system. The Output of the processing layer is sent back to the Examon monitoring system as another monitoring plugin, which allows easy integration into existing visualization and reporting systems [31].

4. Experimental Evaluation

We have conducted two main experiments: 1) anomaly prediction capability of the GNN-based approach (Sec.4.1); and 2) analysis of the deployment costs of the GNNs through MLOps and Kubernetes (Sec.4.2).

4.1. Prediction performance of the GNN models

The Marconi 100 (M100) HPC system located in CINECA was utilized to conduct an experimental evaluation. The dataset comprises 31 months wherein all 49 computer racks of the Marconi 100 system were subjected to observation. To establish an efficient model, 80% of the data mentioned above was designated as the train set, while historically, the remaining 20% was allocated for testing purposes.

The experimental part evaluated two GNN model architectures: rack-level GNN based on the line graph and the GNN covering the whole compute room. GNN approaches were tested against all known per-node anomaly anticipation approaches to demonstrate the necessity of the GNN. The label creation and the train/test split were the same across all examined models. Specifically, we have chosen the deep per-node neural network (DNN), and random forest (RF) inspired by Carvalho et al. [20] and the gradient boosting trees (GBT) and decision trees (DT) inspired by Behera et al. [21]. The hyperparameters of all the models and the accompanying code for the experimental evaluation are described and included in the code repository. The experimental evaluation is also performed on an open and publicly accessible dataset. This ensures transparency and reproducibility of the experimental setup for further research in this field.

4.1.1. Anomaly Prediction Model Performance

In line with other works in Operational Data Analytics and, specifically, anomaly detection [3], the area under the Receiver-Operator characteristic (ROC) curve (AUC) is selected as the primary evaluation metric for the experimental analysis. The predictive models output the probability of the anomaly for a given future window (instead of just predicting class 0 or 1). This helps system operators, as shown in Sec. 4.2.2. Furthermore, examining the whole ROC curve allows for a more general analysis that does not favor only the performance of the classifiers at a specific threshold. AUC is also chosen as a primary performance metric as it protects from pitfalls shared by other relatively more common metrics (precision, recall, f1) [33]. For comparison against other methods, f1 scores, as well as true positive, true negative, false positive, and false negative numbers, are reported in the code repository.

We have compared two GNN approaches, the rack-level GNN and the room-level GNN. Across all Future Windows (FW), the rack-level GNN significantly outperforms all other approaches that operate on per-node data. In contrast, the room-level GNN achieves the AUC values even lower than the per-node baselines. This suggests that, for the anomaly anticipation, the rack-level line graph GNNs are the optimal choice. All AUC values are collected in Table 1 where rows denote future windows (FW) and columns with different prediction methods. The performance of all models decreases with future observation windows, showing the increasing difficulty in detecting anomalies further away in the future. The comparative advantage of the rack-level GNN model also decreases with larger future windows. At the last observed window 288 (72 hours in the future), the performance differences between various mod-

els become almost negligible. All models are slightly better than the random choice model, which has a 0.5 AUC score.

The performance of the rack-level GNN model is above 0.65 AUC up to a future window of 64 (8 hours in the future). Then, the performance of all other models (including the per-node NN) dips to around the performance of the random choice. In other words, predicting up to 16 hours in the future is only possible with rack-level GNNs. The performance of all other per-node models is within expectation. The best performing is the per-node DNN, followed by GBT, RF, and, lastly, DT. The experimental analysis shows that the rack-level GNN model is far superior for predicting anomalies in HPC systems. For this reason, the rack-level GNN is chosen as the model to be deployed within the GRAAFE framework.

4.1.2. Importance of the graph structure

The work presented in this paper is motivated by introducing the graph structure for the predictive models in HPC systems. This section aims to answer a fundamental scientific question related to this premise: What is the importance of the graph structure, and how much does it contribute to the (per-rack) graph models outperforming the per-node models?

Based on the predictive performance, the best-performing model is the rack-based GNN model that takes a line graph of a compute rack as an input. To evaluate the actual contribution of the graph structure, we repeat the experimental evaluation with the same structure of the graph neural network but with the random graph as an input.

For each compute rack, the compute nodes (vertex of our graph) are connected to random other compute nodes from the same rack. For each compute rack, a different, random connectivity is chosen. Then a new GNN is trained on this rack structure, exactly repeating the testing and training procedure described above. This experimental procedure ensures that we are estimating a random AUC across a wide number of possible dummy graph topologies. Compared to the proposed model, the GNN receives the same vertex attribute (node) data; the only difference between the experiments is in the graph topology (connectivity between vertexes).

As seen in sub-Figure 3b, the AUC of the rack graph models, trained on random graphs, is lower than the performance of the rack models, trained on the proposed graph topology. The difference in performance, as depicted in Figure 3a, increases with the larger future window. As the performance of the predictive models decreases, so does the advantage that the sensible graph topology brings to the predictive model.

The results obtained by this experiment are significant as they clearly demonstrate the impact the graph topology has on the performance of the graph model. Both sets of models are built on the same attribute training data; the graph's structure alone, however, results in a difference in AUC as high as 0.2.

The average performance of random rack models is only slightly higher than the best-performing per-node model deep neural network (DNN in table 1). This shows that, while there is an advantage in processing the compute racks together, this advantage is minimal if the correct graph structure does not support it.

The results also motivate future work in building predictive graph-based models for HPC systems. Alongside the research in novel modeling approaches and neural network structures, future research has to focus on finding better connectivity topologies between compute nodes. The connectivity between the compute nodes and the associated graph structures discussed in this work is based on the physical proximity and organization of compute nodes in a computing room. We propose a graph topology that produces good results and demonstrates the importance of the graph topology. However, the question of the optimal graph structure remains a topic of ongoing research.

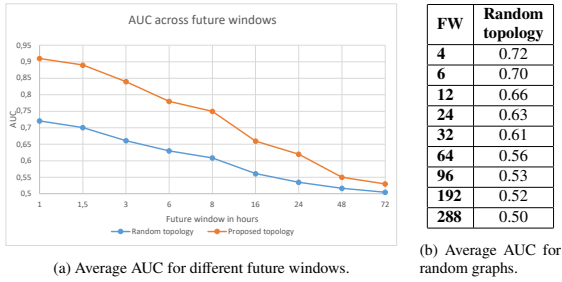


Figure 3: The average area under the curve (AUC) of the random per-rack graph topology is compared to that of the proposed per-rack graph topology. For each of the per-rack models, a different random graph is initialized.

4.2. Deployment Evaluation

To implement the proposed framework, we employ a cloud system co-hosted in the CINECA supercomputing facility (on-premise) without creating any overhead on the HPC nodes. This cloud infrastructure is based on the OpenStack version of Wallaby. The nodes of this cloud system are composed of Dual-Socket Dell PowerEdge servers, 2xCPU 8260 Intel CascadeLake processors (24 cores, 2.4GHz), 48 cores per node, hyperthreading x2, 768GB DDR4 RAM, and an internal network of Ethernet 100GbE. The OpenStack virtual machine executes the ExaMon production, which we extended with additional ones for the Kubeflow and Kubernetes pods needed by the MLOps. The computational resources available for the ExaMon monitoring systems are 300GB of RAM and 40 vCores. We also collect standard Kubernetes metrics. For implementing the MLOps framework, we used Kubernetes version 1.24 in our framework for automated deployment, scaling, and management of containerized applications. Our Kubernetes cluster has 48 vCPUs and 360 GB of RAM available. For Kubeflow, we used the canonical Charmed Kubeflow version 1.6.

We evaluated six deployment configurations to evaluate the inference pipeline’s computing time, memory, and computing cost. We started with a single-rack evaluation and scaled up to the entire supercomputer. We maintained the same frequency for prediction as training samples were aggregated with a 15-minute sampling frequency. However, we also wanted to see the maximum speed at which we can run the pipeline for the entire supercomputer and the impact of this on the MLOps and

ODA framework. We conducted experiments with two modes: 1) one with a 900-second frequency period, where the inference is run every 15 minutes, and 2) a continuous one, where the pipeline executes on streaming data without any sleep interval. Table 2, reports the frequency for the continuous conditions.

The term “One Rack” in the experiment name indicates that inference was performed for a single rack. In contrast, “All Racks” in the experiment name indicates that inference was performed on all racks (and nodes) of the HPC cluster (49 racks with 1K nodes). “Continuous” means that inference was performed continuously. This means that as soon as one inference action was completed, the next inference started without any delay. “Discrete” means that inference was performed periodically within a 900-second (or 15-minute) period. “Multiple Pods” in the experiment name indicates that the pipeline for different racks was performed in parallel by utilizing different pods. On the other hand, “One Pod” indicates that inference was performed serially, rack by rack, using just one pod. The latter configuration allows us to evaluate the proposed approach’s scalability and how the MLOps framework’s overheads introduce changes with that.

We collected several metrics to evaluate our pipeline, including data extraction latency, preprocessing time, inference computing time, and publishing results latency. These metrics are measured in seconds and presented in Table 2. We also monitored resource usage, including CPU and memory usage and the number of pods used. The corresponding metrics are summarized in Table 3, which shows resource usage for the baseline setup and the six different deployment configurations; the results are grouped into five sub-tables, reporting the resource usage for the monitoring system/ODA (“ExaMon” sub-table), Kubernetes management, Kubeflow management, user workload not including the GNN inference (“User Namespace” sub-table), and the workload due to the GNN models pipeline (“Anomaly Prediction pipeline” sub-table).

4.2.1. Computational Resources Overhead

Table 2 shows the latency for different pipeline parts. The last column reports the inference rate, measured as the number of inferences per hour each configuration achieves. The longer the pipeline stages take to execute, the lower the framework overhead the inference rate will be. In discrete pipelines, this is typically done at a fixed period of 900 seconds (15 minutes) by definition. However, in continuous pipelines, this rate depends on the processing time and latency of the pipeline. Data extraction is the most time-consuming step in the pipeline. Preprocessing only takes up 1% of the data extraction latency, and inference time is less than 1%. As long as all models proposed in this manuscript share the same input parameters, we can run multiple models in production without any extra cost.

By comparing the continuous and discrete methods, it is clear that the continuous method reduces overall latency by expanding the data extraction time. In contrast, the discrete method requires significantly higher amounts of data to be extracted from databases (12x to 100x), since the pipeline waits 15 minutes for the next sample. This is visible for the One Rack

Configuration Name	MLOps Framework Configurations			MLOps Pipeline Stage Execution Time [s]					# Inference/Hour
	Deployment	Inference Mode	Size [Rack]	Data Extraction [s]	Preprocessing [s]	Inference [s]	Publishing Results [s]	Total [s]	
One Rack, One Pod, Continuous	Single Pod	Continuous	1	8.405	0.084	0.016	0.003	8.508	423
One Rack, One Pod, Discrete	Single Pod	Discrete	1	10.213	0.115	0.014	0.002	10.344	4
All Racks, Multiple Pods, Continuous	Multiple Pods	Continuous	49	69.607	0.063	0.022	0.002	69.694	51
All Racks, Multiple Pods, Discrete	Multiple Pods	Discrete	49	31.208	0.102	0.022	0.002	31.334	4
All Racks, One Pod, Continuous	Single Pod	Continuous	49	25.783	4.697	0.303	0.062	30.845	116
All Racks, One Pod, Discrete	Single Pod	Discrete	49	51.138	4.557	0.337	0.060	56.092	4

Table 2: Processing time and latency of different deployment configuration.

Configuration Name	ExaMon				Kubernetes					
	#vcores	Mem [GB]	Net in [kB/s]	Net out [KB/s]	pods	#vcores	Mem [GB]	Net in [kB/s]	Net out [KB/s]	
Baseline	3.08	189.96	6670	6739	13	0.31	0.63	1330	864	
One Rack, One Pod, Continuous	4.47	188.7	10457	10490	13	0.35	0.63	2000	1040	
One Rack, One Pod, Discrete	3.59	189.42	9588	7832	13	0.31	0.63	1750	880	
All Racks, Multiple Pods, Continuous	7.9	189.64	18310	12570	13	0.48	0.64	4280	1870	
All Racks, Multiple Pods, Discrete	3.84	189.45	8724	8152	13	0.33	0.64	1950	848	
All Racks, One Pod, Continuous	5.87	189.51	9190	8546	13	0.31	0.63	1770	945	
All Racks, One Pod, Discrete	3.41	189.43	8686	7738	13	0.31	0.63	1900	761	
Configuration Name	Kubeflow / User Namespace				Anomaly Prediction Pipeline					
	pods	#vcores	Mem [GB]	Net in [kB/s]	Net out [KB/s]	pods	#vcores	Mem [GB]	Net in [kB/s]	Net out [KB/s]
Baseline	59/ 3	0.22 / 0.13	5.44 / 0.47	23 / 7	28 / 1	-	-	-	-	-
One Rack, One Pod, Continuous	59/ 3	0.24 / 0.14	5.41 / 0.92	39 / 42	32 / 26	1	0.13	0.45	34	26
One Rack, One Pod, Discrete	59/ 3	0.23 / 0.02	5.41 / 0.91	26 / 8	32 / 1	1	0.01	0.44	1	1
All Racks, Multiple Pods, Continuous	59/ 3	0.23 / 0.8	5.45 / 21.9	70 / 211	62 / 156	49	0.78	21.4	204	153
All Racks, Multiple Pods, Discrete	59/ 3	0.23 / 0.12	5.44 / 22.1	29 / 33	21 / 14	49	0.11	21.4	32	13
All Racks, One Pod, Continuous	59/ 3	0.23 / 0.49	5.42 / 1.54	38 / 47	32 / 12	1	0.48	0.96	39	11
All Racks, One Pod, Discrete	59/ 3	0.22 / 0.4	5.41 / 1.63	31 / 21	19 / 1	1	0.03	1.06	14	1

Table 3: HPC monitoring and MLOps framework computation resource requirements and anomaly prediction pipeline deployment overhead; the 5 main sub-tables indicate the different framework's components.

and All Racks, One Pod in Table 2. We recall that the continuous methods at each step only query the missing data and do not request the entire 15 minutes of data.

As evident in Table 2, when scaling the pipeline to all the racks of the Marconi100 supercomputers, we notice that (i) the data extraction time scales sublinearly - while increasing the data request of 49x (from one rack data to 49 racks data) the query time to the ExaMon monitoring system increases only by less than 10x, and (ii) the Multiple Pod configuration underperform w.r.t. the One Pod configuration due to the congestion in the monitoring system. This is visible by the data extraction latency, which is higher for the All Racks, Multiple Pod, Continuous than the All Racks, One Pod, Continuous case - this can be explained by the fact that even if in the case of Multiple Pods, the data extraction is done in parallel by the Pods, the requests serializes to the worker in the Monitoring Framework.

After extracting the data, the data preprocessing step requires the second most computing time, while the inference and result publishing steps take negligible time. Interestingly, using multiple pods to perform inference and other steps in parallel for racks of the supercomputer takes more time than using a single pod and performing inference rack-by-rack in a serial approach - with this latter setting, the framework can run 116 model inferences for all the nodes each hour (All Racks, One Pod, Continuous in Table 2), or that the pipeline could process 30x more compute nodes guaranteeing real-time performance - one inference every 15minutes. This result indicates that the proposed framework can scale to exascale system requirements. Moreover, being the pipeline bottleneck, the data extraction of more complex models can be afforded with the current system at a negligible cost.

To better understand the implication and cost of the pro-

posed MLOps framework in conjunction with ODA, we collected resource usage data for different parts of the monitoring system, Kubernetes, and Kubeflow without running any pipelines to determine the base load of the framework (as shown in Table 3). The different sets of deployment configurations allow an understanding of the optimal deployment configuration according to different requirements due to the HPC facility resources and pre-existing cluster configurations.

By looking at the baseline case (Baseline in Table 3), we can notice that the ExaMon ODA framework under normal operations (continuous data collection from the different sensors and dashboards) consumes 3 virtual cores (vcores) and 190GB of memory, while the MLOps framework while not processing any data analytics pipeline uses 75 pods (13 used by Kubernetes, 59 Kubeflow, 3 user namespace), 0.66 vcores and almost 7GBs of memory for its micro-services - almost the 22% more vcores and 4% more memory than the pure monitoring framework. Interestingly, when a real-time anomaly prediction pipeline is performed (All Racks, One Pod, Discrete in Table 3) for all the nodes of the Marconi100 supercomputer, the ExaMon load increases from 3.08 to 3.42. The MLOps load slightly reduces, from 0.66 vcores to 0.60 vcores due to the lower load in the user namespace micro-services and relatively negligible cost of real-time inference (0.03 vcores) - overall supporting real-time anomaly prediction in-production on the Marconi100 supercomputer costs the 30% of more vcore resources than only monitoring it. Of this 30% increase, 11% is due to an increase in the monitoring system load, while the remaining part is related to the MLOps part. The anomaly prediction pipeline accounts for less than 1% of the entire overhead, making it ready to scale to larger supercomputers, like exascale systems. When using multiple pods in a continuous approach (All racks, Multi-

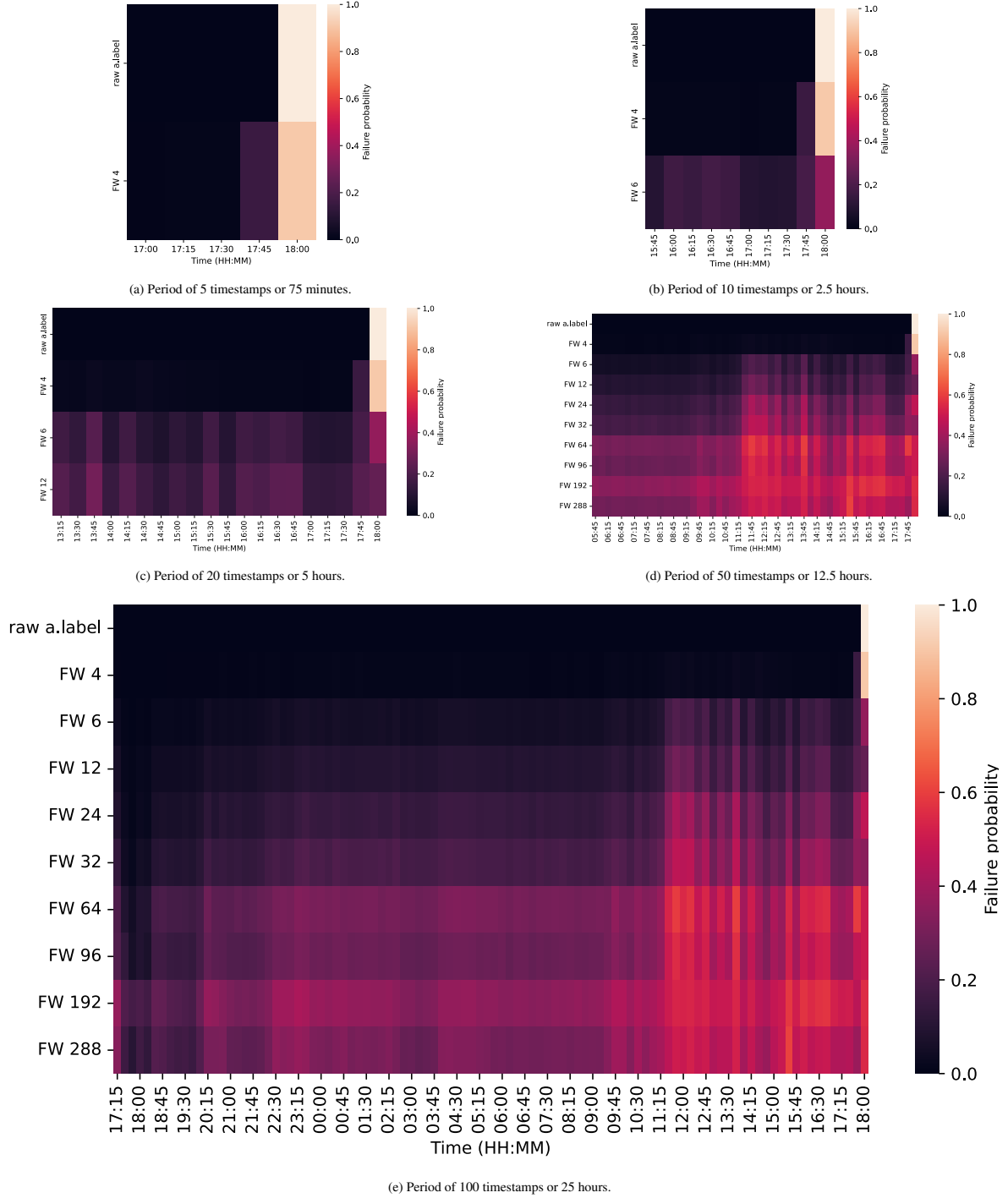


Figure 4: The same rising-edge event, observed over multiple magnifications. The last timestamp is the occurrence of an anomaly; the rest are the preceding period. Prob fw is the probability for class 1 as generated by the GNN trained with the future window with the length of fw (the duration of the future window is $f_w * 15$ minutes).

ple pods, Continuous in Table 3), the CPU load of the monitoring system increases by 2.5 times (7.9 vcores), while the load of the MLOps framework increases by 3 times (1.8 vcores).

Except for the "Supercomputer Multiple Pods Continues" setup, the CPU usage during the different configurations is very close to the base CPU usage of Kubernetes; this is a very welcomed sign, as it means that the GNN inference is not impacting too significantly the standard operations of the monitoring framework. It must also be noticed that implementing a pipeline in a continuous approach increases the write load on the KairosDB and Cassandra clusters.

4.2.2. Visualization of anomaly anticipation

Figure 4 shows the visualization available to system administrators provided by GRAAFE for a single rising edge event (transition from state 0 to state 1). This event is taken from node 240, located in compute rack 12. The anomaly occurred on 7/5/2022 at 6:45. The event with different lengths of the preceding periods is plotted in Fig. 4. The visualization reports the (normalized) probability for the anomaly (class 1) trained with different future windows.

First, we focus on the shortest prediction window classifier - the classifier with the future window of 4 timesteps of 1 hour in the future. This classifier anticipated the anomaly but only with one timestep of anticipation (15 minutes). Still, it produces a clear and distinctive anomaly signal before the rising edge. Moving on to longer observation windows like 2.5 and 5 hours, we see that classifiers with longer observation windows also anticipate the anomaly correctly. According to expectations, the classifier trained to detect anomalies one hour in advance (prob4) predicts no anomalies in this larger observation window. With much larger time windows (from 7.5 to 25 hours) all other classifiers produce the expected spike in probability (see Fig. 4e), with density increasing closer to the actual anomaly.

4.2.3. Financial impact of anomaly anticipation

Deployment of the anomaly prediction model in real-life operations consists of estimating the positive aspects and benefits of the deployment against the potential negative aspects and costs. To do this, we have developed a model, which is included in the code repository, that models the benefits of the model against the costs associated with mispredictions. We are combining the model with some assumptions about the operational aspect of a typical supercomputer and the additional results of our predictive models, as described in the additional results in the code repository. We are using the optimal threshold for the classifier and the true positive and false positive rates, as reported in the additional results. The general equation of the cost-benefit model is:

$$\text{Benefit} = \text{Cost of False Negatives} - \text{Cost of False Positives}$$

Compared to the case where no predictive model is deployed, the deployment of the GRAAFE framework results in no additional costs of false negatives. For this reason, the cost of false negatives is modeled as zero in our model. The cost of

false positives, however, is associated with the time that the system administrators waste analyzing false positive signals. We are modeling it as an unnecessary action cost. Unnecessary action cost is calculated as:

$$\begin{aligned} \text{Unnecessary action cost} &= \text{false positive probability} \\ &\quad \times \text{analysis duration} \\ &\quad \times \text{hourly pay of system administrator} \end{aligned}$$

The unnecessary action cost increases for larger future windows as the false positive rate increases and the overall anticipation accuracy decreases.

Hours	Optimistic	Conservative	Pessimistic
0	0.0	0.0	0.0
1	0.7	0.3	0.0
3	0.8	0.6	0.0
6	-	-	0.3
8	0.8	0.8	0.8
72	0.8	0.8	0.8

Table 4: Three scenarios are defined based on the probability that the system administrators can prevent an anomaly if given a signal within a future window (in hours).

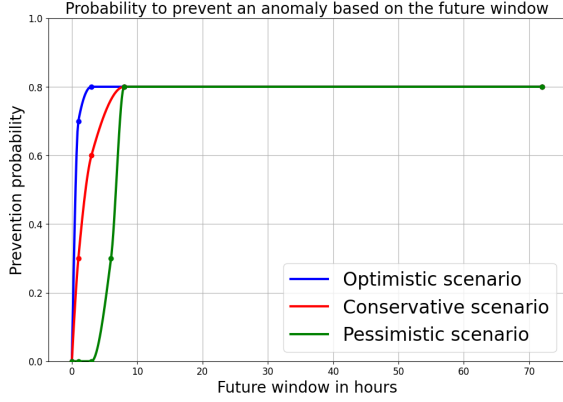
The benefit of the model development is estimated as the expected value of the deployment benefit:

$$\begin{aligned} \text{Benefit} &= \text{probability to detect failure (FW)} \\ &\quad \times \text{probability to prevent failure (FW)} \\ &\quad \times \text{cost of failure} \end{aligned}$$

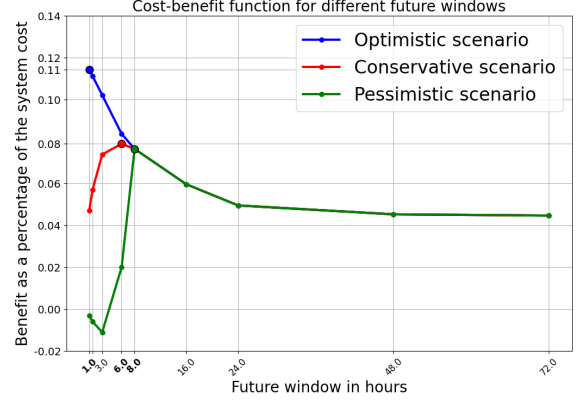
The probability of *detecting failure* is a characteristic of the predictive model, and it equals the true positive rate. For the analysis presented in this section, we are using the true positive rate of the per-rack GNN model, as recorded in the additional results. The cost of failure is related to the opportunity cost of the HPC system downtime. Our analysis estimates it to be a fraction of the overall cost of the HPC system.

The probability of *preventing failure* is a characteristic of the organizational specifics of each data center and its system administration team. Since we do not have accurate organizational information for the Marconi 100 supercomputer, we have prepared three scenarios demonstrating different system administration teams. Based on the warning period (in hours), the system administrator teams prevent the anomaly with different probabilities as depicted in table 4. Optimistic scenario depicts a very effective system administration team that can prevent 70% of anomalies with a warning window of one hour and 80% with a warning period of three or more hours. Conservative scenario depicts a conservative estimation of the system administration team's effectiveness. System administrators can only prevent 30% of the anomalies with a prediction window of one hour. Pessimistic scenario is an overly pessimistic option where system administrators cannot prevent anomalies if they have a warning period of less than six hours.

Based on the important points of each scenario, a smooth (double differentiable) function is fitted over them. This gives



(a) The prevention probability function models the probability that the system administrators will be able to prevent the anomaly if given the anomaly signal in a given future window.



(b) The cost-benefit function models the net benefit (benefit minus the cost) of the model adoption for different future windows, expressed as the percentage of the total HPC system cost.

Figure 5: Depending on the scenario and the associated probability of preventing the anomaly, given a specific warning window, the projected benefit is achieved by deploying the predictive system at different future windows.

us the probabilities for every other future window. The function is based on the assumption of being monotonic positive as the prevention probability cannot decrease. Based on three scenarios, we get three different anomaly prevention probability functions as depicted in Figure 5a.

Given the anomaly prevention probability function, we can estimate the cost-benefit of deploying the model in each scenario. The cost-benefit, measured in the percentage of the overall system cost, is depicted in Figure 5b. In the optimistic scenario, the maximum benefit is achieved for the one-hour future prediction window. For the conservative scenario, the maximum is achieved for the prediction window of 3 hours. Interestingly, for the conservative scenario - the cost-benefit is negative for future windows up to 3 hours. This is because the system administrators cannot prevent the anomalies but would still have to react to the false positive signal (and incur the associated costs). As seen in Figure 5, the model’s benefit and the optimal deployment strategy (size of the future window) depend strongly on the organizational characteristics of each data center.

Estimating the organizational characteristics of the individual data center is beyond the scope of this paper. For this reason, we have prepared a computational model included in the code repository (https://gitlab.com/ecs-lab/GRAAFE/-/blob/main/Cost_benefit.ipynb) that takes the operational parameters as input and outputs the cost-benefit function for the defined specific scenario. This allows potential adopters of our framework to decide on the optimal deployment strategy based on their organizational reality and needs.

5. Conclusion

In this paper, we proposed two contributions: 1) the first anomaly prediction approach for HPC systems explicitly exploiting the underlying proximity structure of computing nodes;

2) an MLOps framework for integrating these rack-based models with monitoring infrastructure and continuously obtaining predictions on live data. The graph structure significantly influences the performance of anomaly anticipation models for supercomputers. Our GNN approach has enabled us to surpass all previously known ML methods in this domain; in particular, we thoroughly demonstrate how we can not only detect failures (as already done in previous work) but predict such failures with several hours of anticipation.

The MLOps framework has been exhaustively evaluated in deployment configurations to measure the overhead caused by the Kubernetes pipeline. In the current setup, the impact on the supercomputer itself is negligible, as the MLOps runs on the same cloud infrastructure (residing in the same HPC facility hosting the supercomputer) where the monitoring infrastructure is deployed. We have also explored different deployment setups, highlighting how different solutions can be adopted, granting greater flexibility while satisfying real-time requirements. Moreover, the additional incurred computational overhead w.r.t. the monitoring infrastructure (which we can assume is present in most supercomputers nowadays) is only 30% more CPU resources and less than 5% more RAM.

Acknowledgments

This research was partly supported by the EuroHPC EU Regale project (g.a. 956560), the HE EU DECICE project (g.a. 101092582), the HE EU Graph-Massivizer project (g.a. 101093202), the HE EU DECICE project (g.a. 101092582), the SPOKE 1: Future HPC & Big Data by PNRR, and the EuroHPC EU Pilot for exascale EUPEX (g.a. 101033975). We also thank CINECA for the collaboration and access to their machines.

References

- [1] J. J. Dongarra, H. W. Meuer, E. Strohmaier, 29th top500 Supercomputer Sites, Tech. rep., Top500.org (Nov. 1994).
- [2] A. Borghesi, M. Molan, M. Milano, A. Bartolini, Anomaly detection and anticipation in high performance computing systems, *IEEE Transactions on Parallel and Distributed Systems* 33 (4) (2022) 739–750. doi:<https://doi.org/10.1109/TPDS.2021.3082802>.
- [3] M. Molan, A. Borghesi, D. Cesarini, L. Benini, A. Bartolini, Ruad: Unsupervised anomaly detection in hpc systems, *Future Generation Computer Systems* 141 (2023) 542–554. doi:<https://doi.org/10.1016/j.future.2022.12.001>.
- [4] D. Jauk, D. Yang, M. Schulz, Predicting faults in high performance computing systems: An in-depth survey of the state-of-the-practice, in: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '19, Association for Computing Machinery, New York, NY, USA, 2019*. doi:<https://doi.org/10.1145/3295500.3356185>. URL <https://doi.org/10.1145/3295500.3356185>
- [5] Q. Guan, Z. Zhang, S. Fu, Proactive failure management by integrated unsupervised and semi-supervised learning for dependable cloud systems, in: *2011 Sixth International Conference on Availability, Reliability and Security, 2011*, pp. 83–90. doi:[10.1109/ARES.2011.20](https://doi.org/10.1109/ARES.2011.20).
- [6] B. Nie, J. Xue, S. Gupta, T. Patel, C. Engelmann, E. Smirni, D. Tiwari, Machine learning models for gpu error prediction in a large scale hpc system, in: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2018*, pp. 95–106. doi:[10.1109/DSN.2018.00022](https://doi.org/10.1109/DSN.2018.00022).
- [7] M. Ott, W. Shin, N. Bourassa, T. Wilde, S. Ceballos, M. Romanus, N. Bates, Global experiences with hpc operational data measurement, collection and analysis, in: *2020 IEEE International Conference on Cluster Computing (CLUSTER), 2020*, pp. 499–508. doi:[10.1109/CLUSTER49012.2020.00071](https://doi.org/10.1109/CLUSTER49012.2020.00071).
- [8] P. Matri, P. Carns, R. Ross, A. Costan, M. S. Pérez, G. Antoniu, Slog: Large-scale logging middleware for hpc and big data convergence, in: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2018*, pp. 1507–1512.
- [9] W. Khan, D. De Chiara, A.-L. Kor, M. Chinnici, Exploratory data analysis for data center energy management, in: *Proceedings of the Thirteenth ACM International Conference on Future Energy Systems, 2022*, pp. 571–580.
- [10] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph neural networks: A review of methods and applications (2018). doi:[10.48550/ARXIV.1812.08434](https://doi.org/10.48550/ARXIV.1812.08434). URL <https://arxiv.org/abs/1812.08434>
- [11] A. Netti, M. Muller, et al., Dcdh wintemute: Enabling online and holistic operational data analytics on hpc systems, in: *Proc. of the 29th International Symposium on High-Performance Parallel and Distributed Computing, ACM, New York, NY, USA, 2020*, p. 101–112.
- [12] F. Beneventi, A. Bartolini, C. Cavazzoni, L. Benini, Continuous learning of hpc infrastructure models using big data analytics and in-memory processing tools, in: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, IEEE, 2017*, pp. 1038–1043.
- [13] A. Bartolini, F. Beneventi, et al., Paving the way toward energy-aware and automated datacentre, in: *Proceedings of the 48th International Conference on Parallel Processing: Workshops, 2019*, pp. 1–8.
- [14] Wikipedia, CINECA — Wikipedia, the free encyclopedia, <http://en.wikipedia.org/w/index.php?title=CINECA&oldid=954269846>, [Online; accessed 04-December-2021] (2021).
- [15] D. Milojicic, P. Faraboschi, N. Dube, D. Roweth, Future of hpc: Diversifying heterogeneity, in: *2021 Design, Automation Test in Europe Conference Exhibition (DATE), 2021*, pp. 276–281. doi:[10.23919/DATES1398.2021.9474063](https://doi.org/10.23919/DATES1398.2021.9474063).
- [16] A. Gainaru, M.-S. Bouguerra, F. Cappello, M. Snir, W. T. C. Kramer, Navigating the blue waters : Online failure prediction in the petascale era, 2013. URL <https://api.semanticscholar.org/CorpusID:16874101>
- [17] S. Lu, B. Luo, T. Patel, Y. Yao, D. Tiwari, W. Shi, Making disk failure predictions smarter!, in: *Proceedings of the 18th USENIX Conference on File and Storage Technologies, FAST'20, USENIX Association, USA, 2020*, p. 151–168.
- [18] Y. Liu, Y. Guan, T. Jiang, K. Zhou, H. Wang, G. Hu, J. Zhang, W. Fang, Z. Cheng, P. Huang, Spae: Lifelong disk failure prediction via end-to-end gan-based anomaly detection with ensemble update, *Future Generation Computer Systems* 148 (2023) 460–471. doi:<https://doi.org/10.1016/j.future.2023.05.020>. URL <https://www.sciencedirect.com/science/article/pii/S0167739X23002030>
- [19] A. Borghesi, C. Di Santi, M. Molan, M. S. Ardebili, A. Mauri, M. Guarasi, D. Galetti, M. Cestari, F. Barchi, L. Benini, F. Beneventi, A. Bartolini, M100 exadata: a data collection campaign on the cineca's marconi100 tier-0 supercomputer, *Scientific Data* 10 (1) (2023) 288. doi:[10.1038/s41597-023-02174-3](https://doi.org/10.1038/s41597-023-02174-3). URL <https://doi.org/10.1038/s41597-023-02174-3>
- [20] T. P. Carvalho, F. A. Soares, R. Vita, R. d. P. Francisco, J. P. Basto, S. G. Alcalá, A systematic literature review of machine learning methods applied to predictive maintenance, *Computers & Industrial Engineering* 137 (2019) 106024.
- [21] S. Behera, A. Choubey, C. S. Kanani, Y. S. Patel, R. Misra, A. Sillitti, Ensemble trees learning based improved predictive maintenance using iiot for turbofan engines, in: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC '19, Association for Computing Machinery, New York, NY, USA, 2019*, p. 842–850. doi:[10.1145/3297280.3297363](https://doi.org/10.1145/3297280.3297363). URL <https://doi.org/10.1145/3297280.3297363>
- [22] J. Zhang, R. Gardner, I. Vukotic, Anomaly detection in wide area network meshes using two machine learning algorithms, *Future Generation Computer Systems* 93 (2019) 418–426. doi:<https://doi.org/10.1016/j.future.2018.07.023>. URL <https://www.sciencedirect.com/science/article/pii/S0167739X18302267>
- [23] F. Monti, D. Boscaini, et al., Geometric deep learning on graphs and manifolds using mixture model cnns, in: *Proc. of the IEEE conference on computer vision and pattern recognition, 2017*, pp. 5115–5124.
- [24] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph neural networks: A review of methods and applications, *AI open* 1 (2020) 57–81.
- [25] A. Chaudhary, H. Mittal, A. Arora, Anomaly detection using graph neural networks, in: *2019 international conference on ML, big data, cloud and parallel computing, IEEE, 2019*, pp. 346–350.
- [26] A. Deng, B. Hooi, Graph neural network-based anomaly detection in multivariate time series, in: *Proceedings of the AAAI conference on artificial intelligence, Vol. 35, 2021*, pp. 4027–4035.
- [27] Y. Song, R. Xin, et al., Identifying performance anomalies in fluctuating cloud environments: a robust correlative-gnn-based explainable approach, *Future Generation Computer Systems* (2023).
- [28] S. Ghiasvand, F. M. Ciorba, Anomaly detection in high performance computers: A vicinity perspective, in: *18th International Symposium on Parallel and Distributed Computing, IEEE, 2019*, pp. 112–120.
- [29] Y. Song, R. Xin, P. Chen, R. Zhang, J. Chen, Z. Zhao, Identifying performance anomalies in fluctuating cloud environments: A robust correlative-gnn-based explainable approach, *Future Generation Computer Systems* 145 (2023) 77–86. doi:<https://doi.org/10.1016/j.future.2023.03.020>. URL <https://www.sciencedirect.com/science/article/pii/S0167739X23000973>
- [30] A. Netti, M. Ott, C. e. a. Guillen, Operational data analytics in practice: experiences from design to deployment in production hpc environments, *Parallel Computing* 113 (2022) 102950.
- [31] A. Borghesi, A. Burrello, A. Bartolini, Examon-x: a predictive maintenance framework for automatic monitoring in industrial iot systems, *IEEE Internet of Things Journal* (2021).
- [32] D. Kreuzberger, N. Kühl, S. Hirschl, Machine learning operations (mlops): Overview, definition, and architecture, *arXiv preprint arXiv:2205.02302* (2022).
- [33] S. Kim, K. Choi, H.-S. Choi, et al., Towards a rigorous evaluation of time-series anomaly detection, in: *Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 36, 2022*, pp. 7194–7201.

Bibliography

- [1] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md. Rafiqul Islam. “A survey of anomaly detection techniques in financial domain”. In: *Future Generation Computer Systems* 55 (2016), pp. 278–288. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2015.01.001>. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X15000023>.
- [2] Burak Aksar et al. “E2EWatch: An End-to-End Anomaly Diagnosis Framework for Production HPC Systems”. In: *European Conference on Parallel Processing*. Springer. 2021, pp. 70–85.
- [3] Burak Aksar et al. “E2EWatch: An End-to-End Anomaly Diagnosis Framework for Production HPC Systems”. In: *Euro-Par 2021: Parallel Processing*. Ed. by Leonel Sousa, Nuno Roma, and Pedro Tomás. Cham: Springer International Publishing, 2021, pp. 70–85. ISBN: 978-3-030-85665-6.
- [4] Burak Aksar et al. “Proctor: A semi-supervised performance anomaly diagnosis framework for production hpc systems”. In: *High Performance Computing: 36th International Conference, ISC High Performance 2021, Virtual Event, June 24–July 2, 2021, Proceedings* 36. Springer. 2021, pp. 195–214.
- [5] Burak Aksar et al. “Proctor: A Semi-Supervised Performance Anomaly Diagnosis Framework for Production HPC Systems”. In: *High Performance Computing: 36th International Conference, ISC High Performance 2021, Virtual Event, June 24 – July 2, 2021, Proceedings*. Ed. by Bradford L. Chamberlain et al. Cham: Springer International Publishing, 2021, pp. 195–214. ISBN: 978-3-030-78712-7. DOI: [10.1007/978-3-030-78713-4_11](https://doi.org/10.1007/978-3-030-78713-4_11). URL: https://doi.org/10.1007/978-3-030-78713-4_11.
- [6] Afroj Alam, Mohd Muqem, and Sultan Ahmad. “Comprehensive review on Clustering Techniques and its application on High Dimensional Data”. In: *International Journal of Computer Science & Network Security* 21.6 (2021), pp. 237–244.
- [7] Pedro Amorim, Fredrik Eng-Larsson, and Catarina Pinto. “The cost of a broken promise: Understanding and mitigating the impact of failure in online retail”. In: *SSRN Electronic Journal* (2021). DOI: [10.2139/ssrn.3855425](https://doi.org/10.2139/ssrn.3855425). URL: <https://doi.org/10.2139/ssrn.3855425>.

- [8] arXiv. "Sustainability in HPC: Vision and Opportunities". In: *arXiv* (2024). URL: <https://arxiv.org/abs/2309.13473>.
- [9] Priati Assiroj et al. "The Form of High-Performance Computing: A Survey". In: *IOP conference series* 662.5 (Nov. 2019), pp. 052002–052002. DOI: 10.1088/1757-899x/662/5/052002.
- [10] Philip Virgil Astillo et al. "Federated intelligence of anomaly detection agent in IoTMD-enabled Diabetes Management Control System". In: *Future Generation Computer Systems* 128 (2022), pp. 395–405. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2021.10.023>. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X21004192>.
- [11] Mohammed Badawy et al. "A survey on exploring key performance indicators". In: *Future Computing and Informatics Journal* 1.1 (2016), pp. 47–52. ISSN: 2314-7288. DOI: <https://doi.org/10.1016/j.fcij.2016.04.001>. URL: <https://www.sciencedirect.com/science/article/pii/S2314728816300034>.
- [12] Dor Bank, Noam Koenigstein, and Raja Giryes. "Autoencoders". In: *CoRR* abs/2003.05991 (2020). arXiv: 2003.05991. URL: <https://arxiv.org/abs/2003.05991>.
- [13] Wolfgang Barth. *Nagios: System and network monitoring*. No Starch Press, 2008.
- [14] Andrea Bartolini, Francesco Beneventi, and et al. "Paving the way toward energy-aware and automated datacentre". In: *Proceedings of the 48th International Conference on Parallel Processing: Workshops*. 2019, pp. 1–8.
- [15] Andrea Bartolini et al. "Paving the Way Toward Energy-Aware and Automated Datacentre". In: *Proceedings of the 48th International Conference on Parallel Processing: Workshops*. ICPP 2019. New York, NY, USA: Association for Computing Machinery, 2019. ISBN: 9781450371964. DOI: 10.1145/3339186.3339215. URL: <https://doi.org/10.1145/3339186.3339215>.
- [16] Elisabeth Baseman et al. "Interpretable Anomaly Detection for Monitoring of High Performance Computing Systems". In: *Outlier Definition, Detection, and Description on Demand Workshop at ACM SIGKDD*. San Francisco (Aug 2016). 2016, pp. 1–27.
- [17] Sanzio Bassini et al. *Cineca HPC Report 2023-2024*. Annual Report. Via Magnanelli 6/3, 40033 - Casalecchio di Reno (BO) - Italy: Cineca Consorzio Interuniversitario, July 2024. URL: www.hpc.cineca.it.
- [18] Sourajit Behera et al. "Ensemble Trees Learning Based Improved Predictive Maintenance Using IIoT for Turbofan Engines". In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. SAC '19. New York, NY, USA: Association for Computing Machinery, 2019, 842–850. ISBN:

9781450359337. DOI: 10.1145/3297280.3297363. URL: <https://doi.org/10.1145/3297280.3297363>.
- [19] Genrich Belitskii et al. *Matrix norms and their applications*. Vol. 36. Birkhäuser, 2013.
- [20] Keren Bergman et al. "Exascale computing study: Technology challenges in achieving exascale systems". In: *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep 15* (2008).
- [21] Liron Bergman and Yedid Hoshen. "Classification-based anomaly detection for general data". In: *arXiv preprint arXiv:2005.02359* (2020).
- [22] Neva Beske. *UG3.2: MARCONI100 UserGuide*. Accessed: 2020-08-17. 2020. URL: <https://wiki.u-gov.it/confluence/pages/viewpage.action?pageId=336727645>.
- [23] I. Boixaderas, D. Zivanovic, and et al. "Cost-Aware Prediction of Uncorrected DRAM Errors in the Field". In: *2020 SC20: International Conference for HPC, Networking, Storage and Analysis (SC)*. Los Alamitos, CA, USA: IEEE Comp. Soc., Sept. 2020, pp. 1–15.
- [24] A. Borghesi, A. Bartolini, and et al. "Anomaly detection using autoencoders in HPC systems". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2019, pp. 24–32.
- [25] A. Borghesi, A. Libri, and et al. "Online anomaly detection in hpc systems". In: *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems*. IEEE. 2019, pp. 229–233.
- [26] 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* (2021).
- [27] Andrea Borghesi, Michela Milano, and Luca Benini. "Frequency Assignment in High Performance Computing Systems". In: *International Conference of the Italian Association for Artificial Intelligence*. Springer. 2019, pp. 151–164.
- [28] Andrea Borghesi et al. "A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems". In: *Engineering Applications of Artificial Intelligence* 85 (2019), pp. 634–644.
- [29] Andrea Borghesi et al. "A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems". In: *Engineering Applications of Artificial Intelligence* 85 (2019), pp. 634–644. ISSN: 0952-1976. DOI: <https://doi.org/10.1016/j.engappai.2019.07.008>. URL: <https://www.sciencedirect.com/science/article/pii/S0952197619301721>.

- [30] Andrea Borghesi et al. "Anomaly Detection and Anticipation in High Performance Computing Systems". In: *IEEE Transactions on Parallel and Distributed Systems* 33.4 (2022), pp. 739–750. DOI: 10.1109/TPDS.2021.3082802.
- [31] Andrea Borghesi et al. "M100 ExaData: a data collection campaign on the CINECA's Marconi100 Tier-0 supercomputer". In: *Scientific Data* 10 (2023), p. 288.
- [32] Andrea Borghesi et al. "Scheduling-based power capping in high performance computing systems". In: *Sustainable Computing: Informatics and Systems* 19 (Sept. 2018), pp. 1–13. DOI: 10.1016/j.suscom.2018.05.007. URL: <https://www.sciencedirect.com/science/article/pii/S2210537917302317>.
- [33] Avishek Bose et al. "HPCGCN: A Predictive Framework on High Performance Computing Cluster Log Data Using Graph Convolutional Networks". In: *2021 IEEE International Conference on Big Data (Big Data)*. IEEE. 2021, pp. 4113–4118.
- [34] GLENN W. BRIER. "VERIFICATION OF FORECASTS EXPRESSED IN TERMS OF PROBABILITY". In: *Monthly Weather Review* 78.1 (1950), pp. 1–3. DOI: [https://doi.org/10.1175/1520-0493\(1950\)078<0001:V0FEIT>2.0.CO;2](https://doi.org/10.1175/1520-0493(1950)078<0001:V0FEIT>2.0.CO;2).
- [35] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: *arXiv preprint arXiv:2005.14165* (2020). URL: <https://arxiv.org/abs/2005.14165>.
- [36] Sathya Bursic, Alessandro D'Amelio, and Vittorio Cuculo. *Anomaly Detection From Log Files Using Unsupervised Deep Learning*. Sept. 2019.
- [37] Maria Carla Calzarossa, Luisa Massari, and Daniele Tessera. "Workload characterization: A survey revisited". In: *ACM Computing Surveys (CSUR)* 48.3 (2016), pp. 1–43.
- [38] L. Le Cam. "Maximum Likelihood: An Introduction". In: *International Statistical Review / Revue Internationale de Statistique* 58.2 (1990), pp. 153–171. ISSN: 03067734, 17515823. URL: <http://www.jstor.org/stable/1403464>.
- [39] Thyago P Carvalho et al. "A systematic literature review of machine learning methods applied to predictive maintenance". In: *Computers & Industrial Engineering* 137 (2019), p. 106024.
- [40] Francesco Cauteruccio et al. "A framework for anomaly detection and classification in Multiple IoT scenarios". In: *Future Generation Computer Systems* 114 (2021), pp. 322–335. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2020.08.010>. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X19335253>.

- [41] Anshika Chaudhary, Himangi Mittal, and Anuja Arora. "Anomaly detection using graph neural networks". In: *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)*. IEEE. 2019, pp. 346–350.
- [42] Xiaomeng Chen et al. "Runtime prediction of high-performance computing jobs based on ensemble learning". In: (June 2020). DOI: 10.1145/3407947.3407968. URL: <https://dl.acm.org/doi/10.1145/3407947.3407968>.
- [43] Cineca. *Leonardo Pre-exascale Supercomputer*. <https://leonardo-supercomputer.cineca.eu/hpc-system>. Accessed: 2024-08-15. URL: <https://www.top500.org/system/180128>.
- [44] Cineca. *Marconi Tier-0 system*. <https://www.hpc.cineca.it/systems/hardware/marconi100>. Accessed: 2024-08-15. URL: <https://www.top500.org/system/179845>.
- [45] John Connolly. "Book Review : The Supercomputer Era". In: *The international journal of supercomputer applications* 2.1 (Mar. 1988), pp. 95–96. DOI: 10.1177/109434208800200108. URL: <https://journals.sagepub.com/doi/10.1177/109434208800200108>.
- [46] *Contemporary High Performance Computing | From Petascale toward Exasca*. Apr. 2018. DOI: 10.1201/9781351104005. URL: <https://www.taylorfrancis.com/books/edit/10.1201/9781351104005/contemporary-high-performance-computing-jeffrey-vetter>.
- [47] Motivair Corporation. "Understanding End-to-End Liquid Cooling for HPC & AI". In: *Motivair Corporation* (2022). URL: <https://www.motivaircorp.com/news/understanding-end-to-end-liquid-cooling-for-hpc-ai>.
- [48] Tommy Dang, Ngan Nguyen, and Yong Chen. "HiperView: real-time monitoring of dynamic behaviors of high-performance computing centers". In: *The Journal of Supercomputing* 77.10 (2021), pp. 11807–11826.
- [49] Mohamed Dani, Henri Doreau, and Samantha Alt. "K-means Application for Anomaly Detection and Log Classification in HPC". In: *Lecture Notes in Computer Science book series (LNAI, volume 10351)*. June 2017, pp. 201–210. ISBN: 978-3-319-60044-4. DOI: 10.1007/978-3-319-60045-1_23.
- [50] DatacenterDynamics. "2023 second-largest investment year in decade for data centers - report". In: *DatacenterDynamics* (2023). URL: <https://www.datacenterdynamics.com/en/news/2023-second-largest-investment-year-decade-data-centers-report/>.
- [51] Burgess Davis and David McDonald. "An elementary proof of the local central limit theorem". In: *Journal of Theoretical Probability* 8.3 (1995), pp. 693–702.

- [52] Ailin Deng and Bryan Hooi. "Graph neural network-based anomaly detection in multivariate time series". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 5. 2021, pp. 4027–4035.
- [53] J. J. Dongarra, H. W. Meuer, and E. Strohmaier. *29th TOP500 Supercomputer Sites*. Tech. rep. Top500.org, Nov. 1994.
- [54] Jack Dongarra. "Report on the Fujitsu Fugaku system". In: *University of Tennessee-Knoxville Innovative Computing Laboratory, Tech. Rep. ICLUT-20-06* (2020).
- [55] Min Du et al. "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, 1285–1298. ISBN: 9781450349468. DOI: 10.1145/3133956.3134015. URL: <https://doi.org/10.1145/3133956.3134015>.
- [56] Damien Fourure et al. "Anomaly Detection: How to Artificially Increase your F1-Score with a Biased Evaluation Protocol". In: *CoRR abs/2106.16020* (2021). arXiv: 2106.16020. URL: <https://arxiv.org/abs/2106.16020>.
- [57] Song Fu et al. "A re-optimized deep auto-encoder for gas turbine unsupervised anomaly detection". In: *Engineering Applications of Artificial Intelligence* 101 (2021), p. 104199. ISSN: 0952-1976. DOI: <https://doi.org/10.1016/j.engappai.2021.104199>. URL: <https://www.sciencedirect.com/science/article/pii/S0952197621000464>.
- [58] Ana Gainaru et al. "Navigating the Blue Waters : Online Failure Prediction in the Petascale Era". In: 2013. URL: <https://api.semanticscholar.org/CorpusID:16874101>.
- [59] M. Gamell, K. Teranishi, and et al. "Modeling and Simulating Multiple Failure Masking Enabled by Local Recovery for Stencil-Based Applications at Extreme Scales". In: *IEEE Transactions on Parallel and Distributed Systems* 28.10 (2017).
- [60] Marta Garcia-Gasulla and Brian JN Wylie. "Performance Optimisation and Productivity for EU HPC Centres of Excellence (and European Parallel Application Developers Preparing for Exascale): Best Practice for Efficient and Scalable Application Performance". In: *Platform for Advanced Scientific Computing (PASC) Conference*. FZJ-2022-00887. Jülich Supercomputing Center. 2021.
- [61] Azul Garza, Cristian Challu, and Max Mergenthaler-Canseco. "TimeGPT-1". In: *arXiv preprint arXiv:2310.03589* (2023). URL: <https://arxiv.org/abs/2310.03589>.

- [62] Siavash Ghiasvand and Florina M Ciorba. "Anomaly detection in high performance computers: A vicinity perspective". In: *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE. 2019, pp. 112–120.
- [63] Alicia Golden et al. "Generative AI Beyond LLMs: System Implications of Multi-Modal Generation". In: May 2024, pp. 257–267. DOI: 10.1109/ISPASS61541.2024.00032.
- [64] Bruno Guindani et al. "Exploring the Utility of Graph Methods in HPC Thermal Modeling". In: *Companion of the 15th ACM/SPEC International Conference on Performance Engineering*. ICPE '24 Companion. London, United Kingdom: Association for Computing Machinery, 2024, 106–111. ISBN: 9798400704451. DOI: 10.1145/3629527.3652895. URL: <https://doi.org/10.1145/3629527.3652895>.
- [65] F Iannone et al. "MARCONI-FUSION: The new high performance computing facility for European nuclear fusion modelling". In: *Fusion Engineering and Design* 129 (2018), pp. 354–358.
- [66] G. Iuhasz and D. Petcu. "Monitoring of Exascale data processing". In: *2019 IEEE International Conference on Advanced Scientific Computing (ICASC)*. 2019, pp. 1–5. DOI: 10.1109/ICASC48083.2019.8946279.
- [67] Grafika Jati et al. "AutoGrAN: Autonomous Vehicle LiDAR Contaminant Detection using Graph Attention Networks". In: *Companion of the 15th ACM/SPEC International Conference on Performance Engineering*. ICPE '24 Companion. London, United Kingdom: Association for Computing Machinery, 2024, 112–119. ISBN: 9798400704451. DOI: 10.1145/3629527.3652896. URL: <https://doi.org/10.1145/3629527.3652896>.
- [68] Grafika Jati et al. "LIDAROC: Realistic LiDAR Cover Contamination Dataset for Enhancing Autonomous Vehicle Perception Reliability". In: *IEEE Sensors Letters* 8.9 (2024), pp. 1–4. DOI: 10.1109/LSENS.2024.3434624.
- [69] Grafika Jati et al. "TinyLid: a RISC-V accelerated Neural Network For LiDAR Contaminant Classification in Autonomous Vehicle". In: *Proceedings of the 21st ACM International Conference on Computing Frontiers*. CF '24. Ischia, Italy: Association for Computing Machinery, 2024, 249–257. ISBN: 9798400705977. DOI: 10.1145/3649153.3649201. URL: <https://doi.org/10.1145/3649153.3649201>.
- [70] David Jauk, Dai Yang, and Martin Schulz. "Predicting Faults in High Performance Computing Systems: An in-Depth Survey of the State-of-the-Practice". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. ISBN: 9781450362290. DOI: 10.1145/3295500.3356185. URL: <https://doi.org/10.1145/3295500.3356185>.

- [71] Morris A. Jette, Andy B. Yoo, and Mark Grondona. "SLURM: Simple Linux Utility for Resource Management". In: *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*. Springer-Verlag, 2002.
- [72] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. 3rd. Available online as a draft. Pearson, Draft 2023. URL: <https://web.stanford.edu/~jurafsky/slp3/>.
- [73] Junaid Ahmed Khan et al. "ExaQuery: Proving Data Structure to Unstructured Telemetry Data in Large-Scale HPC". In: *Companion of the 15th ACM/SPEC International Conference on Performance Engineering*. ICPE '24 Companion. London, United Kingdom: Association for Computing Machinery, 2024, 127–134. ISBN: 9798400704451. DOI: 10.1145/3629527.3652898. URL: <https://doi.org/10.1145/3629527.3652898>.
- [74] Wania Khan et al. "Exploratory data analysis for data center energy management". In: *Proceedings of the Thirteenth ACM International Conference on Future Energy Systems*. 2022, pp. 571–580.
- [75] Siwon Kim et al. "Towards a Rigorous Evaluation of Time-series Anomaly Detection". In: *CoRR abs/2109.05257 (2021)*. arXiv: 2109.05257. URL: <https://arxiv.org/abs/2109.05257>.
- [76] Siwon Kim et al. "Towards a rigorous evaluation of time-series anomaly detection". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 7. 2022, pp. 7194–7201.
- [77] Peter Kogge and David R. Resnick. *Yearly update: exascale projections for 2013*. Oct. 2013. DOI: 10.2172/1104707.
- [78] MIT Lincoln Laboratory. "AI models are devouring energy. Tools to reduce consumption are here, if data centers will adopt." In: *MIT Lincoln Laboratory* (2024). URL: <https://www.ll.mit.edu/news/ai-models-devouring-energy-tools-reduce-consumption-are-here-if-data-centers-will-adopt>.
- [79] Grafana Labs. *Grafana: The open observability platform*. URL: <https://grafana.com/> (visited on 04/29/2020).
- [80] Ki Bum Lee, Sejune Cheon, and Chang Ouk Kim. "A convolutional neural network for fault classification and diagnosis in semiconductor manufacturing processes". In: *IEEE Transactions on Semiconductor Manufacturing* 30.2 (2017), pp. 135–142.
- [81] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning". In: *Journal of Machine Learning Research* 18.17 (2017), pp. 1–5.

- [82] Xiaopeng Li et al. "Learning latent superstructures in variational autoencoders for deep multidimensional clustering". In: *arXiv preprint arXiv:1803.05206* (2018).
- [83] Benjamin Lindemann et al. "A survey on long short-term memory networks for time series prediction". In: *Procedia CIRP* 99 (2021), pp. 650–655.
- [84] Sidi Lu et al. "Making Disk Failure Predictions SMARTer!" In: *Proceedings of the 18th USENIX Conference on File and Storage Technologies*. FAST'20. USA: USENIX Association, 2020, 151–168. ISBN: 9781939133120.
- [85] Preeti Malakar et al. "Benchmarking Machine Learning Methods for Performance Modeling of Scientific Applications". In: (Nov. 2018). DOI: 10.1109/pmbs.2018.8641686. URL: <https://ieeexplore.ieee.org/document/8641686/>.
- [86] Inês Martins et al. "Host-based IDS: A review and open issues of an anomaly detection system in IoT". In: *Future Generation Computer Systems* 133 (2022), pp. 95–113. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2022.03.001>. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X22000760>.
- [87] Pierre Matri et al. "Slog: Large-scale logging middleware for hpc and big data convergence". In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2018, pp. 1507–1512.
- [88] McKinsey. "The state of AI in 2023: Generative AI's breakout year". In: *McKinsey & Company* (2023). URL: <https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/the-state-of-ai-in-2023-generative-ais-breakout-year>.
- [89] E. Meneses, X. Ni, and et al. "Using Migratable Objects to Enhance Fault Tolerance Schemes in Supercomputers". In: *IEEE Transactions on Parallel and Distributed Systems* 26.7 (2015), pp. 2061–2074. DOI: 10.1109/TPDS.2014.2342228.
- [90] Dejan Milojicic et al. "Future of HPC: Diversifying Heterogeneity". In: *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2021, pp. 276–281. DOI: 10.23919/DATE51398.2021.9474063.
- [91] Gregor Molan et al. "Model for Quantitative Estimation of Functionality Influence on the Final Value of a Software Product". In: *IEEE Access* 11 (2023), pp. 115599–115616. DOI: 10.1109/ACCESS.2023.3325338.
- [92] Marrin Molan et al. "An Explainable Model for Fault Detection in HPC Systems". In: *High Performance Computing*. Ed. by Heike Jagode et al. Cham: Springer International Publishing, 2021, pp. 378–391. ISBN: 978-3-030-90539-2.

- [93] Martin Molan. *Pre-processing for Anomaly Detection on Linear Accelerator*. CERN openlab online summer intern project presentations. Sept. 2020.
- [94] Martin Molan et al. "Analysing Supercomputer Nodes Behaviour with the Latent Representation of Deep Learning Models". In: *Euro-Par 2022: Parallel Processing*. Ed. by José Cano and Phil Trinder. Cham: Springer International Publishing, 2022, pp. 171–185. ISBN: 978-3-031-12597-3.
- [95] Martin Molan et al. "GRAAFE: GRaph Anomaly Anticipation Framework for Exascale HPC systems". In: *Future Generation Computer Systems* 160 (2024), pp. 644–653. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2024.06.032>. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X24003327>.
- [96] Martin Molan et al. "Graph Neural Networks for Anomaly Anticipation in HPC Systems". In: *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. 2023, pp. 239–244.
- [97] Martin Molan et al. "Machine Learning Methodologies to Support HPC Systems Operations: Anomaly Detection". In: *European Conference on Parallel Processing*. Springer. 2022, pp. 294–298.
- [98] Martin Molan et al. "RUAD: Unsupervised anomaly detection in HPC systems". In: *Future Generation Computer Systems* 141 (2023), pp. 542–554. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2022.12.001>. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X2200406X>.
- [99] Martin Molan et al. "Semi-supervised anomaly detection on a Tier-0 HPC system". In: *Proceedings of the 19th ACM International Conference on Computing Frontiers*. 2022, pp. 203–204.
- [100] Martin Molan et al. "The Graph-Massivizer Approach Toward a European Sustainable Data Center Digital Twin". In: *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE. 2023, pp. 1459–1464.
- [101] Federico Monti, Davide Boscaini, and et al. "Geometric deep learning on graphs and manifolds using mixture model cnns". In: *Proc. of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5115–5124.
- [102] Adam Morrow, Elisabeth Baseman, and Sean Blanchard. "Ranking Anomalous High Performance Computing Sensor Data Using Unsupervised Clustering". In: *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2016, pp. 629–632. DOI: 10.1109/CSCI.2016.0124.
- [103] Giulia Moschini et al. *Anomaly and Fraud Detection in Credit Card Transactions Using the ARIMA Model*. 2020. arXiv: 2009.07578.

- [104] Nature. "Generative AI's environmental costs are soaring — and mostly secret". In: *Nature* (2024). URL: <https://www.nature.com/articles/d41586-024-00478-x>.
- [105] Alessio Netti, Zeynep Kiziltan, and et al. "FINJ: A fault injection tool for HPC systems". In: *European Conference on Parallel Processing*. Springer. 2018, pp. 800–812.
- [106] Alessio Netti et al. "A Conceptual Framework for HPC Operational Data Analytics". In: *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. 2021, pp. 596–603. DOI: 10.1109/Cluster48925.2021.00086.
- [107] Alessio Netti et al. "A Conceptual Framework for HPC Operational Data Analytics". In: *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. 2021, pp. 596–603. DOI: 10.1109/Cluster48925.2021.00086.
- [108] Alessio Netti et al. "A machine learning approach to online fault classification in HPC systems". In: *Future Generation Computer Systems* (2019).
- [109] Alessio Netti et al. "Online Fault Classification in HPC Systems through Machine Learning". In: *European Conference on Parallel Processing*. Springer. 2019, pp. 3–16.
- [110] Alessio Netti et al. "Operational data analytics in practice: experiences from design to deployment in production HPC environments". In: *Parallel Computing* 113 (2022), p. 102950.
- [111] OpenAI. "GPT-4 Technical Report". In: *arXiv preprint arXiv:2303.08774* (2023). URL: <https://arxiv.org/abs/2303.08774>.
- [112] Guansong Pang et al. "Deep Learning for Anomaly Detection: A Review". In: *ACM Comput. Surv.* (Mar. 2020). DOI: 10.1145/3439950.
- [113] Guansong Pang et al. "Deep Learning for Anomaly Detection: A Review". In: *ACM Comput. Surv.* 54.2 (Mar. 2021). ISSN: 0360-0300. DOI: 10.1145/3439950. URL: <https://doi.org/10.1145/3439950>.
- [114] Lynn A. Parnell et al. "Trends in High Performance Computing: Exascale Systems and Facilities Beyond the First Wave". In: *2019 18th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*. 2019, pp. 167–176. DOI: 10.1109/ITHERM.2019.8757229.
- [115] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [116] *Receiver operating characteristic*. Nov. 2021. URL: https://en.wikipedia.org/wiki/Receiver_operating_characteristic.
- [117] Robin Rombach et al. "High-Resolution Image Synthesis with Latent Diffusion Models". In: *arXiv preprint arXiv:2112.10752* (2021). URL: https://arxiv.org/abs/2112.10752?utm_source=chatgpt.com.

- [118] Luis Rosa et al. "Intrusion and anomaly detection for the next-generation of industrial automation and control systems". In: *Future Generation Computer Systems* 119 (2021), pp. 50–67. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2021.01.033>. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X21000431>.
- [119] Tara Salman et al. "Machine Learning for Anomaly Detection and Categorization in Multi-cloud Environments". In: *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)* (2017). DOI: 10.1109/CSCloud.2017.15. arXiv: 1812.05443.
- [120] Ketan Rajshekhkar Shahapure and Charles Nicholas. "Cluster Quality Analysis Using Silhouette Score". In: *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. 2020, pp. 747–748. DOI: 10.1109/DSAA49011.2020.00096.
- [121] Woong Shin et al. "Revealing Power, Energy and Thermal Dynamics of a 200PF Pre-Exascale Supercomputer". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1–14. ISBN: 9781450384421. DOI: 10.1145/3458817.3476188. URL: <https://doi.org/10.1145/3458817.3476188>.
- [122] Chunfeng Song et al. "Auto-encoder Based Data Clustering". In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Ed. by José Ruiz-Shulcloper and Gabriella Sanniti di Baja. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 117–124. ISBN: 978-3-642-41822-8.
- [123] Yujia Song et al. "Identifying performance anomalies in fluctuating cloud environments: a robust correlative-GNN-based explainable approach". In: *Future Generation Computer Systems* (2023).
- [124] Konstantin S Stefanov et al. "A Review of Supercomputer Performance Monitoring Systems". In: *Supercomputing Frontiers and Innovations* 8.3 (2021), pp. 62–81.
- [125] Erich Strohmaier et al. *TOP500 Statistics - Development over Time*. Tech. rep. Application Area. Top500.org, June 2024.
- [126] Erich Strohmaier et al. *TOP500 Statistics - Development over Time*. Tech. rep. Segments. Top500.org, June 2024.
- [127] Erich Strohmaier et al. *TOP500 Statistics - Performance Development*. Tech. rep. Top500.org, June 2024. URL: <https://www.top500.org/statistics/perfdevel/>.
- [128] Olivier Terzo and Jan Martinovič. *HPC, Big Data, and AI Convergence Towards Exascale: Challenge and Vision*. CRC Press, 2022.

- [129] The European Commission. *The EU Innovation Radar Platform. INNOVATION: Graph anomaly anticipation tool for exascale HPC systems*. [Online; accessed 31-December-2024]. The European Commission, 2024. URL: <https://innovation-radar.ec.europa.eu/innovation/58770>.
- [130] *TOP500 LIST - NOVEMBER 2023*. Nov. 2023. URL: <https://www.top500.org/lists/top500/list/2023/11/>.
- [131] *Top500List*. <https://www.top500.org/lists/top500/2020/06/>. 2020.
- [132] Hugo Touvron et al. "LLaMA: Open and Efficient Foundation Language Models". In: *arXiv preprint arXiv:2302.13971* (2023). URL: <https://arxiv.org/abs/2302.13971>.
- [133] *Trends in high performance computing: a historical overview and examination of future developments* | IEEE Journals and Magazine. URL: <https://ieeexplore.ieee.org/document/1598076/>.
- [134] Ozan Tuncer, Emre Ates, and et al. "Diagnosing performance variations in HPC applications using machine learning". In: *International Supercomputing Conference*. Springer, 2017, pp. 355–373.
- [135] Ozan Tuncer, Emre Ates, and Yijia et al. et Zhang. "Online Diagnosis of Performance Variation in HPC Systems Using Machine Learning". In: *IEEE Transactions on Parallel and Distributed Systems* (Sept. 2018).
- [136] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. "Singular value decomposition and principal component analysis". In: *A practical approach to microarray data analysis*. Springer, 2003, pp. 91–109.
- [137] Sean Wallace et al. "A Data Driven Scheduling Approach for Power Management on HPC Systems". In: (Nov. 2016). DOI: 10.1109/sc.2016.55. URL: <https://ieeexplore.ieee.org/document/7877134/>.
- [138] Haoyu Wang, Zetian Liu, and Haiying Shen. "Job scheduling for large-scale machine learning clusters". In: (Nov. 2020). DOI: 10.1145/3386367.3432588. URL: <https://dl.acm.org/doi/10.1145/3386367.3432588>.
- [139] Wei Wang et al. "Clustering with orthogonal autoencoder". In: *IEEE Access* 7 (2019), pp. 62421–62432.
- [140] Sholom M. Weiss et al. *Text Mining with Machine Learning: Principles and Techniques*. London, UK: Springer, 2010. ISBN: 978-1-84996-225-4.
- [141] Wikipedia. *CINECA* — *Wikipedia, The Free Encyclopedia*. <http://en.wikipedia.org/w/index.php?title=CINECA&oldid=954269846>. [Online; accessed 04-December-2021]. 2021.
- [142] Wikipedia. *Jira (software)* — *Wikipedia, The Free Encyclopedia*. [http://en.wikipedia.org/w/index.php?title=Jira%20\(software\)&oldid=1052315603](http://en.wikipedia.org/w/index.php?title=Jira%20(software)&oldid=1052315603). [Online; accessed 04-December-2021]. 2021.

- [143] Altair PBS Works. *PBS Professional®14.2 Plugins (Hooks) Guide*. <https://pbsworks.com/pdfs/PBSHooks14.2.pdf>. 2017.
- [144] Nan Wu and Yuan Xie. “A survey of machine learning for computer architecture and systems”. In: *ACM Computing Surveys (CSUR)* 55.3 (2022), pp. 1–39.
- [145] Peng Wu et al. “Unsupervised anomaly detection for underwater gliders using generative adversarial networks”. In: *Engineering Applications of Artificial Intelligence* 104 (2021), p. 104379. ISSN: 0952-1976. DOI: <https://doi.org/10.1016/j.engappai.2021.104379>. URL: <https://www.sciencedirect.com/science/article/pii/S095219762100227X>.
- [146] Rongbin Xu et al. “Improved Long Short-Term Memory based anomaly detection with concept drift adaptive method for supporting IoT services”. In: *Future Generation Computer Systems* 112 (2020), pp. 228–242. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2020.05.035>. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X20302235>.
- [147] Bin Yang et al. “End-to-end {I/O} Monitoring on a Leading Supercomputer”. In: *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 2019, pp. 379–394.
- [148] Renyu Yang et al. “Intelligent Resource Scheduling at Scale: A Machine Learning Perspective”. In: (Mar. 2018). DOI: 10.1109/sose.2018.00025. URL: <https://ieeexplore.ieee.org/document/8359158/>.
- [149] Xin Yang et al. “Machine Learning and Deep Learning Methods for Cybersecurity”. In: *IEEE Access* 6 (Jan. 2018), pp. 35365–35381. DOI: 10.1109/access.2018.2836950. URL: <https://ieeexplore.ieee.org/document/8359287/>.
- [150] Xuejun Yang et al. “The reliability wall for exascale supercomputing”. In: *IEEE Transactions on Computers* 61.6 (2012), pp. 767–779.
- [151] Felipe Vieira Zacarias et al. “Intelligent colocation of HPC workloads”. In: *Journal of Parallel and Distributed Computing* 151 (May 2021), pp. 125–137. DOI: 10.1016/j.jpdc.2021.02.010. URL: <https://arxiv.org/abs/2103.09019>.
- [152] Chuxu Zhang et al. “A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data”. In: *CoRR* abs/1811.08055 (2018). arXiv: 1811.08055.
- [153] Jie Zhou et al. *Graph Neural Networks: A Review of Methods and Applications*. 2018. DOI: 10.48550/ARXIV.1812.08434. URL: <https://arxiv.org/abs/1812.08434>.
- [154] Jie Zhou et al. “Graph neural networks: A review of methods and applications”. In: *AI open* 1 (2020), pp. 57–81.