



Machine learning for fault classification within the context of the energy graph-based visualisation technique

PJM Terreblanche

 **orcid.org/ 0009-0005-4016-3511**

Dissertation accepted in fulfilment of the requirements for the degree *Master of Engineering in Computer and Electronic Engineering* at the North-West University

Supervisor: Prof G van Schoor

Co-supervisor: Prof KR Uren

Graduation: July 2026

Acknowledgements

*“Trust in the Lord with all your heart and lean not on your own understanding;
in all your ways acknowledge him, and he will make your paths straight.”*

– Proverbs 3:5-6

I would like to express my gratitude to North-West University for allowing me the opportunity to enhance my skills and knowledge. Next, I would like to sincerely thank the following people, in no specific order, for the contribution they made to the completion of this dissertation:

- My supervisor, Prof. George van Schoor, for his leadership, wisdom, and guidance. Thank you for helping me to grow as a person. I admire your analytical and critical thinking approach to tackling problems.
- My co-supervisor, Prof. Kenny Uren, for his informative contributions, patience, and encouraging words throughout. Thank you for always being available for questions. Your passion and dedication to the field inspire me.
- My parents, sister and grandmother. Thank you for all your love, support, and encouragement throughout my years of study. I am immensely grateful for everything you have done for me over the course of my life.
- My friends who are doing their master’s, especially Ruart Victor, Andries Oelofse, Ruan van der Spoel, and Arnold van der Merwe, thank you for your friendship and support. You have made this journey worthwhile!
- A special word of thanks goes to Mr. Hendré Swart. Thank you for exposing me to the field of engineering during high school, motivating me, and teaching me the value of hard work.
- To all the unmentioned family and friends who have supported me in numerous ways over the years.

Abstract

Modern processing plants are complex and span multiple physical domains, posing challenges for fault detection and isolation (FDI). The advent of Industry 4.0 has facilitated the extensive collection of data, thereby accelerating the adoption of data-driven methods for fault detection and isolation. Techniques like energy graph-based visualisation (EGBV) address the challenges of multi-domain systems by using exergy and energy to characterise the system and analyse historical process data.

The EGBV technique is a recent FDI technique that aims to exploit the energy interactions between system components to describe the state of health of the system. In this study, the potential of machine learning (ML) to detect and isolate faults from the energy interactions within the EGBV context was investigated. Several objectives were identified to successfully apply and evaluate the performance of ML techniques in the context of the EGBV technique. A Simulink implementation of the Tennessee Eastman Process was used to generate process data to implement and evaluate the use of ML techniques for FDI purposes within the EGBV context.

Exploratory data analysis was used on the different types of EGBV data to assess their effectiveness in representing process data where system faults were present. This analysis also aided in explaining why some ML techniques had difficulty detecting and classifying certain faults when using specific EGBV data types.

Subsequent objectives involved the application of appropriate ML techniques to the different types of EGBV data, taking their abstractness and structures into account. Furthermore, the fault detection and classification performances of the ML techniques were compared with each other to identify if a certain data type allowed for better fault detection and classification.

Finally, the performance of these ML techniques was compared with benchmarks, including the original non-ML EGBV technique. The results showed that the non-ML EGBV technique surpassed both ML techniques and other benchmarks in fault detection, while ML exceeded in fault classification over the other mentioned methods. This study concluded that the use of energy and exergy data with the EGBV technique produced more

accurate results compared to the raw data used by other benchmark techniques.

Keywords: Machine learning; Fault detection and isolation; Tennessee Eastman Process; Energy graph-based visualisation.

Contents

1	Introduction	1
1.1	Background and motivation	1
1.2	Research question	3
1.3	Research objectives and methodology	3
1.3.1	Exploratory data analysis of data within the EGBV technique . . .	3
1.3.2	Application and evaluation of machine learning techniques using cost matrix data	4
1.3.3	Application and evaluation of machine learning techniques using node signature matrix data	4
1.3.4	Comparison of machine learning techniques against benchmark techniques	5
1.4	Dissertation outline	5
2	Literature survey and study	7
2.1	Introduction	7
2.2	Literature survey	8
2.2.1	Graph methods used in fault detection and isolation	8
2.2.2	Exergy and energy methods used in fault detection and isolation . .	9
2.2.3	Machine learning methods used in fault detection and isolation . . .	11
2.3	Process monitoring terminology	13

2.3.1	Fundamental element definitions	13
2.3.2	Function definitions	14
2.4	Fault detection and isolation	15
2.4.1	Overview	15
2.4.2	Fault types and behaviours	16
2.4.3	Types of FDI approaches	18
2.4.4	Comparison of FDI approaches	20
2.5	The EGBV technique	22
2.5.1	Overview of the EGBV technique	22
2.5.2	Operation of the EGBV technique	22
2.5.3	Historical use of the EGBV technique for FDI purposes	24
2.6	The Tennessee Eastman Process	26
2.6.1	Overview of the Tennessee Eastman Process	26
2.6.2	Types of faults in the Tennessee Eastman Process	28
2.6.3	Exergy and energy-based Tennessee Eastman model	30
2.7	Critical literature review	31
3	Exploratory data analysis	34
3.1	Introduction	34
3.2	Data format overview	35
3.3	Principal component analysis	37
3.3.1	Overview	37
3.3.2	Results and discussion	37
3.4	Distance matrices	41
3.4.1	Overview	41

3.4.2	Distance matrix examples	41
3.4.3	Results and discussion	44
3.5	Self-organising maps	49
3.5.1	Overview	49
3.5.2	Results and discussion	50
3.6	Distinctness of data representations	55
3.7	Conclusion	58
4	Machine learning for the cost matrix data	59
4.1	Introduction	59
4.2	The cost matrix structure	59
4.3	Machine learning techniques	60
4.3.1	Deep neural networks	60
4.3.2	Convolutional neural networks	64
4.4	Training, validation and testing datasets description	70
4.5	Architectures of machine learning models	70
4.6	Results	72
4.7	Discussion	74
4.8	Conclusion	77
5	Machine learning for the node signature matrix data	78
5.1	Introduction	78
5.2	Graph neural networks	78
5.2.1	GCN	81
5.2.2	ChebConv	81
5.2.3	GATConv	81

5.3	Training, validation and testing datasets description	82
5.4	Architectures of graphical machine learning models	82
5.5	Results	84
5.6	Discussion	86
5.7	Conclusion	89
6	Comparative study	90
6.1	Introduction	90
6.2	Traditional fault detection and classification	90
6.2.1	PCA	91
6.2.2	ICA	92
6.2.3	PLS	92
6.2.4	One-class SVM	93
6.2.5	SVM	93
6.2.6	ANN	94
6.3	EGBV SVD	94
6.3.1	Fault detection	94
6.3.2	Fault classification	95
6.4	Results	97
6.5	Discussion	98
6.6	Comparison with ML techniques	101
6.6.1	Results	101
6.6.2	Discussion	107
6.7	Conclusion	109
7	Conclusion	111

7.1	Summary and conclusions	111
7.1.1	Exploratory data analysis of cost matrix and node signature matrix data	111
7.1.2	Application and evaluation of machine learning techniques for the cost matrix data	112
7.1.3	Application and evaluation of machine learning techniques for the node signature matrix data	112
7.1.4	Comparison between machine learning techniques and benchmark techniques	112
7.2	Future work and recommendations	113
7.2.1	Feature selection for cost and node signature matrices	113
7.2.2	Alternative methods of fault signature generation with standard EGBV technique	113
7.2.3	Comparison with temporal machine learning techniques	114
7.2.4	Hybrid FDI techniques	114
7.2.5	Explainable AI	114
7.3	Closure	115
A	Tennessee Eastman Process variables	135
B	Exploratory data analysis	137
B.1	Principal component analysis results	137
B.2	Distance matrices results	137
B.3	Self-organising map results	137
C	Machine learning for the cost matrix data	139
D	Machine learning for the node signature matrix data	141

List of Figures

2.1	Steps of process monitoring (Adapted from [2]).	13
2.2	Typical process to implement an FDI system (Adapted from [2]).	15
2.3	Illustration of various hardware faults impacting a dynamic control system inside a processing plant (Adapted from [64]).	16
2.4	Broad classification scheme of various FDI techniques.	18
2.5	Simplified illustration of a coal fired power plant (left) and its attributed energy graph (right) (Adapted from [16]).	23
2.6	Process flowsheet of the TEP (Adapted from [98]).	27
2.7	Energy graph based visualisation of the TEP.	30
3.1	Distance matrix for random noise	42
3.2	Distance matrix for a step change.	42
3.3	Distance matrix for a drift.	43
3.4	Distance matrix for an oscillation.	43
3.5	Distance matrix for a decaying oscillation.	44
3.6	Distance matrix for normal operating conditions using cost matrix data. . .	45
3.7	Distance matrix for normal operating conditions using the node signature matrix data.	47
3.8	Structure of a self-organising map (Adapted from [135]).	49
3.9	Output of self-organising map for normal operating conditions, using cost matrix data.	51

3.10	Output of self-organising map for normal operating conditions, using the node signature matrix data.	54
4.1	Possible distributions of informative entries inside the cost matrix.	60
4.2	Architectures for a simple neural network (left) and a deep neural network (right) (Adapted from [154]).	61
4.3	Structure of a neuron (Adapted from [156]).	62
4.4	Examples of common activation functions (Adapted from [159]).	63
4.5	A neural network without Dropout (left) and a neural network with Dropout applied (right) (Adapted from [166]).	64
4.6	Structure of a CNN (Adapted from [170]).	65
4.7	Two-dimensional convolution operation performed by convolutional layer (Adapted from [174]).	66
4.8	One-dimensional convolution operation performed by convolutional layer (Adapted from [175]).	67
4.9	A convolutional kernel with a stride of 1 (top) and a convolutional kernel with a stride of 2 (bottom) (Adapted from [176]).	68
4.10	Max pooling (top) and average pooling (bottom) (Adapted from [148]). . .	69
5.1	Classification scheme for GNNs.	79
5.2	Typical structure of a graph CNN (Adapted from [197]).	80
6.1	Fault detection rates (%) for all techniques.	102
6.2	Fault classification rates (%) for all techniques.	105

List of Tables

2.1	Original faults that can be simulated in the TEP [98].	29
3.1	PCA results for cost matrix data.	38
3.2	PCA results for node signature matrix data.	40
3.3	Distance matrix results for cost matrix data.	46
3.4	Distance matrix results for node signature matrix data.	48
3.5	Self-organising map results for cost matrix data.	52
3.6	Self-organising map results for node signature matrix data.	55
3.7	Degree of distinctness for cost matrix data.	57
3.8	Degree of distinctness for node signature matrix data.	58
4.1	Symbol legend for different types of layers.	71
4.2	Optimised architectures of traditional non-graphical ML techniques for the cost matrix data.	72
4.3	Fault detection rates (%) for traditional non-graphical ML techniques using the cost matrix data.	73
4.4	Fault classification rates (%) for traditional non-graphical ML techniques using the cost matrix data.	74
5.1	Symbol legend for different types of graphical layers.	83
5.2	Optimised architectures of graphical ML techniques for the node signature matrix data.	84

5.3	Fault detection rates (%) for graphical ML models using the node signature matrix data.	85
5.4	Fault classification rates (%) for graphical ML models using the node signature matrix data.	86
6.1	Fault detection rates (%) for benchmark methods.	97
6.2	Fault classification rates (%) for benchmark methods.	98
6.3	Best fault detection rate (%) per fault.	103
6.4	Best fault classification rate (%) per fault.	106
A.1	Manipulated variables within the Tennessee Eastman process.	135
A.2	Measured variables within the Tennessee Eastman process.	136
C.1	Batch sizes used for the traditional non-graphical machine learning technique for the cost matrix data.	139
C.2	Learning rates used for the traditional non-graphical machine learning technique for the cost matrix data.	140
C.3	Datasets used for the traditional non-graphical machine learning technique for the cost matrix data.	140
C.4	Dropout values used for the layers of the DNN.	140
C.5	Dropout values used for the layers of the 1D CNN.	140
C.6	Dropout values used for the layers of the 2D CNN.	140
D.1	Batch sizes used for the graphical machine learning technique for the node signature matrix data.	141
D.2	Learning rates used for the graphical machine learning technique for the node signature matrix data.	141
D.3	Datasets used for the graphical machine learning technique for the node signature matrix data.	142
D.4	Dropout values used in the layers of the GCN-Net.	142
D.5	Dropout values used in the ChebConv-Net.	142

D.6 Dropout values used in the GATConv-Net.	142
---	-----

Table of abbreviations

AI	Artificial intelligence
ANN	Artificial neural network
CNN	Convolutional neural network
EDA	Exploratory data analysis
EGBV	Energy graph-based visualisation
EWMA	Exponential weighted moving average
DNN	Deep neural network
FDD	Fault detection and diagnosis
FDI	Fault detection and isolation
GNN	Graph neural network
HEOM	Heterogeneous Euclidean overlap metric
ICA	Independent component analysis
ML	Machine learning
NOC	Normal operating conditions
NS	Node signature
PCA	Principal component analysis
PLS	Partial least squares
RUL	Remaining usefull life
SGD	Stochastic gradient descent
SOM	Self-organizing map
SVD	Singular value decomposition
SVM	Support vector machine
TE	Tennessee Eastman
TEP	Tennessee Eastman Process
WSN	Wireless sensor networks

Chapter 1

Introduction

1.1 Background and motivation

Safety and reliability are critical factors influencing modern processing industries. In modern processing industries, process monitoring techniques can be used to ensure the safety and reliability of the processing plant [1]. Among the various process monitoring techniques, fault detection and diagnosis (FDD) plays a critical role in ensuring that the process can operate safely in the presence of faults [2]. In addition to ensuring safe operating processes, FDD also ensures that the process does not degrade when faults are present.

The absence of suitable FDD techniques poses significant threats to modern processing industries. Failing to react to system faults, results in a variety of adverse consequences, ranging from affecting product quality to posing threats to human operators and the surrounding environment [3]. Examples of severe industrial failures caused by inadequate FDD techniques include the Fukushima Dai-Ichi accident [4] and the Bhopal gas tragedy [5], where significant human life loss and disastrous environmental consequences occurred. In addition, untreated faults within the system also cause considerable economic losses. For example, the USA loses about \$ 20 billion annually [6], and China more than one million dollars daily due to untreated faults within processing industries [7].

Modern processing plants typically consist of several interconnected subsystems, which receive multiple inputs, followed by a typically non-linear process to transform the multiple inputs into multiple outputs. The complex nature of modern processing plants and their interconnected process subsystems make FDD difficult. Furthermore, the processing units inside the plant also consist of multiple multi-domain interconnected and interdependent components, which further complicate the matter [8]. Modern processing plants also have aspects that further complicate the FDD process, such as the mode of operation used

within the plant [9].

The complexity and scale of modern processing plants mentioned make it difficult for human operators to manually monitor the process for faults and therefore FDD approaches within modern processing plants have become more automated, requiring less intervention from human operators [10]. Despite this automation, some approaches still required input from human operators to operate successfully, such as their knowledge obtained through experience or a mathematical model of the system. However, these approaches have several drawbacks such as not being able to adapt to new previously unseen types of faults and the difficulty associated with creating an accurate first-order mathematical model of the system [9, 11]. Furthermore, the speed at which new equipment and processes are advancing makes previous knowledge used for FDD obsolete [12].

The advancement of Industry 4.0 has allowed modern processing industries to take advantage of the continuous flow of large amounts of data available within the processing plant [13]. Collecting this continuous flow of data has allowed modern processing industries to acquire a large amount of historical process data, giving rise to data-driven FDD approaches. Data-driven FDD approaches use machine learning (ML) or statistical techniques to extract patterns from historical process data for FDD, and do not require knowledge or model-based information [9].

Modern processing systems within the petrochemical industry, can be viewed as multi-domain systems, meaning that the different subsystems within the processing system belong to different physical domains and energy is converted between these domains [14]. The different domains may include domains such as electrical, mechanical, chemical, and thermodynamic. Attempting to model such a multi-domain system, where non-linear interactions exist between domains, is complex and challenging. Energy is a useful quantity that can be used to characterise and monitor such multi-domain systems [15].

The energy graph-based visualisation (EGBV) technique is a novel exergy and energy-based fault detection and classification technique that can be used for complex multi-domain systems [16]. Using the energy interactions between system components, the EGBV technique can describe the state of health of the system. The EGBV technique is a hybrid fault detection and classification technique that combines a qualitative exergy and energy model of the system with historical exergy and energy information. Furthermore, the EGBV technique offers different levels of abstraction when monitoring a system. These different levels of abstraction correspond to two matrix-like structures, called the node signature matrix and the cost matrix, with the cost matrix representing the highest level of abstraction.

Fault detection and isolation (FDI) within the EGBV context have been conducted mainly using statistical techniques, such as singular value decomposition (SVD), at the highest level of abstraction (cost matrix). There exists a need to explore alternative techniques

such as ML within the EGBV context to determine if there exist alternative patterns within the various levels of abstraction that can be used for the purpose of fault detection and classification.

The Tennessee Eastman Process (TEP) is a popular benchmark process that has been widely used in the field of FDD [17]. Irreversible exothermic gas-liquid reactions take place within the TEP, to simulate a complex realistic chemical process. Furthermore, the TEP is also commonly used to test new FDD techniques and to compare them against existing techniques.

1.2 Research question

This study attempts to answer the following question: *How does the use of ML techniques within the EGBV context for FDI purposes compare against other benchmark FDI techniques?* In order to answer this question a comparative study will be undertaken between ML techniques, applied within the EGBV context, and other benchmark FDI techniques, including the standard EGBV technique without any ML. Furthermore, the TEP will be used as a benchmark process to compare the mentioned techniques. The performance of the FDI techniques will be quantified and compared in terms of their fault detection and fault classification rates.

1.3 Research objectives and methodology

1.3.1 Exploratory data analysis of data within the EGBV technique

As mentioned in Section 1.1, the results of the SVD of the cost matrix have been commonly used for fault detection and classification within the EGBV context. The raw elements within the cost and node signature matrices have not been widely investigated for fault detection and classification purposes. Therefore, it is necessary to explore how the raw elements within the cost and node signature matrices will behave under changed conditions, when different types of faults are present within the system.

Exploratory data analysis (EDA) techniques will be used to investigate the raw elements inside the cost and node signature matrices. Furthermore, the EDA techniques may provide insight into the dynamics of the data inside the cost and node signature matrices under different conditions, for example, when there is no fault present within the system or a fault had occurred.

Additionally, EDA techniques such as dimensionality reduction will make it possible to

visualise highly dimensional data, thereby seeing how the data changes when a fault is introduced into the system. Clustering techniques will also be used to visualise if there is a difference between “faulty” and “normal” data. If there is a difference between “faulty” and “normal” data, it will result in distinct clusters being identified by the clustering technique.

1.3.2 Application and evaluation of machine learning techniques using cost matrix data

The SVD results of the cost matrix have been mainly used for fault detection and classification. However, the raw elements inside the cost matrix have not yet been used for fault detection and classification purposes. The raw data inside the cost matrix may contain patterns within, which, if extracted, could potentially allow for more accurate fault detection and classification compared to the SVD approach used in the standard EGBV FDI technique. Therefore, ML techniques will be implemented with the cost matrix data, to extract potential patterns within the data for the purpose of fault detection and classification.

The cost matrix does not contain node attributes or information on the connectivity of the nodes inside a graph. Instead, the cost matrix represents the relative Euclidean distances between the nodes of two graphs. Since the cost matrix does not represent a graph, non-graphical ML techniques will be investigated and applied to the cost matrix data. The fault detection and classification results of the non-graphical ML techniques will be compared amongst each other as well as the graphical ML techniques used for the node signature matrix data.

1.3.3 Application and evaluation of machine learning techniques using node signature matrix data

Unlike the cost matrix, the node signature matrix has not been extensively used for fault detection and classification purposes. The node signature matrix could potentially offer more accurate fault detection and classification compared to the cost matrix. Consequently, ML techniques will be implemented with the data of the node signature matrix for the purpose of fault detection and classification.

The node signature matrix represents a graph and contains information such as the node attributes and the connectivity between the different nodes. Therefore, graphical ML techniques will be investigated and applied to the node signature matrix data. The fault detection and classification results of the graphical ML techniques will be compared amongst each other and to the non-graphical ML techniques used for the cost matrix data.

1.3.4 Comparison of machine learning techniques against benchmark techniques

The fault detection and classification results obtained with the ML techniques applied to the cost and node signature matrices need to be compared to benchmark FDI techniques to determine whether the use of ML within the EGBV context offers benefits in terms of fault detection and classification. Several benchmark FDI techniques, including the standard EGBV technique, will be investigated and applied. Furthermore, the fault detection and classification results of the benchmark techniques will be compared with the ML techniques to reach conclusions.

1.4 Dissertation outline

Chapter 2 provides a brief literature survey and a literature study. The literature survey focusses on data-driven FDI techniques where graphs, exergy, energy, or ML have been used. The literature study provides the necessary terminology and concepts relevant to the topic of this current study.

The results of the EDA on the data for both the cost and node signature matrices for certain faults are provided in **Chapter 3**. Before the results are presented, brief descriptions of the various EDA techniques are provided. Furthermore, the results of the EDA for the different faults are presented and summarised.

Chapter 4 details how ML techniques are applied to the cost matrix data. Initially, a brief overview is given on how the selected ML techniques function, followed by the fault detection and fault classification results of the ML techniques. Finally, the fault detection and fault classification results are briefly interpreted and discussed.

Chapter 5 focusses on how graphical ML techniques are applied to the node signature matrix data. Firstly, a brief overview of graphical ML techniques is provided, and the selected graphical ML techniques are briefly described. This is followed by the fault detection and fault classification results of the graphical ML techniques and a brief interpretation and discussion.

Different benchmark FDI techniques are briefly described and implemented in **Chapter 6**. The results of fault detection and classification of the benchmark FDI techniques are presented and interpreted. In addition, the fault detection and classification results of the benchmark techniques are compared with the results obtained in Chapters 4 and 5 and discussed.

Chapter 7 concludes this study and summarises the results obtained within each objec-

tive. In addition, future work and recommendations are provided.

Chapter 2

Literature survey and study

2.1 Introduction

In this chapter a broad overview of the field of fault and process monitoring will be provided, as well as highlighting the concepts and terminology relevant to this study. Initially, the various terminology used within the fault and process monitoring field will be described and clarified. Thereafter, a brief overview of FDI, which is a classification task, will be given, followed by the generic steps required to implement an FDI scheme as well as the difficulties experienced when implementing an FDI scheme.

The different types of faults and types of FDI approaches are discussed and in addition, the different types of ML-based FDI approaches are discussed in more detail. Following the discussion on the different types of FDI approaches, these approaches are compared to identify advantages and disadvantages.

The EGBV technique will be discussed in detail, which includes a brief overview of the EGBV technique, followed by the procedure when applying the EGBV technique. The mathematical structures and formulas used within the EGBV technique are given, as well as a brief discussion on how the EGBV technique data have been used for FDI purposes.

Lastly, the Tennessee Easman Process (TEP) will be discussed, which includes the layout of the TEP and the chemical process that takes place inside the TEP. In addition, the various types of faults that can be present inside of the TEP is discussed, followed by a brief discussion on the exergy and energy-based Tennessee Eastman (TE) model that was used for the current study.

2.2 Literature survey

This subsection presents a brief literature survey on data-driven FDI techniques where graphs, exergy, energy, or ML have been used. The purpose of this brief literature survey is to investigate the different approaches used in other studies and how it differs from the EGBV technique, regarding how graphs, and exergy and energy are used. Furthermore, the brief literature survey also highlights popular ML methods that are used in data-driven FDI.

2.2.1 Graph methods used in fault detection and isolation

Khorasgani *et al.* used a graph convolutional neural network (CNN) for FDI within a single domain system [18]. Their single domain case study system consisted of 100 simulated water tanks that were connected to each other. They represented their water tank network as a weighted undirected graph, where each node represented a water tank, and the edge weights were proportional to the inverse of the distance between two water tanks. The authors were able to detect whether a node had a fault or not by using binary classification and fault isolation to identify which node had a fault.

A new method for accessing global graph context was proposed by Lyu *et al.* [19]. Allowing nodes to use additional information regarding the global graph context instead of only information from immediate neighbours resulted in better fault detection when using graph neural networks (GNNs). The authors primarily focused on industrial control systems, specifically the communication system within the system. Furthermore, the authors used the industrial field network of the system to construct the graph, where each node represented a device within the network. Temporal compression was used to convert the time series data of each node/device into its features for a given timestamp.

Wu *et al.* proposed a complex-value graph CNN that could work with spatiotemporal data that contained complex values [20]. The authors focused on electric grid networks and used it as undirected weighted graphs. The nodes inside the graph represented buses, and the edges represented the transmission lines connecting the buses. The ability of the graph CNN to work with complex values allowed the authors to use the voltage phasor data of each bus as node attributes within the graph. Lastly, the ability of the graph CNN to process temporal data makes it possible to predict the future state of buses using past data.

He *et al.* used a graph to model the spatial and temporal relationships of data generated by different modules inside of an unmanned aerial vehicle (UAV) for fault detection purposes [21]. A GNN was used to extract spatial features within the data and a temporal neural network to extract temporal features from the data. The authors then used the GNN and

temporal neural network to predict the different variables of the UAV. The predictions were compared against measurements and the residuals were used for fault detection. A threshold method was used to determine if the residuals are above or below a threshold, thereby indicating the presence of a fault.

A graph CNN was used by Li *et al.* to process aircraft sensor data to diagnose faults [22]. First, they converted the raw sensor data into grayscale images, called sensor data images, to capture the temporal and cross-sensor relationships within the data. This was followed by clustering techniques applied to the sensor data images to construct a graph representation of the data, which was then used with the graph CNN for fault diagnosis.

Chen *et al.* proposed an interaction-aware technique that could be used to learn possible graph representations of a sensor network, which in turn can be used with a graph CNN for fault diagnosis [23]. The interaction-aware technique searched for different types of relationships between sensor signals such as correlations, similarity, and functional. Different types of relationship would result in different graph representations for the sensor network, which were combined and used with the graph CNN for fault diagnosis. Furthermore, the authors found that representing the sensor network by using different graph representations resulted in superior fault diagnosis performance compared to other benchmark techniques. This was attributed to the different graph representations that were capable of representing different aspects of the sensor network.

2.2.2 Exergy and energy methods used in fault detection and isolation

Choi *et al.* suggested that by using past energy consumption patterns it would be possible to predict future energy consumption [24]. The authors argued that the energy consumption data of a machine contains valuable information regarding the state of the machine. A machine tool spindle was used as a case study, and a random forest was used to predict its future energy consumption. The predictions made by the random forest were compared to the actual measurements and the residuals were used for fault detection. If a residual was above or below a certain threshold, it would be indicative of an anomaly/fault.

The thermoeconomic method and an artificial neural network (ANN) were used by Orozco *et al.* to determine malfunctioning components within an externally fired gas turbine [25]. According to the authors malfunctions would either cause a decrease in the output power or would result in an increase in the consumed fuel to produce a constant output power. Thermoeconomic diagnosis compares the current state of a system against a reference state, thereby obtaining deviations in the parameters. The authors compared the thermoeconomic results of the ANN against the thermoeconomic result of the real measurements and based on the residuals it is was possible to identify malfunctions in the equipment.

Wang *et al.* used advanced exergy analysis to detect and classify malfunctions within a

thermal power plant [26]. The authors defined an internal exergy parameter used to detect a malfunction and defined it as a ratio by using a reference state when the component was still functioning normally. After detecting the malfunction, the authors used advanced exergy analysis to classify the malfunction, focusing on the destruction of exergy that occurred within each component and divided the destroyed exergy into endogenous and exogenous parts.

Exergy models and a probabilistic neural network were used by Cao *et al.* to diagnose faults within a marine gas turbine [27]. Two exergy models were used to model the loss of exergy within the marine gas turbine. One exergy model represented the actual system, while the other exergy model was an estimation model for the system. The difference between the loss in exergy of the two models was determined and used as input for the probabilistic neural network.

Wang *et al.* used exergy data and a combination of ML techniques to diagnose faults within a pneumatic cylinder [28]. The authors calculated the exergy of the compressed air system and used it as input to a combination of ML techniques. The first ML technique was a stacked auto-encoder used to extract features from the exergy data. Extracted features from the exergy data were then fed into a support vector machine (SVM) or an ANN and the results were compared. The authors found that the combination of the stacked auto-encoder and the SVM had the best performance. Lastly, the authors also argued that exergy is a good type of data since it can fuse information from other variables such as pressures and flows.

Santolamazza *et al.* used energy consumption residuals and an ANN to detect faults within a compressed air generation system [29]. The ANN was used to simulate a “healthy” system that was compared to the actual energy consumption of the system to generate residuals. A control chart was then used to determine whether the residuals represented an anomaly/fault present within the system.

Picallo-Perez *et al.* used thermoeconomic diagnosis to identify malfunctioning components within a heating and domestic heated water system [30]. Their thermoeconomic diagnosis defined a parameter per component called the malfunction cost, which represented the additional fuel and resources required within the total system due to the malfunction impacting the component. The component with the largest malfunction cost would be the component that caused the most additional resource consumption and would be labelled as the largest faulty component within the system. In addition, the authors also used dysfunction analysis to investigate exergy consumption within components due to malfunctions present in other components.

Xu *et al.* combined traditional thermoeconomic diagnosis with the operational characteristic curves of components to identify faulty components within a marine diesel engine [31]. The operational characteristic curves attempted to identify the independent variables,

known as operational characteristic variables, that govern the exergy loss associated with each component. In addition, the authors also created reference operational characteristic curves to represent the system under normal conditions. The difference between the current operational characteristic curves and the reference operational characteristic curves and traditional thermoeconomic analysis were used to identify faulty components.

2.2.3 Machine learning methods used in fault detection and isolation

An *et al.* used a combination of a long short-term network (LSTM) and ANNs to classify bearing faults [32]. The authors specifically tested their approach on bearings within a time-varying system, where the speed in the system varied due to changing loads or required power. Inspired by the infinitesimal method, the authors devised an approach to segment the time-varying input data before being used to train their proposed model. Furthermore, the LSTM made it possible to learn temporal relationships between the segments that represent the time-varying data. Finally, the authors tested their approach against other benchmark techniques used in related work, such as CNNs and ANNs, to show the superior performance of their approach.

An adaptive neuro-fuzzy inference system (ANFIS) was used by Tightiz *et al.* to diagnose faults within a power transformer [33]. The dissolved gas analysis method (DGAM) was used to monitor the power transformer, and the resulting ratios used as input to train the ANFIS. Two practical datasets, each representing different power transformers, were used to evaluate the performance of the ANFIS. Furthermore, the authors compared the performance of the ANFIS with other benchmark ML techniques such as SVMs and ANNs on the same two datasets. The authors found that the ANFIS had slightly higher classification accuracies compared to the benchmark techniques on the two datasets.

Chae *et al.* used a wide variety of ML techniques, a SVM, a CNN, and a LSTM, to diagnose flow accelerated corrosion within the pipes of a nuclear power plant [34]. Furthermore, they used vibration data as input for the mentioned ML techniques. All three techniques could accurately diagnose scenarios that resulted in distinct changes in the vibrational data. However, subtle and ambiguous changes within the vibrational data, due to incipient scenarios, could only be accurately diagnosed with the LSTM, since the LSTM was able to model the gradual temporal change within the vibrational data.

A hybrid ML model consisting of a LSTM and an ANN was proposed by Cai *et al.* to monitor the condition of tools [35]. Raw sensor data of the tool was fed into the LSTM to extract temporal features. The extracted temporal features were then combined with the process information and the combined vector was given as input to the ANN to predict tool wear. Furthermore, the authors compared the performance of their hybrid ML model with other benchmark techniques such as an ANN, a CNN, and an SVM. Two datasets,

the NASA Ames milling dataset and the 2010 PHM Data Challenge dataset, were used to evaluate and compare the hybrid ML model with the benchmark techniques. The ability of the LSTM to extract temporal features from the sensor data allowed it to more accurately predict the condition of the tools compared to the benchmark techniques.

Ince *et al.* utilised a one-dimensional CNN and motor current data to detect bearing faults within a motor [36]. The motor current data was first filtered to remove the fundamental frequency of the motor and downsampled before being provided as input to the one-dimensional CNN. The fault detection performance of the one-dimensional CNN was compared with other benchmark techniques, including an SVM and an ANN. It was found that the one-dimensional CNN had slightly lower performance metrics compared to the benchmark techniques; however its performance was still comparable.

Wen *et al.* used a CNN with image representations of sensor data to diagnose bearing faults within a motor [37]. Most ML techniques cannot work with raw sensor data and require the data to be converted to a useful form where features can be extracted. The authors used a conversion process where time-domain signals were converted to grayscale images that were used to train the CNN. Furthermore, the performance of the CNN was compared with other benchmark techniques, such as a SVM and an ANN. The CNN greatly outperformed the SVM and ANN; however, it only slightly outperformed the other benchmark techniques and was more comparable.

A deep belief network (DBN) was proposed by Che *et al.* to diagnose faults within rolling bearings [38]. The authors used a combination of raw vibration signals and their corresponding time and frequency domain indicators as input for the DBN. Furthermore, they made the DBN domain adaptable by allowing it to diagnose faults under different modes of operation. The performance of the domain-adaptable DBN was evaluated against benchmark techniques such as a SVM and an ANN under different modes of operation. It was found that under different operating conditions (domains), the domain-adaptable DBN significantly outperformed the other benchmark techniques.

Audibert *et al.* did a comparative study by comparing conventional statistical fault detection techniques, such as principal component analysis (PCA) and independent component analysis (ICA), against ML techniques using five public benchmark datasets [39]. The comparison did not reveal significant differences between the performance of the conventional statistical techniques and the ML techniques. However, the authors found that by increasing the amount of data allowed the ML techniques to achieve better results. Lastly, the authors argued that conventional statistical techniques should not be ignored, thereby considering them to be a useful benchmark against which other techniques can be compared.

2.3 Process monitoring terminology

Process monitoring is accomplished through a cycle consisting of several steps/functions, which include fault detection, fault identification, fault diagnosis, and process recovery [40]. A graphical representation of the steps inside of process monitoring is shown in Figure 2.1 [2]. However, since process monitoring is a multi-disciplinary field, there exists no standard terminology to describe the mentioned functions, as well as the fundamental elements of each function [41]. Therefore, the terminology needs to be clarified to allow the reader to understand the concepts. Therefore, the terminology regarding the functions mentioned and their fundamental elements will be discussed in the following subsections.

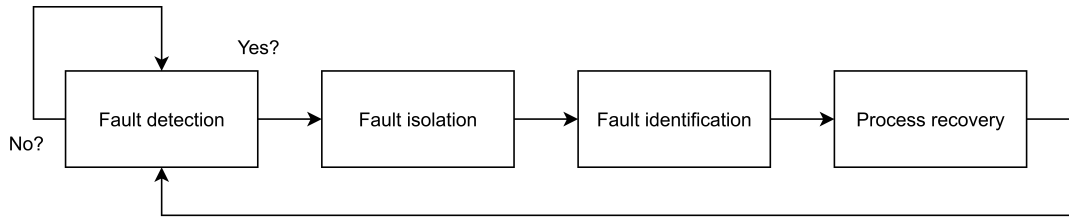


Figure 2.1: Steps of process monitoring (Adapted from [2]).

2.3.1 Fundamental element definitions

A fault can be defined as an event where at least one of the system's variables or properties experiences an undesirable change [42]. An alternative definition for a fault is provided by Aldrich *et al.*, referring to a fault as any abnormal behaviour causing a system or process to change in an undesirable way different from its normal operating point [43]. Furthermore, according to Isermann, a fault can be defined as a state that can lead to a malfunction or failure [44].

A malfunction refers to an intermittent irregularity that prevents a system from performing a desired function [45]. According to Venkatasubramanian *et al.*, malfunctions are caused by changes in the system, such as structural changes or erroneous sensors and actuators [46]. Lastly, as the complexity of the system grows, the chances of faults and therefore malfunctions also tend to increase [47].

Unlike a malfunction, a failure is defined as a permanent interruption in the ability of a system to fulfil a certain function under a specific operating condition [45]. Furthermore, a failure can also be viewed as a severe fault, which has evolved and led to the end of the system being able to function, thereby causing safety hazards to the environment and human operators [47].

2.3.2 Function definitions

Various functions exist within the process monitoring domain regarding faults, which include fault detection, fault isolation, fault identification, fault diagnosis, and fault prognosis [40, 42, 48]. The mentioned process monitoring functions are listed in terms of their complexity and order of application.

Fault detection refers to the ability to discern whether a fault is present within the system and to provide the time when the fault was detected [42]. Fault detection forms the first part of fault monitoring and is usually followed by fault isolation, fault identification, or fault diagnosis, as mentioned in several studies [2, 40, 42, 49].

Fault isolation refers to the ability to determine the location of the fault and the type of fault that occurred [50]. It should be noted that the “isolation” part of “fault isolation” does not refer to the isolation of specific components in the system with the aim of stopping the fault [51]. Instead, it refers to the ability to identify the faults and the components that they affect [50, 52].

Fault identification is similar to fault isolation but additionally determines the magnitude of the fault and the time-variant behaviour of the fault [42, 50]. In some texts, fault identification and fault isolation have been used interchangeably, without a definite distinction between them [51].

Fault diagnosis refers to the ability to determine the type of fault, as well as the location, size, and time of detection associated with the fault [42]. Furthermore, fault diagnosis can also refer to the ability to perform a root cause analysis of the fault and identify the variables that caused the fault [51]. Ultimately, fault diagnosis can be seen as a combination of the functions mentioned above.

Fault prognosis is focused on predicting when possible future faults may occur with the aim of applying predictive maintenance and scheduled repairs [48]. Furthermore, fault prognosis aims to predict the future condition of the system using metrics such as the remaining useful life (RUL) of the system [53]. However, due to the uncertainty present in the data and other sources, fault prognosis models may not be accurate and therefore are not widely implemented in the industry [54].

2.4 Fault detection and isolation

2.4.1 Overview

An FDI system refers to a system that is capable of both detecting and isolating a fault [41]. Data-driven FDI approaches, including ML techniques are discussed in Section 2.4.3, considering the FDI process as a classification task [55]. Consequently, the classification task performed by FDI ML approaches requires it to be able to detect if a fault is present and then to correctly classify the detected fault.

Several processes have to be completed in order to implement an FDI system. The first step in implementing an FDI system is the identification of a process or system that should be monitored for faults. Therefore the data of this identified system has to be collected and analysed. The collected data mentioned can later be used to derive process models of the system [56] or to learn the complex non-linear relationships present within the system [57]. After collecting the system processing data, there is an optional step of extracting and selecting features from the data. These extracted features of the data may allow more valuable information on the current state of the process, compared to using only the original data [58]. Moreover, only certain features can be selected from the extracted features, to ensure that the FDI system has a better ability to differentiate between different types of faults [59]. Following the optional feature extraction and selection steps, the next step is to select an FDI technique and use the collected data or features to adapt it to the system or process. After the FDI technique is adapted to the system or process, its efficacy can be validated and tested. An illustration of the various steps required to implement an FDI system is shown below in Figure 2.2 [2].

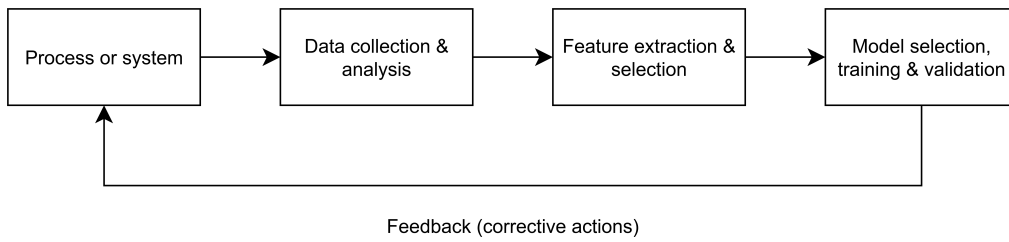


Figure 2.2: Typical process to implement an FDI system (Adapted from [2]).

Effective FDI systems should have desirable properties such as being able to be sensitive and robust to all types of faults [40]. Furthermore, according to Venkatasubramanian *et al.* other properties of effective FDI systems include fast detection, being able to uniquely identify faults, being adaptable, and offering an explanation facility for the system's decisions [46]. More advanced properties may include the ability to identify multiple faults simultaneously, as well as the computational and storage requirements of FDI systems

that must operate in real time.

Implementing fault monitoring systems in real-world scenarios requires the system to be carefully designed keeping the operating environmental conditions of the processing plant in mind [60]. Additionally, real-world fault monitoring systems require the sensors of the processing plant to be optimally placed, since their locations within the processing plant determine the degree to which the fault monitoring system can detect and identify faults [61]. Furthermore, real-world sensors pose several difficulties when used for fault monitoring systems, since the sensors may have different sampling frequencies and measure heterogeneous data at different scales [62].

2.4.2 Fault types and behaviours

Various factors can cause faults, and these factors can be classified as internal or external factors [41]. External factors refer to influences from the environment and can include phenomena such as dust, humidity, and high temperature. Conversely, internal factors refer to phenomena directly related to the actual system, such as overheating, missing lubrication, and leaks.

Faults can be classified into different categories according to the origin of the fault. In the context of a modern processing plant, faults can be classified as hardware, software, or network and communication faults [63]. In addition, hardware faults can be subdivided into changes in the process parameters, structural changes, and the malfunction of sensors and actuators [46]. An illustration of the various types of hardware faults that can influence a dynamic control system inside a modern processing plant is shown in Figure 2.3 [64].

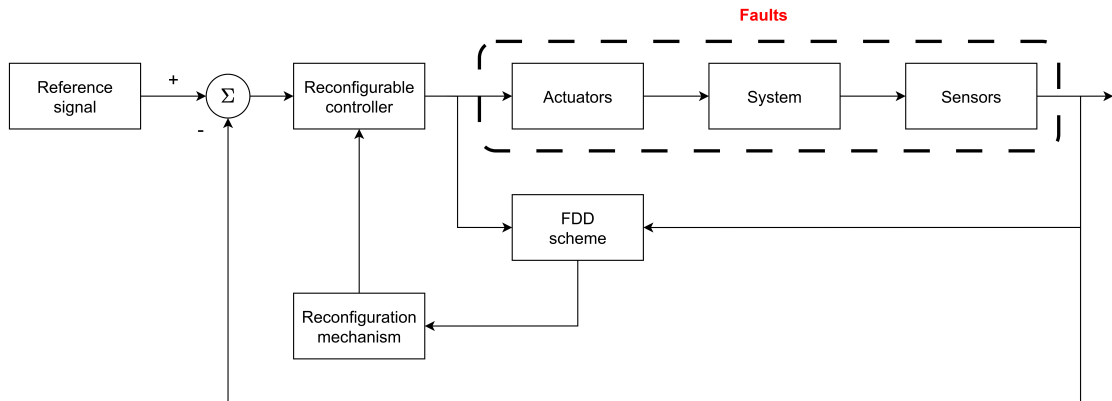


Figure 2.3: Illustration of various hardware faults impacting a dynamic control system inside a processing plant (Adapted from [64]).

Due to the complexity and scale of modern processing plants, it is difficult to measure

all these parameters. Consequently, there may be a change in some of the unmeasured parameters over time. This change in unmeasured process parameters may go unnoticed for some time, until it affects the rest of the system, which can lead to a hardware fault.

Structural changes refer to physical changes in the process itself. Examples of structural changes include broken equipment such as a controller failure or a leaking pipe. Furthermore, structural changes alter the way information flows between the different variables that describe the processing plant [46].

Each material inside the structure contains an inherent number of defects. The number of defects within a material is unknown. A large number of these defects inside the material can cause the material to become damaged, which later turns into structural faults [60].

The malfunction of sensors and actuators refers to sensors and actuators that are not properly functioning or not functioning as anticipated. Sensor and actuator malfunctions are typically due to a lack of maintenance. Therefore, this lack of maintenance may cause sensors to develop biases, give incorrect measurements and limit the range of operation of actuators.

Wireless sensor networks (WSN) have been widely applied in the monitoring of modern petrochemical processes [65]. These sensor networks consist of “smart” sensors that can connect to each other to form networks and contain batteries onboard, enabling them to process data and connect to the network. However, these “smart” sensors face additional faults compared to traditional sensors, and these faults can be broadly classified as soft or hard faults [66]. Soft faults collectively refer to faults present when the “smart” sensors process data and include faults such as offsets and data loss. On the other hand, hard faults refer to faults present within the actual “smart” sensor and include faults such as the sensor not being calibrated or a battery failure.

Faults can also be divided into additive or multiplicative faults, according to how process variables are changed [64]. Additive faults add to the process variables, such as an offset appearing on a sensor [67]. Conversely, multiplicative faults refer to faults that cause the process to change by the product of process variables, such as changes within system parameters [68].

The different faults can be further classified according to their time behaviour, including categories such as transient, intermittent, incipient, and permanent faults [63]. These various types of faults differ in terms of their duration and the extent of their damage to the system and are briefly discussed below.

- Transient faults refer to faults caused by the transient behaviour of the system; however, their lifetime is relatively short, and they disappear quickly [63]. Transient faults are usually random in nature and do not cause permanent damage to the

system [69].

- Intermittent faults are cyclic in nature and generally occur between state transitions [70]. After intermittent faults have disappeared, the system can usually function correctly without the need for external correction [71].
- Incipient faults are slow and gradually develop within the system [72]. Due to their slow and gradual development, incipient faults are typically masked by the normal variations inside the system parameters, causing them to only be detectable once their effects become severe and cause irreversible damage to the system. Lastly, closed-loop systems make the detection of incipient faults difficult, since the faults are masked with control actions [72].
- Permanent faults refer to faults due to broken equipment and will disappear only once the faulty equipment is repaired or replaced [73]. Therefore, the system will only be able to operate normally once the broken component has been repaired or replaced.

2.4.3 Types of FDI approaches

According to Venkatasubramanian and coworkers, FDI techniques can be broadly classified as model-based or process history-based [46, 74, 75]. In addition, model-based FDI techniques are further subdivided into qualitative and quantitative model-based techniques. Process history-based FDI techniques, also commonly referred to as data-driven techniques, are subdivided into statistical and non-statistical approaches. Figure 2.4 illustrates the broad classification scheme of FDI techniques suggested by Venkatasubramanian *et al.*

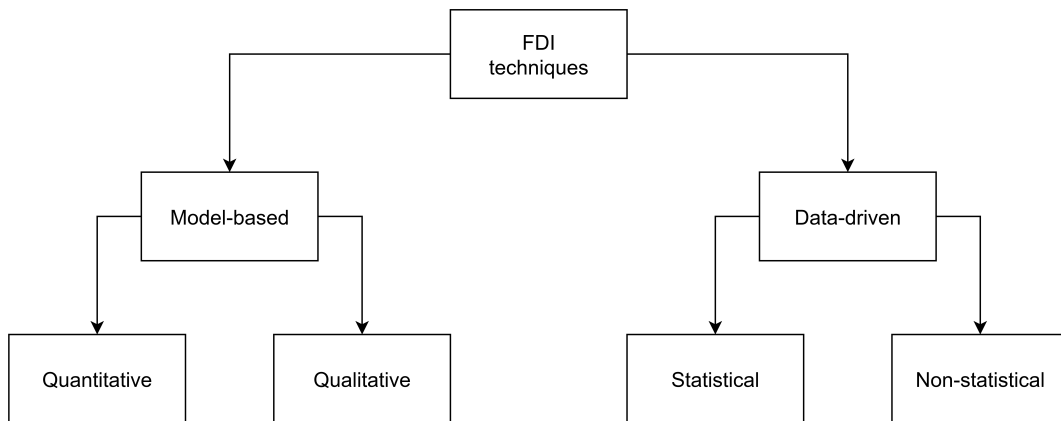


Figure 2.4: Broad classification scheme of various FDI techniques.

Other classification schemes for the various FDI techniques exist, such as the classification proposed by Dai and Gao [56], that argued that all FDI techniques can be seen

as data-driven techniques, since each technique uses some form of data. Moreover, using data-driven techniques as the umbrella term, Dai and Gao divided FDI techniques into model-based, signal-based, and knowledge-based techniques. Another classification scheme proposed by Li *et al.* classified FDI techniques as being either data-driven, deep knowledge-based or analytical-model-based techniques [76]. However, the classification scheme (Figure 2.4) proposed by Venkatasubramanian *et al.* will be used for the current dissertation.

Qualitative model-based FDI techniques refer to knowledge-based techniques where prior information about the process is used to examine the system for faults [74]. Prior information usually originates from an experienced human operator or observations of the plant, and the system will use this prior information with a set of rules to determine whether a fault has occurred. One major disadvantage of qualitative models is that they can generate physically unrealisable solutions, also known as spurious solutions, due to the qualitative reasoning followed by the models [74].

Quantitative model-based FDI techniques refer to approaches where a first-principle mathematical model of the process is used to determine if there are faults in the system [46]. Additionally, this mathematical model must be able to correctly represent the dependencies between the signals generated by the process, actuators, and sensors [44]. The performance of the mathematical model is continuously compared with the performance of the real-world processing plant, and any discrepancies between the two are used to determine if a fault has occurred. One disadvantage of a quantitative model-based approach is that the accuracy of the model severely decreases when there are unmodeled disturbances and uncertainties present [63]. Furthermore, quantitative model-based FDI techniques can usually only be applied to very specific non-linear and linear processes [46].

Unlike model-based approaches, data-driven approaches do not require prior information about the process, be it quantitative or qualitative information. Instead, it only requires large amounts of historical data [75]. However, the success of a data-driven approach depends on the quality of the data and how the data was obtained [77, 78]. Due to Industry 4.0 where large amounts of data are generated by modern processing plants, there has been a growing interest in data-driven approaches.

Data-driven approaches can be divided into statistical and non-statistical approaches [63]. Statistical approaches can be divided into univariate and multivariate statistical approaches [75]. Univariate statistical approaches are unable to exploit the correlation between variables, while multivariate statistical approaches are able to work with a large number of highly correlated variables.

Statistical approaches are suitable for fast fault detection; however, they fall short when fault classification is required. Most statistical approaches assume that there are linear relationships between variables, which may not be the case for a processing plant with

complex non-linear behaviour [79]. Furthermore, some statistical approaches assume that the process data belong to a certain statistical distribution, however, automated complex plants have dynamic behaviour, and their data may not conform to the assumed statistical distribution [80].

Non-statistical approaches include techniques such as pattern recognition and ML [8]. In contrast to statistical approaches, non-statistical approaches can learn the non-linear behaviour of the system [79]. However, these non-statistical approaches require considerably more data than the statistical approaches to learn the non-linear behaviour present within the system. In addition to requiring more data than statistical approaches, some of the non-statistical approaches, such as supervised ML techniques, may require labelled data for training [81].

ML techniques can be classified according to the type of data they receive, leading to supervised, semi-supervised, and unsupervised ML techniques [82]. With supervised ML techniques, all the data is labelled, while limited labelled data is available for semi-supervised ML techniques, and no labelled data is available for unsupervised ML techniques. Labelled data refers to data with meaning, i.e., it is data that is identifiable and can be classified.

Semi-supervised and unsupervised ML techniques are the most applicable for real-world processing plants. The limited or usually lack of labelled data makes it difficult to apply supervised ML techniques in real-world processing plant scenarios. The huge amounts of data generated by complex modern processing plants in addition to the rarity of severe faults that occur, are the reasons for the lack of labelled data.

2.4.4 Comparison of FDI approaches

Model-based approaches might be easier to interpret since they allow one to model the cause-effect relationships within the system [83]. These cause-effect relationships make it possible to see why the model produced the certain output, when given a certain input. On the contrary, some data-driven approaches are labelled as black-box models, which means that it is not easy to interpret why they got a certain output for a corresponding input [81].

In order to derive a quantitative model of a process or system, in-depth knowledge is required about that process or system, which includes physical and mathematical knowledge [1]. Furthermore, it is difficult to derive accurate quantitative or mathematical models for complex processing plants [84]. Conversely, with data-driven techniques, no in-depth knowledge of the system or process is required and only historical data of the system or process is needed. Data-driven techniques have been successfully applied to systems that exhibit behaviours such as being dynamic, non-linear, non-Gaussian, time-varying, and non-stationary [2]. All these behaviours would have been difficult to model with a

quantitative mathematical model.

Provisions must be made for when there are uncertainties and noise present within the process data when either quantitative model-based techniques or data-driven techniques are used [78, 85]. Uncertainties and measurement noise must be considered when designing the quantitative model, since the model is not a perfect reflection of the process, and failure to account for these uncertainties and noise will result in an inaccurate model. Noisy data may cause data-driven techniques to learn the noise present within the data, resulting in a phenomenon called over-fitting [86]. Instead of extracting useful relationships within the data, over-fitting will cause the data-driven technique to learn the noise within the data. On the contrary, qualitative model-based techniques, such as expert systems which are developed with the behaviour and knowledge of human experts, can deal with uncertainty and noise present in the data [87].

Quantitative model-based techniques, such as dynamic process models, make it possible to detect faults under different operating conditions [83]. Data-driven techniques, on the other hand, will only be able to detect faults under certain operating conditions, based on the operating conditions that their training data represent [55]. However, this limitation of data-driven techniques can be overcome by using training data, which represent the system under different operating conditions instead of a single operating condition.

Conventional statistical approaches for FDD are not as effective when dealing with situations where the process data exhibit behaviours such as being non-linear, non-Gaussian, time-varying, and non-stationary [88]. ML techniques, on the other hand, are capable of dealing with highly dimensional data, which exhibit some of the behaviours mentioned above [89].

It is clear from the above discussion that no single approach, i.e. quantitative, qualitative model-based or data-driven approaches, can meet all the requirements for an efficient fault diagnostic system. Instead, as suggested by Venkatasubramanian *et al.* a hybrid approach, combining aspects of model-based and data-driven approaches, could be beneficial to ensure that most of the requirements for an efficient fault diagnostic system are met [75]. Consequently, a hybrid approach called the energy graph-based visualisation (EGBV) technique, which will be used in this dissertation, will be discussed in detail in the next section.

2.5 The EGBV technique

2.5.1 Overview of the EGBV technique

The EGBV technique, developed by van Schoor and Uren, is a hybrid FDI technique that entails a combination of qualitative model-based and data-driven based techniques [90]. The qualitative model aspect of the EGBV technique focusses on representing the energy interactions between the different processing units in the processing plant using an attributed directed graph. In addition, the directed graph allows the energy information from the processing plant to be represented in a structured manner. Lastly, the data-driven aspect of the EGBV technique requires historical energy data from the processing plant to detect and classify faults within the system.

2.5.2 Operation of the EGBV technique

The EGBV FDI technique determines the energy information of the system, by converting the raw measured variables of the system, and using thermodynamic laws and energy balance equations, to represent the energy flow between the different system components, as well as the change in exergy across the different components [16]. Exergy is defined as the amount of useful work a system or component can do, and unlike energy, which is conserved, exergy can be destroyed [91]. Furthermore, there are different types of exergies, such as physical and chemical exergy [92].

Once the energy and exergy information of the system has been derived, it can be represented as a directed graph, as shown in Figure 2.5, which signifies the attributed energy graph of a coal fired power plant [16]. In addition, the various nodes shown in Figure 2.5 represent the various processing units within the system, while the arrows indicate how energy flows between the different processing units. Note that in Figure 2.5, the change in total exergy, i.e. both physical ($\Delta\dot{B}_{pn}$) and chemical exergy ($\Delta\dot{B}_{cn}$), across node n is represented by $\Delta\dot{B}_n$, and the energy flow from node i to node j is represented by $\dot{E}_{ij}$.

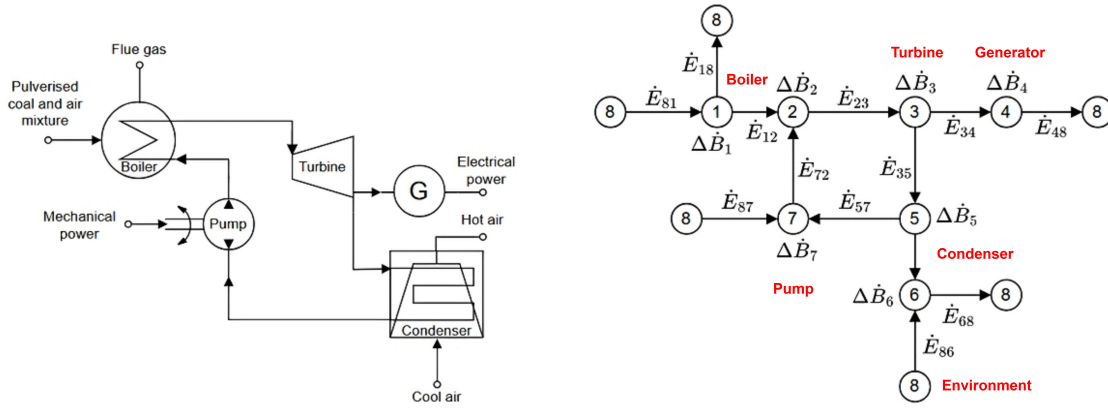


Figure 2.5: Simplified illustration of a coal fired power plant (left) and its attributed energy graph (right) (Adapted from [16]).

After deriving the attributed energy graph of the system, it can be more compactly written as an augmented matrix, known as the node signature matrix. An example of a node signature matrix can be seen in (2.1). Note that in (2.1) the changes in exergies across the various nodes are written to the left of the vertical line. The change in physical exergy across node n is indicated as $\Delta \dot{B}_{pn}$ while the change in chemical exergy across node n is indicated as $\Delta \dot{B}_{cn}$. The directed energy flows between the various nodes are written to the right of the vertical line. The order of the subscripts in the energy flow terms is important since they indicate the direction of the energy flow, i.e. the term $\dot{E}_{mn}$ denotes the flow of energy from node m to node n . Lastly, each row vector inside the node signature matrix contains the exergy and energy information of a specific processing unit inside the system; more specifically, each row vector indicates the change in physical and chemical exergy across the node, as well as indicating the flow of energy in and out of the node.

$$N_s = \left[\begin{array}{cc|cccc} \Delta \dot{B}_{p1} & \Delta \dot{B}_{c1} & \dot{E}_{11} & \cdots & \cdots & \dot{E}_{1N} \\ \Delta \dot{B}_{p2} & \Delta \dot{B}_{c2} & \dot{E}_{21} & \cdots & \cdots & \dot{E}_{2N} \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ \Delta \dot{B}_{pN} & \Delta \dot{B}_{cN} & \dot{E}_{N1} & \cdots & \cdots & \dot{E}_{NN} \end{array} \right] \quad (2.1)$$

After constructing the node signature matrix (N_s), the EGBV technique can be used to detect and identify faults by comparing the constructed node signature matrix against a baseline node signature matrix. The baseline node signature matrix mentioned refers to the derived node signature matrix of the system when it is “healthy”, meaning that the system is operating under normal conditions. Comparing two node signature matrices results in an additional square matrix, called the cost matrix. The cost matrix represents the distances between the entries inside the two node signature matrices and is obtained

by applying the Heterogeneous Euclidean Overlap Metric (HEOM) function to the two node signature matrices [90].

The distance determined by the HEOM function between node i of a “healthy” node signature matrix $(N_{s,hi})$ and node j from the current operating node signature matrix $(N_{s,oj})$ is given by:

$$\text{HEOM}(N_{s,hi}, N_{s,oj}) = \sqrt{\sum_{a=1}^k \delta(N_{s,hi}(a), N_{s,oj}(a))^2}, \quad (2.2)$$

where $N_{s,hi}(a)$ refers to the (i, a) entry inside the $N_{s,hi}$ node signature matrix, and $N_{s,oj}(a)$ refers to the (j, a) entry inside the $N_{s,oj}$ node signature matrix. Furthermore, a refers to the column number, while k denotes the total number of columns in both node signature matrices. The function δ , used in (2.2), is applied to the pair of numeric values $(N_{s,hi}(a), N_{s,oj}(a))$ and is given by:

$$\delta(N_{s,hi}(a), N_{s,oj}(a)) = \frac{|N_{s,hi}(a) - N_{s,oj}(a)|}{\text{range}_a}. \quad (2.3)$$

It is evident from (2.3) that the δ function determines the absolute difference between the two numbers and divides the result by a normalisation factor (range_a). Note that range_a denotes the difference between the maximum and minimum value inside of column a of the “healthy” node signature matrix $(N_{s,h})$.

The cost matrix can be constructed by applying the HEOM function between each pair of nodes inside the two node signature matrices. The structure of the cost matrix is shown in (2.4). Note that entry C_{ij} inside of the cost matrix represents the result of applying the HEOM function between node i of a “healthy” node signature matrix $(N_{s,h})$ and node j from the current operating node signature matrix $(N_{s,o})$.

$$C = \begin{bmatrix} C_{11} & \cdots & \cdots & C_{1N} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ C_{N1} & \cdots & \cdots & C_{NN} \end{bmatrix} \quad (2.4)$$

2.5.3 Historical use of the EGBV technique for FDI purposes

Data from both the node signature matrices and cost matrices have been used for FDI purposes [93]. Furthermore, based on several studies, it appears that the cost matrix data are a more popular choice to use for FDI purposes compared to the node signature matrix

data [90, 94, 95, 96, 97]. A brief discussion of how the data of the cost and node signature matrices are used differently for FDI purposes is provided below.

Various properties of the cost matrix data can be used for FDI purposes, which include its eigenvalues and eigenvectors, as well as a descriptive statistic called the distance parameter [90], [94]. The eigenvalues and eigenvectors of the cost matrix are typically used to create qualitative fault signatures that can be used to identify and classify faults [95]. Alternatively, the eigenvalues and eigenvectors can be used in a quantitative manner, by having a database of node signature matrices corresponding to the different states of the system. Thereby, cost matrices can be created by comparing these database node signature matrices against the current operating node signature matrix, using the HEOM function [96]. The cost matrix with the smallest eigenvalues and eigenvectors, that correlates with the node signature matrix entry inside the database with the closest resemblance to the current operating node signature matrix, can be used to classify the current state of the system.

The distance parameter mentioned simplifies the cost matrix into a single number. Similar to the quantitative eigendecomposition approach, the distance parameter approach requires a database of node signature matrices corresponding to the different states of the system. Various cost matrices are then created by comparing the current operating node signature matrix against the node signature matrices inside the database, using the HEOM function. The different cost matrices that were generated in the above-mentioned step can all then be simplified using the distance parameter. The smallest distance parameter value corresponding to the database entry of the node signature matrix that closest resembles the current operating node signature matrix can be used to classify the current state of the system [96]. Furthermore, there are different techniques for calculating the distance parameter, such as the Hungarian method [94] or by determining the average of the diagonal entries within the cost matrix [96].

A residual-based approach for FDI purposes using the node signature matrix data was suggested by Nesar *et al.* [93]. The residual-based approach required a database of reference node signature matrices, corresponding to the different states of the system. The difference was determined between the current operating node signature matrix and the reference node signature matrices inside the database. Determining the above-mentioned difference required that all entries within the current operating node signature matrix to be subtracted from the corresponding entries inside the reference node signature matrix and dividing the results with the corresponding entries inside the reference node signature matrix. This operation resulted in a matrix, called the residual graph matrix, which represented the differences between the corresponding entries inside the two node signature matrices. Furthermore, the residual graph matrix can be normalised by dividing each entry with the absolute maximum value of the corresponding column. Thereby, using the normalised residual matrices and thresholds, based on the standard deviation, it was

possible to detect and classify faults.

2.6 The Tennessee Eastman Process

In this subsection a brief overview of the TEP and its operating principles are provided. Furthermore, the different types of faults that can be simulated within the TEP are also briefly summarised. Finally, the Simulink implementation of the TEP used in this study, with additional exergy and energy variables, is presented and briefly described.

2.6.1 Overview of the Tennessee Eastman Process

The TEP is a model that simulates an industrial chemical process and was developed to allow researchers to design and evaluate process control strategies and applications [98]. Fault diagnostics was one of the potential applications for which the TEP model was developed. Furthermore, the ability of the TEP to model a realistic chemical industrial process, including the non-linear relationships between the different process units, made it an attractive benchmark for researchers to test and compare different process control strategies.

Inside the TEP there are eight components, which include four reactants, two products, as well as an inert and a by-product [99]. Note that the four reactants are indicated as A, C, D, and E, the two products as G, and H, the inert product as B, and the byproduct as F, for subsequent discussions and figures. Additionally, the chemical reactions that occur within the TEP are irreversible gas-liquid exothermic reactions [98]. Furthermore, the processing units within the TEP consist of a reactor, condenser, separator, compressor, and stripper unit [40]. The various processing units within the TEP are shown in Figure 2.6 [98].

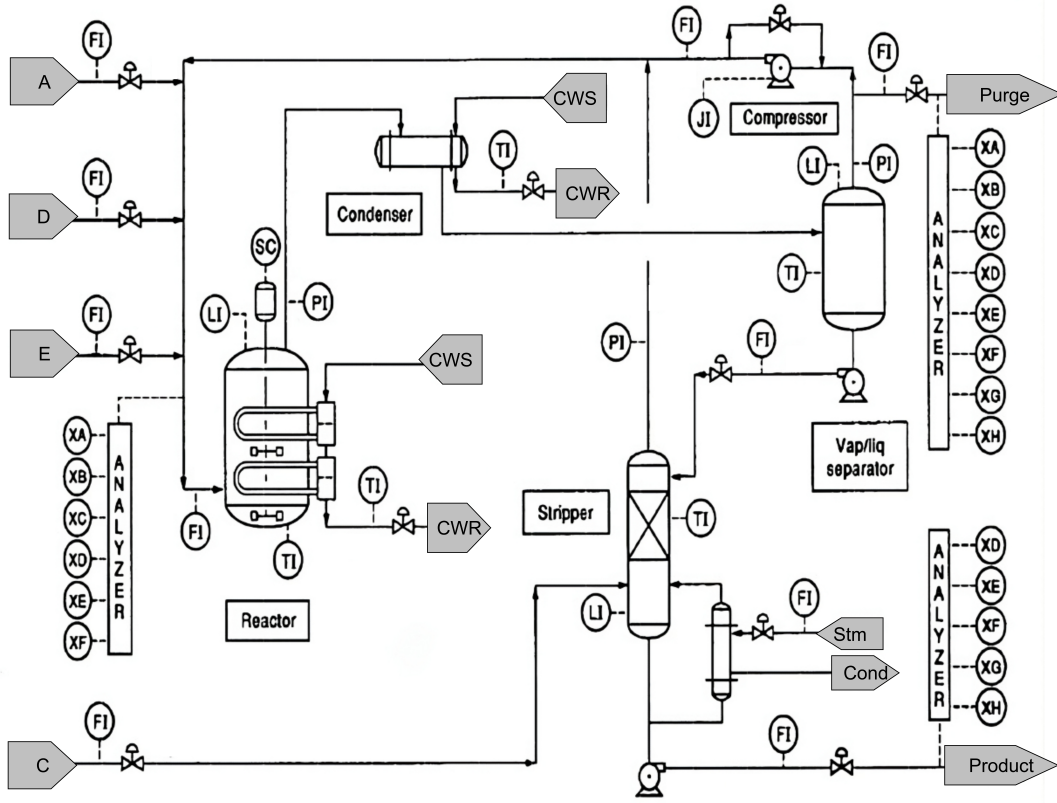


Figure 2.6: Process flowsheet of the TEP (Adapted from [98]).

Within the TEP, the four reactants are initially fed into the reactor with additional catalysts, where they react with each other to form a vapour product [100]. The vapour product is then fed into a condenser, where cooling is used to convert most of the vapour into a liquid product [101]. After the condenser, the vapour-liquid mixture enters a vapour-liquid separator, where the vapour-liquid mixture is separated into a gas and a liquid. The gaseous components are recycled and fed back to the reactor through the compressor, and a portion of the gaseous components is purged to ensure that there is no build-up of inert or by-products within the system [40]. While the liquid components are fed to the stripper, any remaining reactants are removed, and thus the stripper ensures that the two products produced by the TEP have an acceptable purity [100].

The TEP has several process operational modes, with each mode depicting a different ratio between the two products that are produced and the production rate at which the two products are produced [98]. In addition to having several operational modes, the TEP requires a control structure, since it is open-loop unstable and can cause sudden shutdowns if not properly controlled [102]. Over the years, various control strategies for the TEP have been suggested [40, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111], allowing the stable control of the TEP. Unfortunately, the feedback nature of closed-loop control strategies may hinder the early detection of faults since the control actions may mask the presence

of faults until they become too large and severe [44].

Further, the TEP contains both measurable and manipulated variables [40]. Measurable variables refer to properties of the system which can be measured such as pressures, levels, and temperatures. Manipulated variables, on the other hand, refer to parts of the system that can be controlled, such as the position of valves, the speed of agitators, and the flowrate of the reactants. Inside the TEP, there are 41 measurable variables and 12 manipulated variables [102] and these variables are summarised in Appendix A. Note that the last manipulated variable (agitator speed, see Appendix A), remains constant during the operation of the TEP and is usually ignored, resulting in a total of 52 variables (41 measurable variables and 11 manipulated variables).

2.6.2 Types of faults in the Tennessee Eastman Process

Originally, 20 faults could be simulated within the TEP and are summarised below in Table 2.1 [98]. The different types of faults influenced the properties of the different processing components within the TEP, such as their temperatures, pressures, and feed rates. Furthermore, some of the faults have unknown origins, meaning that they cannot be linked to certain hardware failures, as shown in Table 2.1. Readers are referred to the original article of Downs and Vogel [98], for additional information regarding the mentioned "Unknown" faults. Lastly, each type of fault shown in Table 2.1, had a specific dynamic, such as occurring instantly or gradually over time. Note that A, C and D represent three of the four reactants and B represents the inert product.

Table 2.1: Original faults that can be simulated in the TEP [98].

Fault number	Description	Behaviour
1	A/C feed ratio, B composition constant	Step
2	B composition, A/C ratio constant	Step
3	D feed temperature	Step
4	Reactor cooling water inlet	Step
5	Condenser cooling water inlet temperature	Step
6	A feed loss	Step
7	C header pressure loss- reduced availability	Step
8	A, B, C feed composition	Random variation
9	D feed temperature	Random variation
10	C feed temperature	Random variation
11	Reactor cooling water inlet temperature	Random variation
12	Condenser cooling water inlet temperature	Random variation
13	Reaction kinetics	Slow drift
14	Reactor cooling water valve	Sticking
15	Condenser cooling water valve	Sticking
16	Unknown	Unknown
17	Unknown	Unknown
18	Unknown	Unknown
19	Unknown	Unknown
20	Unknown	Unknown

The various faults mentioned in Table 2.1, were reported to have varying degrees of difficulty in being detected and classified [112]. Several studies have reported that, specifically faults 3, 9 and 15 were notoriously complex faults to detect and classify [113, 114, 115]. Additionally, other studies have shown that it is difficult to distinguish between different faults when there is a significant overlap between most of the data from the various faults [116, 117].

Over the years, additional faults have been added to the list of faults shown in Table 2.1. One example is the additional valve fault introduced by Chiang *et al.* [40] and to date, 28 faults can be induced within the TEP [118]. However, the 28 faults mentioned can only be induced using a revised model of the TEP [119]. Furthermore, this revised model includes additional variables compared to the original model, which contained only 53 variables (41 measurable variables and 12 manipulated variables). The exergy and energy-based TE model that will be used in this study is only capable of inducing the original 20 faults mentioned in Table 2.1 and will be described in more detail in the next section.

2.6.3 Exergy and energy-based Tennessee Eastman model

In this current study, a Simulink implementation of the TEP with additional exergy and energy variables to characterise it, developed by Vosloo and based on previous research by her and colleagues [120], will be used to generate data. Furthermore, the Simulink model can generate the exergy and energy information of the plant, as well as the original 53 variables mentioned in Section 2.6.1. Note that the model used in this current study is only capable of inducing the original 20 faults mentioned in Section 2.6.2.

An energy graph-based depiction of the TEP is shown in Figure 2.7. This energy graph representation of the TEP is used within the Simulink model to calculate the change in exergy across the different processing units as well as how energy flows between these different processing units. Note that the total change in exergy across node n is indicated as $\Delta \dot{B}_n$, and the directed energy flow from node i to node j is indicated as $\dot{E}_{ij}$.

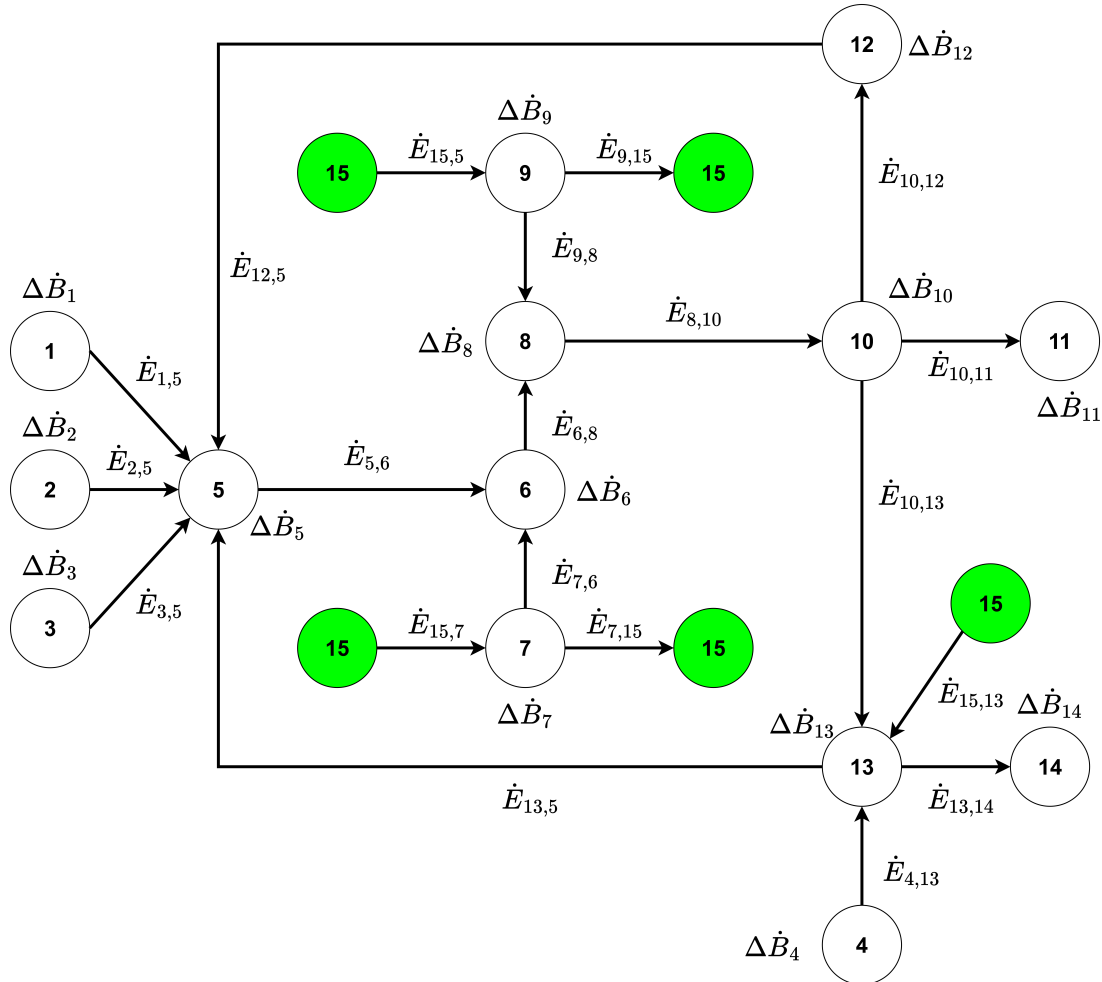


Figure 2.7: Energy graph based visualisation of the TEP.

The various nodes shown in Figure 2.7, correspond to the various processing units within

the TEP, as well as the surrounding environment, and are listed below. Note that A, C, D, and E represent the four reactants, and B represents the inert product.

1. A feed stream to the mixing point
2. D feed stream to the mixing point
3. E feed stream to the mixing point
4. A/B/C feed stream to the steam stripper
5. Mixing point where the three feed streams and two recycle streams mix
6. Reactor
7. Reactor internal heat transfer section
8. Condenser
9. Condenser internal heat transfer section
10. Vapour liquid separator
11. Purge stream leaving the system
12. Recycle compressor
13. Steam product stripper
14. Product stream leaving the system
15. Environment node

The dimensions of the node signature matrices, which contain the exergy and energy information of the TEP, will be a 15×17 augmented matrix. Note that the number of rows in the augmented matrix corresponds to the number of nodes shown in Figure 2.7. The first two columns inside the augmented matrix represent the changes in physical and chemical exergy across the different nodes, while the remaining columns represent the directed energy flows between a node and the rest of the nodes. The dimensions of the cost matrix, after applying the HEOM function to a pair of node signature matrices, will be a square 15×15 matrix.

2.7 Critical literature review

In the process monitoring field, specifically the fault monitoring field, there exists a wide range of different terms that describe different quantities and tasks. As stated in the

beginning of the chapter, there exists no standard agreed upon terminology for the terms in the fault monitoring field. Consequently, the author felt the need to clarify the various terms within the fault monitoring field to allow the reader to correctly interpret the findings of this current study and the discussion on the findings of other researchers.

Real-world practical FDI systems are complex and require careful planning and design. Various factors must be considered, such as the environment surrounding the process and the placement of sensors within the process. Although this study focusses on a simulated plant, the steps required to create an FDI system remain the same as those for a practical real-world FDI system. The steps required to create an FDI system are an iterative process where the data of the system are collected and analysed, and then used within a model, to test the performance of the model. The model in this context refers to a data-driven FDI model that includes statistical and ML-based techniques. The process of selecting a suitable data-driven model may be repeated numerous times until the performance of the selected model meets the necessary requirements.

Various types of faults have been shown to exist within a process. In addition, these various types of faults can be classified according to different criteria such as their dynamics, how they influence process parameters, and what aspects of the physical process they influence such as hardware or software faults. Due to the various types of faults, there are varying degrees of success to detect and classify these different types of faults.

Different classes of FDI approaches exist within the literature, i.e. quantitative and qualitative model-based techniques and data-driven techniques, and these FDI approaches all have their respective advantages and disadvantages. It was argued that a hybrid FDI approach would work best by combining the advantages of the different types of FDI approaches.

The EGBV technique is a hybrid FDI technique that combines aspects of qualitative model-based and data-driven FDI techniques and uses exergy and energy to characterise a process. Since exergy and energy are universal quantities, the EGBV technique can be applied to multidomain processes, i.e. processes that span across different physical domains such as mechanical, electrical, and chemical. In addition, the EGBV technique uses two different data structures, i.e., node signature and cost matrices, to monitor the process.

Within the EGBV technique, only the cost matrix data have been predominantly used for FDI purposes. Further, mostly statistical FDI approaches, i.e. the eigenvalues and eigenvectors obtained after applying SVD, have been applied to the cost matrix data. Consequently, there exists a need to investigate the potential for applying other data-driven FDI approaches, such as ML, to the different data structures within the EGBV technique. Therefore, this current study will investigate the potential of applying ML approaches to the EGBV technique for FDI purposes.

The TEP will be selected as a benchmark for this study, since it models a complex non-linear chemical plant and has been widely used as a benchmark for other FDI studies. Furthermore, the TEP contains different types of faults, which have varying degrees of difficulty in being detected and classified, and thus this research will determine how effective it would be to combine ML with the EGBV technique for FDI purposes. More relevant literature regarding this study will be discussed in the relevant chapters to follow.

Chapter 3

Exploratory data analysis

3.1 Introduction

This chapter presents the results of the exploratory data analysis (EDA) on the cost and node signature matrix data, obtained from the Tennessee Eastman Process (TEP) model. First, an overview of the different data formats of the cost and node signature matrix data produced by the TEP model will be discussed. In the overview, the relationship between the cost and node signature matrices will be briefly described. Furthermore, the differences between the cost and node signature matrix data formats will be highlighted, as well as what they represent, respectively. By exploring how the cost and node signature matrices are related and differ, can provide insight why EDA results may differ between the matrices. The different types of EDA techniques that were applied in this study are briefly described, and finally, the results of the different EDA techniques will be summarised and interpreted.

EDA plays a pivotal role as the initial step in any task which requires the processing of data, including ML tasks. EDA allows for the discovery of patterns within the data, thereby allowing one to construct possible future hypotheses [121]. Furthermore, EDA is useful for generating hypotheses based on the data such as which faults would be difficult to detect and classify. However, the hypotheses generated with the EDA will only be confirmed once the data-driven technique has been applied to the data and the results were analysed [122]. Consequently, the results of the EDA in this chapter may provide evidence for future chapters to explain the phenomena why certain ML models might perform better than others and why ML models might struggle to identify and classify certain types of faults.

Visual EDA techniques were selected to investigate if there are differences between the data representations offered by the cost and node signature matrices, respectively. These

differences in data representation can allow one form of representation to express data in a more distinct way compared to the other. Furthermore, some of the visual EDA techniques were used to determine how the presence of certain faults influenced the dynamics of the system when viewed through representations of either the cost matrix or node signature matrix. For the visual EDA techniques all entries/variables inside the cost and node signature matrices were used, and no variables/feature selection was performed.

3.2 Data format overview

As mentioned in Chapter 2, the TEP model used in this study can calculate and record the exergy and energy data inside the plant. The calculated exergy and energy data can be rearranged into a directed graph as shown in Figure 2.7. The directed graph can be compactly written as an augmented matrix, known as the node signature matrix. An example of the format of a node signature matrix is shown in (3.1).

$$N_s = \left[\begin{array}{cc|cccc} \Delta\dot{B}_{p1} & \Delta\dot{B}_{c1} & \dot{E}_{11} & \cdots & \cdots & \dot{E}_{1N} \\ \Delta\dot{B}_{p2} & \Delta\dot{B}_{c2} & \dot{E}_{21} & \cdots & \cdots & \dot{E}_{2N} \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ \Delta\dot{B}_{pN} & \Delta\dot{B}_{cN} & \dot{E}_{N1} & \cdots & \cdots & \dot{E}_{NN} \end{array} \right] \quad (3.1)$$

In (3.1) the entries to the left of the dividing line represent the changes in physical and chemical exergy across the different nodes, whereas the entries to the right of the dividing line represent the directed energy flows between the different nodes. More specifically, $\Delta\dot{B}_{pn}$ refers to the change in physical exergy across node n , while $\Delta\dot{B}_{cn}$ refers to the change in chemical exergy across node n , and $\dot{E}_{ij}$ refers to the directed energy flow from node i to node j .

Conversely, the cost matrix offers a different level of abstraction compared to the node signature matrix. In essence, the cost matrix represents the differences between two node signature matrices by comparing the distances between the entries of the different nodes inside the matrices. As mentioned in Chapter 2, the difference is usually determined between a reference node signature matrix and the current node signature matrix, which represents the plant while it is under operation. The reference node signature matrix represents the system while under normal operating conditions (NOC), meaning that the system was “healthy” with no faults present.

The distance determined by the HEOM function between node i of a “healthy” node signature matrix ($N_{s,h}$) and node j from the current operating node signature matrix ($N_{s,o}$) is given by:

$$\text{HEOM}(N_{s,hi}, N_{s,oj}) = \sqrt{\sum_{a=1}^k \delta(N_{s,hi}(a), N_{s,oj}(a))^2}, \quad (3.2)$$

where $N_{s,hi}(a)$ refers to the (i, a) entry in the $N_{s,h}$ node signature matrix, and $N_{s,oj}(a)$, refers to the (j, a) entry in the $N_{s,o}$ node signature matrix respectively. Furthermore, a refers to the column number, while k denotes the total number of columns in both node signature matrices. The function δ in (3.2) is given by:

$$\delta(N_{s,hi}(a), N_{s,oj}(a)) = \frac{|N_{s,hi}(a) - N_{s,oj}(a)|}{\text{range}_a}. \quad (3.3)$$

In (3.3), the δ function determines the absolute difference between two numbers, and divides the result with a normalisation factor (range_a). Note that range_a refers to the difference between the maximum and minimum value inside of column a of the “healthy” node signature matrix ($N_{s,h}$).

The cost matrix can be constructed by applying the HEOM function between each pair of nodes inside the two node signature matrices. The structure of the cost matrix is shown in (3.4). Note that entry C_{ij} inside of the cost matrix represents the result of applying the HEOM function between node i of a “healthy” node signature matrix ($N_{s,h}$) and node j from the current operating node signature matrix ($N_{s,o}$).

$$C = \begin{bmatrix} C_{11} & \cdots & \cdots & C_{1N} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ C_{N1} & \cdots & \cdots & C_{NN} \end{bmatrix} \quad (3.4)$$

The data formats of the cost and node signature matrices differ from each other regarding two aspects. The first being the type of data that each matrix represents. The node signature matrix, which contains node attributes and connectivity between nodes, represents a directed graph, which is a **non-Euclidean** data structure. On the other hand, the cost matrix contains the relative Euclidean distances between the nodes of two graphs, which are **Euclidean** in nature. The second aspect is the dimensions of the data, where the dimensions of the node signature matrix are 15×17 , while the dimensions of the cost matrix are 15×15 .

3.3 Principal component analysis

3.3.1 Overview

Principal component analysis or PCA was first introduced in 1901 by Pearson [123]. PCA is a statistical technique that is used to reduce the data to its essential features, also known as principal components, which are responsible for describing most of the variance (trends) of the variables in the data [124]. In order to find the principal components of the data, orthogonal decomposition must be applied to the covariance matrix of the data [75]. The orthogonal decomposition results in a set of vectors onto which the data can be projected to obtain the principal components. By choosing only a subset of vectors, the dimensionality of the projected data can be reduced so that it is lower than the dimensionality of the original data.

Several studies that focused on fault detection, classification and diagnosis in various industries, such as automotive, electrical, and chemical, have used PCA to visualise high dimensional data [125, 126, 127, 128, 129, 130, 131]. Using PCA to visualise high dimensional data makes it possible to see if there are clusters inside the data, meaning that faults can be separated from the system when under normal operating conditions. Additionally, visualising the highly dimensional data shows how the process data of the system changes over time as a fault is introduced.

3.3.2 Results and discussion

This subsection summarises the results of the PCA plots for both the cost and the node signature matrix data. The PCA plots of the different faults for both the cost and node signature matrix data can be found in Appendix B. The PCA plots in Appendix B were generated using labelled datasets obtained from the Simulink exergy and energy-based TEP model. Note that the labelled datasets contained the data for the cost and node signature matrices, that were sampled at 3-minute intervals.

Cost matrix data

The PCA plots of the different faults in Appendix B contain normal samples (blue dots) and faulty samples (red dots). Normal samples represent the cost matrix data while the system is operating normally, while faulty samples represent the cost matrix data as the specific fault is induced and present within the system. Furthermore, the PCA plots indicate how the cost matrix data evolve over time as the fault is introduced, thereby showing if the faulty samples are distinct from normal samples.

Different degrees of separability were identified between normal and faulty samples using the PCA plots in Appendix B. The degrees of separability used to categorise the various faults were "Clear separation", "Limited separation" and "No separation". "Clear separation" refers to when a clear distinction can be observed between normal and faulty samples, such as the case for fault 1 (see Appendix B). "Limited separation" is when there is a degree of clustering present in either the normal or faulty samples, however, no distinct separation can be observed between normal and faulty samples. An example of "Limited separation" would be fault 8 (see Appendix B), where normal samples were clustered together being surrounded by faulty samples. "No separation" refers to the cases where there is no distinct separation between normal and faulty samples and there was no clustering present in either the normal or faulty samples. The PCA plot for fault 3 (see Appendix B) would be classified as "No separation" since there is no distinct separation or clustering present. The degree of separability for the different faults using the cost matrix data, are summarised in Table 3.1.

Table 3.1: PCA results for cost matrix data.

Fault number	Clear separation	Limited separation	No separation
1	✓		
2	✓		
3			✓
4	✓		
5	✓		
6	✓		
7			✓
8		✓	
9			✓
10		✓	
11		✓	
12		✓	
13		✓	
14			✓
15			✓
16			✓
17		✓	
18		✓	
19			✓
20			✓

Faults categorised as having "Clear separation" would be easier to detect and classify since a clear distinction can be made between normal and faulty samples. On the other hand,

faults categorised as having "Limited separation" would be more difficult to detect and classify because the separation between normal and faulty samples is not that distinct. Lastly, faults categorised as having "No separation" are possibly the hardest types of fault to detect and classify since there is no separation between normal and faulty samples.

Node signature matrix data

Similar to the approach used for the cost matrix data in the previous subsection, the degree of separability of the different faults was determined using the node signature matrix data and PCA plots. The PCA plots for the faults using the node signature matrix data can be found in Appendix B. The degree of separability for the different faults using the node signature matrix data are summarised in Table 3.2.

Table 3.2: PCA results for node signature matrix data.

Fault number	Clear separation	Limited separation	No separation
1	✓		
2	✓		
3	✓		
4	✓		
5	✓		
6	✓		
7			✓
8		✓	
9		✓	
10		✓	
11		✓	
12		✓	
13		✓	
14			✓
15			✓
16		✓	
17		✓	
18		✓	
19			✓
20			✓

3.4 Distance matrices

3.4.1 Overview

The use of distance matrices for process data analytics have been proposed by Melo and coworkers [132]. Distance matrices are symmetrical matrices comprised of the distances between each pair of data points. Typically, the Euclidean distance is used to determine the distance between data points. Furthermore, distance matrices are represented as grayscale colourmaps to illustrate how the distances between data points vary as time progresses.

Distance matrices make it possible to represent the intrinsic data patterns within the process data. Additionally, distance matrices allow one to visualise the changing dynamics of the system, for example when a fault is introduced into the system. It is thus possible to see how a fault affects the system variables.

3.4.2 Distance matrix examples

In this subsection, various distance matrices corresponding to typical process signals will be illustrated. Further, an accompanying interpretation is also provided for each distance matrix example. Note that darker colours inside of a distance matrix correspond to smaller distances between samples, while lighter colours correspond to larger distances. Lastly, note that the axes of the distance matrices correspond to the sample numbers of the data used.

Figure 3.1 below illustrates what a distance matrix would look like for a signal containing random noise. The small changes in colour of the distance matrix correspond to the small step changes and oscillations caused by the random noise. Furthermore, it is apparent that the distance matrix contains mostly dark colours, implying that the distances between data samples are not large and that the signal containing the data does not change much.

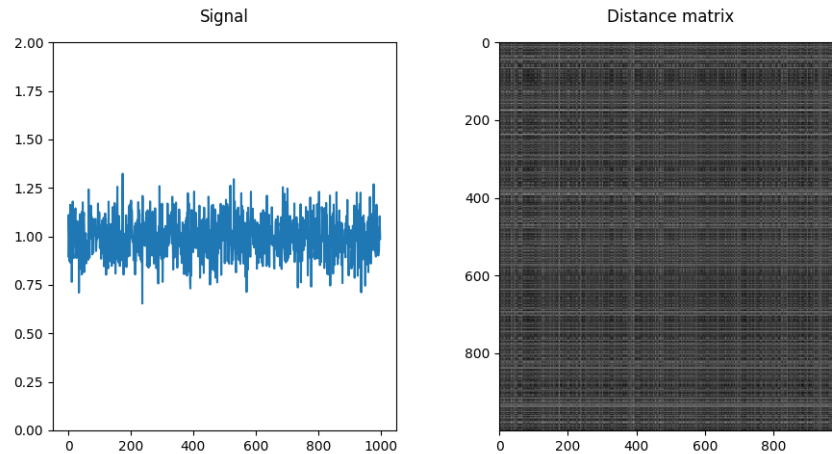


Figure 3.1: Distance matrix for random noise

The distance matrix for a step change in a signal is illustrated in Figure 3.2. The step change inside the signal causes an apparent change in the colour intensity of the distance matrix. As the step change occurs, the colour gradient inside of the distance matrix changes abruptly from a dark colour to a bright colour. Conversely, as the step change ends, the colour gradient inside of the distance matrix abruptly changes from a light colour to a dark colour.

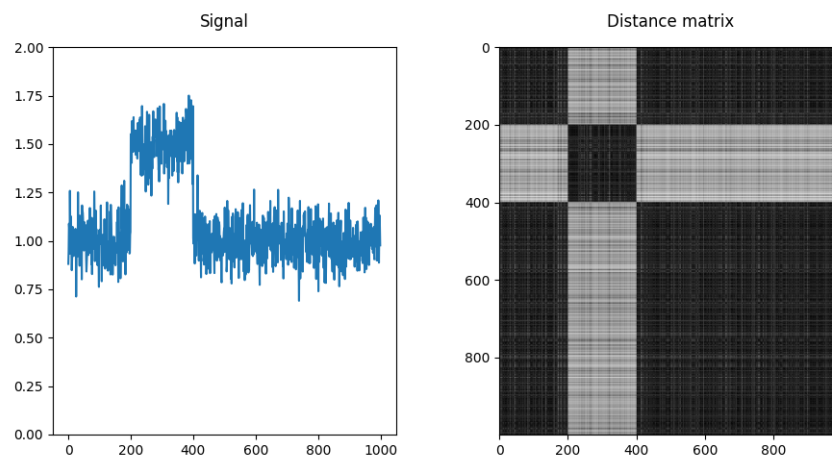


Figure 3.2: Distance matrix for a step change.

A drift in the signal results in a distance matrix similar to the one shown in Figure 3.3. The drift causes the colour gradient inside the distance matrix to gradually change from

darker to lighter. The gradual change from a darker to lighter colour indicates that the drift is causing the signal to gradually increase, causing subsequent data points to vary greatly from initial data points.

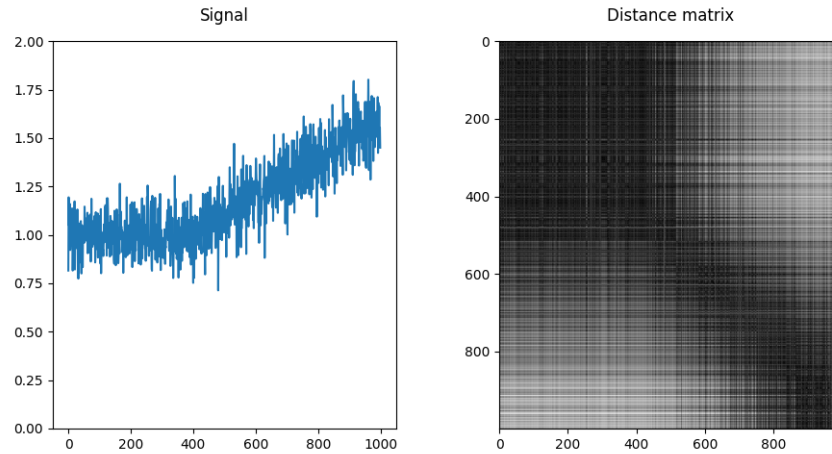


Figure 3.3: Distance matrix for a drift.

Figure 3.4 illustrates the distance matrix that accompanies an oscillation present in the signal. An oscillation in the signal will cause the distance matrix to contain alternating black and white dots. The alternating black and white dots correspond to the local minima and maxima inside the signal, introduced by the oscillations.

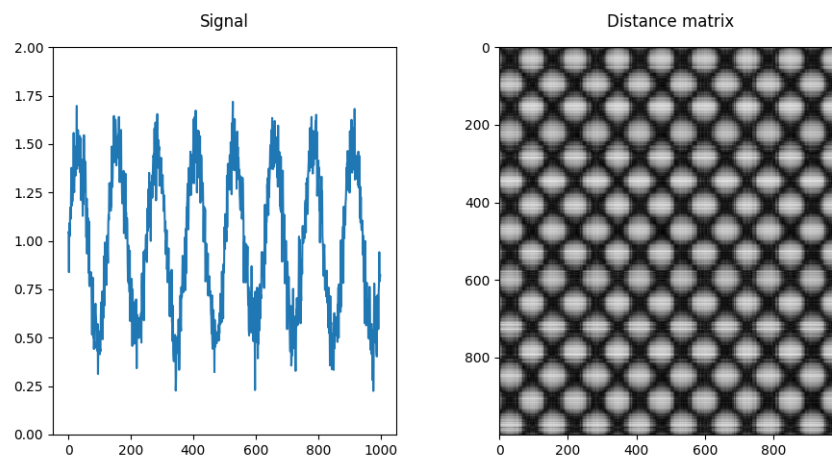


Figure 3.4: Distance matrix for an oscillation.

The distance matrix for a decaying oscillation is shown in Figure 3.5. As can be seen, the oscillatory nature of the decaying oscillation is similar to that in Figure 3.4, showing the alternating black and white dots. However, because of the decay, the black and white dots will not continue indefinitely, but will be followed by a bar of constant colour.

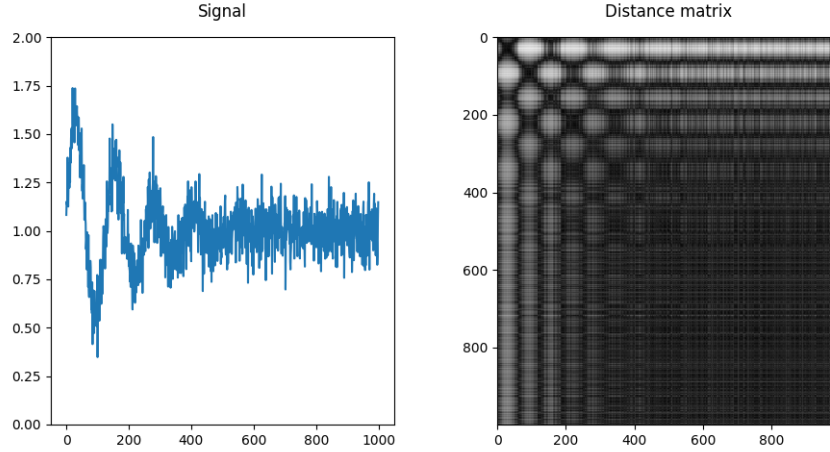


Figure 3.5: Distance matrix for a decaying oscillation.

3.4.3 Results and discussion

In this subsection, the distance matrices for both the cost and node signature matrices data will be summarised and interpreted. In order to interpret the dynamics present within the distance matrices, they are compared to the example distance matrices used in Section 3.4.2. Note that the subsequent results represent the dynamics observed within the distance matrices and not the simulated faults themselves. Furthermore, since the exergy and energy conversions used within the EGBV technique are non-linear, the dynamics observed within the distance matrices do not necessarily correspond to the dynamics of the simulated faults, see Table 2.1.

Unlike the one-dimensional signal examples shown in Section 3.4.2, the cost and node signature matrices contain multiple entries for a given sample. Consequently, to determine the distance between either two node signature matrices or two cost matrices, the matrices will be treated as n -dimensional vectors and the Euclidean distance will be determined between these two n -dimensional vectors.

The distance matrices for the faults using the cost matrix and node signature matrix data can be found in Appendix B. Note that lighter colours represent larger distances between data points, while darker colours represent smaller distances between data points. Lastly,

note that the axes of the distance matrices correspond to the sample numbers of the data used.

Cost matrix data

Figure 3.6 shows the distance matrix for the TEP while under normal operating conditions (NOC), using the cost matrix data. No significant trends or events could be seen within the distance matrix. Furthermore, the small fluctuations in the colour gradient of the distance matrix were attributed to the random noise present in the system, causing small steps and oscillations. The distance matrix depicted in Figure 3.6 was similar to the distance matrix depicting random noise (Figure 3.1).

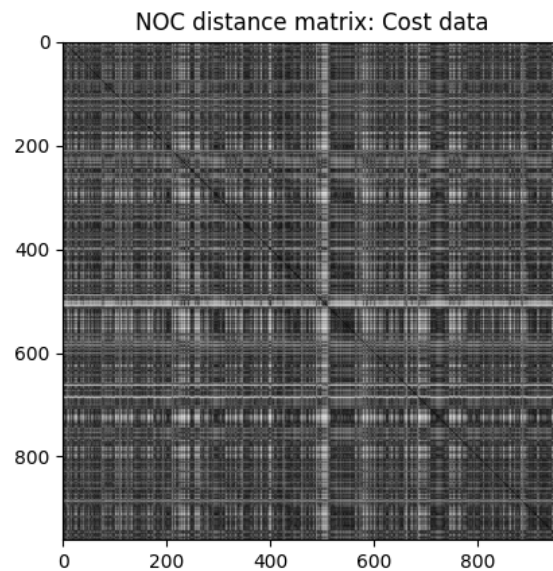


Figure 3.6: Distance matrix for normal operating conditions using cost matrix data.

The dynamics identified in the distance matrices of the various faults using the cost matrix data are summarised in Table 3.3. The distance matrices of the faults using the cost matrix data can be found in Appendix B. The different categories within Table 3.3 corresponded to the dynamics mentioned in Section 3.4.2, with the "Oscillations" category referring to both normal and decaying oscillations. If the distance matrix for a fault indicated the presence of multiple dynamics, then the corresponding dynamics were indicated in Table 3.3. Note that if the distance matrix for a fault did not indicate any trends or events associated with certain dynamics, the "No change" category was chosen since the distance matrix of the fault looked similar to that of random noise (Figure 3.1) or when the system was operating normally (Figure 3.6).

Table 3.3: Distance matrix results for cost matrix data.

Fault number	Step change	Drift	Oscillations	No change
1	✓			
2	✓			
3				✓
4	✓			
5	✓	✓		
6	✓	✓		
7	✓		✓	
8	✓			
9	✓			
10	✓			
11	✓	✓		
12	✓		✓	
13		✓	✓	
14				✓
15				✓
16	✓			
17	✓	✓		
18	✓		✓	
19				✓
20	✓			

Faults within the "No change" category would be difficult to detect and classify, since their dynamics are similar to that of random noise or the system operating normally. Moreover, the ease with which faults with other dynamics can be detected and classified will depend on the size of the dynamics. If the dynamics within the fault, for example step changes and oscillations, are too small, then it would be difficult to detect and classify faults since the resulting distance matrix would be almost similar to that of the system

being under normal operating conditions. In contrast, if the dynamics are relatively large then the distance matrix for the fault would differ significantly from the distance matrix corresponding to normal conditions, making it easier to detect and classify the fault.

Node signature matrix data

Figure 3.7 shows the distance matrix for the system while operating under normal conditions using the node signature matrix data (NS data). It is evident from the distance matrix, shown in Figure 3.7, that there were no major trends or events present in the system variables. Instead, the system variables appeared to contain mostly random noise, which caused small fluctuations in colour. Furthermore, these small fluctuations in colour were due to the small step changes and oscillations present in the random noise. The distance matrix shown in Figure 3.7 was similar to the distance matrix associated with random noise (Figure 3.1).

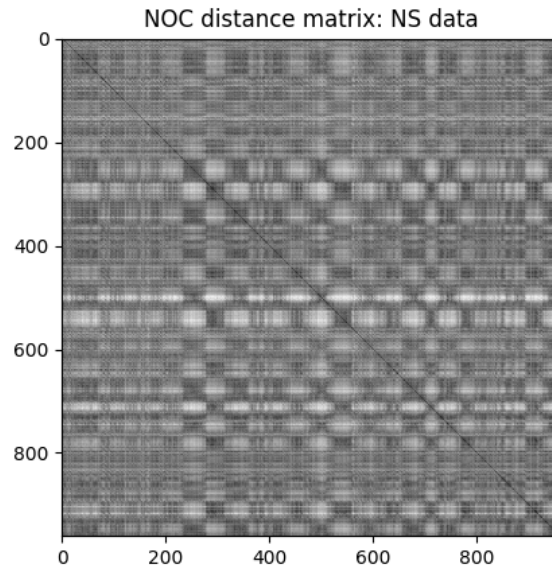


Figure 3.7: Distance matrix for normal operating conditions using the node signature matrix data.

The dynamics identified in the distance matrices of the different faults using the node signature matrix data are summarised in Table 3.4. Note the distance matrices obtained with the node signature matrix data for the different faults can be found in Appendix B.

Table 3.4: Distance matrix results for node signature matrix data.

Fault number	Step change	Drift	Oscillations	No change
1	✓		✓	
2	✓			
3	✓	✓		
4	✓			
5	✓		✓	
6	✓	✓		
7	✓		✓	
8	✓			
9				✓
10	✓		✓	
11	✓	✓		
12	✓	✓		
13			✓	
14				✓
15				✓
16				✓
17	✓	✓		
18	✓	✓		
19				✓
20	✓		✓	

3.5 Self-organising maps

3.5.1 Overview

Self-organising maps, or SOMs, were developed by Kohonen [133] and are a type of neural network that contain only a single layer. SOMs are commonly used for visualisation purposes, by employing unsupervised learning [134]. Moreover, SOMs use a special type of unsupervised learning, called competitive learning. The process of competitive learning and the general training procedure of a SOM are discussed below.

Initially, the weights of the neurons inside the single-layered neural network are initialised. Next, a random sample of the data is chosen and compared with the weights of the neurons. The neuron whose weights most closely resemble the input sample is chosen as the best matching unit. The nodes surrounding the best matching unit are then also identified. Both the best matching unit and its surrounding nodes are updated so that their weights more closely resemble the input data sample. The process is repeated for the remaining samples, leading to the emergence of clusters/neighbourhoods corresponding to different groups found in the data. These clusters/neighbourhoods are represented using a 2D grid. This SOM training process is illustrated in Figure 3.8 [135], with the best matching unit corresponding to the red output node and the surrounding green nodes corresponding to the neighbouring nodes. Note that the weights of the nodes inside the SOM are indicated as $\mathbf{W}_i$, and the output nodes of the SOM are indicated as a rectangular 2D grid.

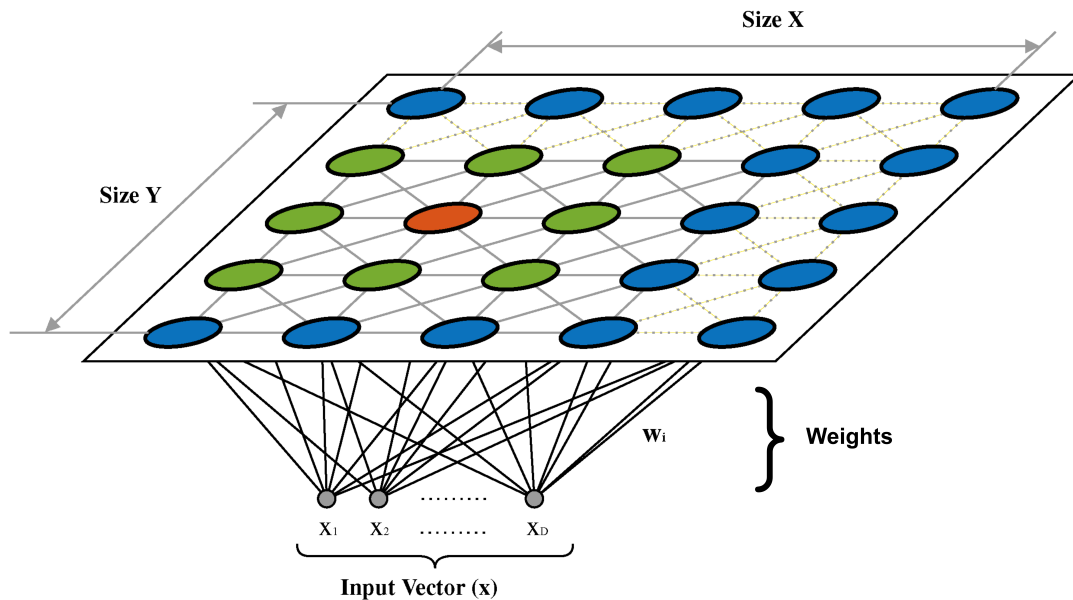


Figure 3.8: Structure of a self-organising map (Adapted from [135]).

Various studies have used the SOM to visually discover clusters within the data for FDD

purposes [136, 137, 138]. SOMs can discover the above-mentioned clusters within the data, since they represent the distribution of the data [139]. The distribution of the data is obtained during the iterative training process of the SOM, where the similarity of the samples within the data is compared with each other. Similar samples cause the weights of neighbouring neurons inside of the neural network to converge, forming clusters during the training process of a SOM.

3.5.2 Results and discussion

In this section, the output of the resulting SOMs for the data of the cost and node signature matrices are summarised and interpreted. In addition, the results will be shown when the system is operating under normal conditions. The output of the SOMs for the faults can be found in Appendix B. Lastly, note that the colour legends accompanying each subsequent figure represent how closely the weights of a neuron correspond to the weights of neighbouring neurons. Darker colours indicate that the neuron's weight differs greatly from the weights of neighbouring neurons, and lighter colours indicate that the neuron's weight more closely resembles the weights of neighbouring neurons.

Cost matrix data

Looking at the output of the SOM, shown in Figure 3.9, no apparent clusters were found in the data. However, this was expected, since the system was under normal operating conditions; consequently, the system variables would remain largely unchanged over time, since no fault was introduced, which could have otherwise caused the system variables to vary significantly over time. The random variations in colour, shown in Figure 3.9, could be attributed to the random noise present in the system, while under normal operating conditions (NOC). The random noise present in the system will cause some of the data points to vary greatly from most of the other data points.

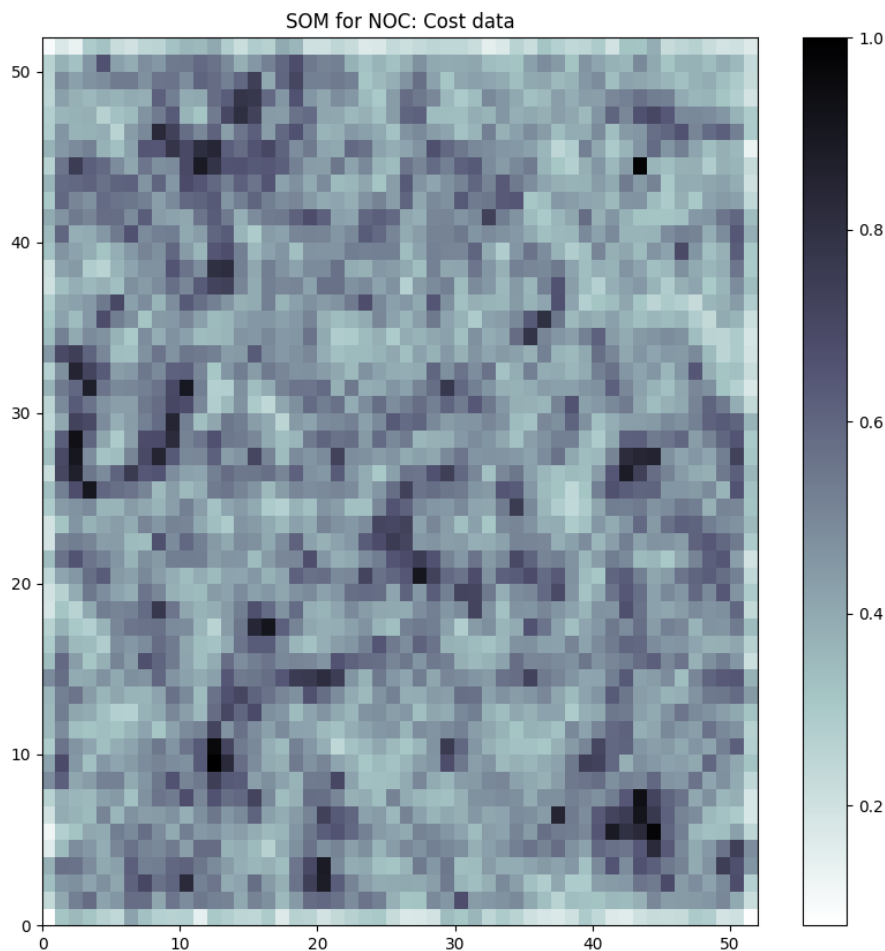


Figure 3.9: Output of self-organising map for normal operating conditions, using cost matrix data.

Different degrees of segmentation were identified in the output of the SOMs in Appendix B. The degrees of segmentation were categorised as "Clusters", "Partial clusters" or "No clusters". The "Clusters" category applies if the output of the SOM indicated the presence of clusters, such as fault 2 (see Appendix B). If no clusters were identified but the output of the SOM showed the presence of some partial segmentation, the category "Partial clusters" was chosen. Fault 10 (see Appendix B) would be classified as belonging to the "Partial clusters" category, due to the dark lines that partially divide the data, but do not form clusters. Faults belonging to the category "No clusters" showed no signs of clusters or partial segmentation in the output of their SOM. An example of a "No clusters" fault would be fault 3 (see Appendix B), where no clusters or partial segmentation is present within the output of the SOM. The output of faults categorised as "No clusters" will have outputs similar to the output of the system that operates normally (Figure 3.9). The degree of segmentation observed in the output of the SOMs for the different faults, using the cost matrix data, is summarised in Table 3.5.

Table 3.5: Self-organising map results for cost matrix data.

Fault number	Clusters	Partial clusters	No clusters
1	✓		
2	✓		
3			✓
4	✓		
5	✓		
6	✓		
7	✓		
8	✓		
9			✓
10		✓	
11		✓	
12	✓		
13	✓		
14			✓
15			✓
16			✓
17		✓	
18	✓		
19			✓
20			✓

Faults falling under the "Clusters" category might be easier to detect and classify compared to the other categories. The presence of clusters indicates that the fault significantly

changes the system variables as it is introduced into the system, resulting in distinct clusters. One of the clusters might correspond to the system operating normally before the fault is induced and present within the system, implying that there is a distinct difference between "normal" and "faulty" samples. However, it is also possible that one of the clusters might contain a mixture of "normal" and "faulty" data, making it more difficult to detect and classify the fault, since there is no separation between "normal" and "faulty" data.

Faults belonging to the "Partial clusters" category might be more difficult to detect and classify compared to the "Clusters" category. Unlike the "Clusters" category, the "Partial clusters" category only indicates partial segmentation within the data and not distinct segmentation that is associated with the presence of clusters. Lastly, faults belonging to the "No clusters" category might be more difficult to detect and classify compared to the other categories, since there is no segmentation present within the output of the SOM. Thereby indicating that the SOM was not able to detect significant differences within the data between "normal" and "faulty" samples.

Node signature matrix data

No apparent clusters could be observed in Figure 3.10, when looking at the output of the SOM, using the node signature matrix data (NS data). This was due to the system being under normal conditions, and therefore no clusters were observed, since there were no faults present to drastically change the system variables. The random variation in the colour of the output shown in Figure 3.10, could be attributed to the presence of random noise in the system variables.

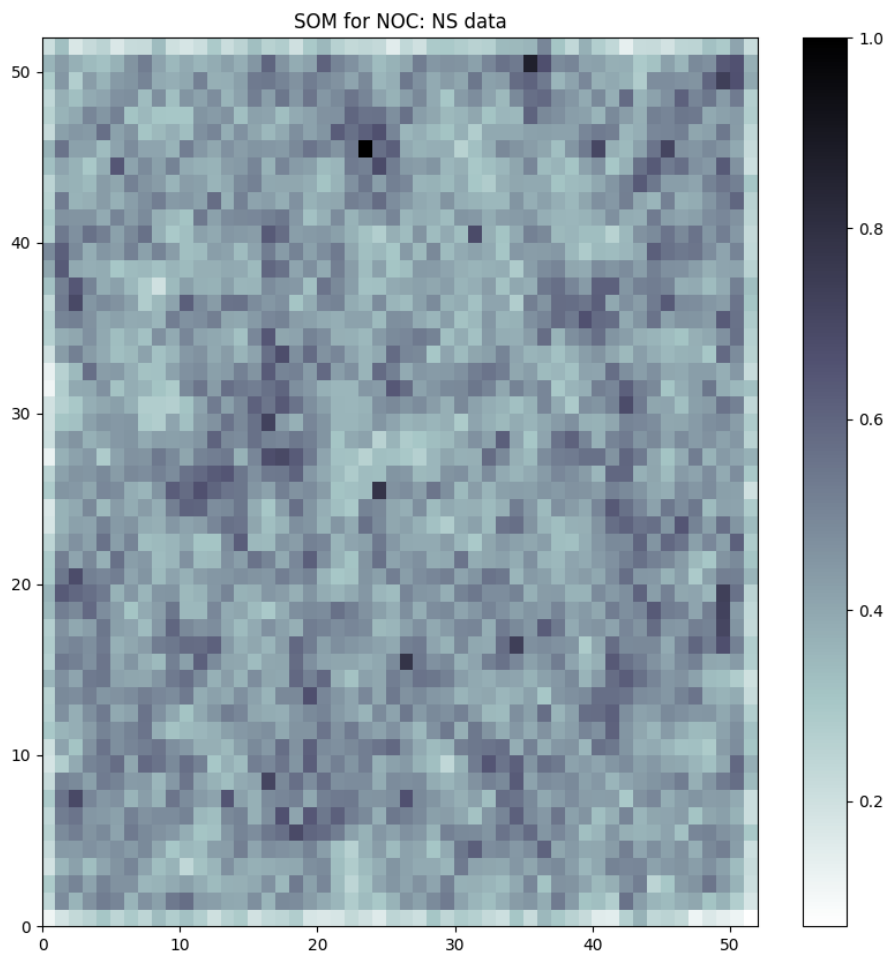


Figure 3.10: Output of self-organising map for normal operating conditions, using the node signature matrix data.

Following a similar approach that was used with the cost matrix data in the previous subsection, the degree of segmentation observed in the output of the SOMs using the node signature matrix data is summarised in Table 3.6. The output of the SOMs for the node signature matrix data can be found in Appendix B.

Table 3.6: Self-organising map results for node signature matrix data.

Fault number	Clusters	Partial clusters	No clusters
1	✓		
2		✓	
3	✓		
4	✓		
5	✓		
6	✓		
7	✓		
8	✓		
9		✓	
10		✓	
11		✓	
12	✓		
13	✓		
14			✓
15			✓
16			✓
17		✓	
18	✓		
19			✓
20			✓

3.6 Distinctness of data representations

This section aims to summarise the results obtained with the EDA techniques for both the cost and the node signature matrix data for the different faults. The metric that was used to summarise the results was distinctness. Furthermore, distinctness in this context refers to the degree to which the faulty data differed from the normal data.

The categories used to denote the degree of distinctness included "Apparent", "Partial", and "None". The "Apparent" category was chosen if the output of the self-organising map indicated the presence of clusters, the PCA plots showed clear separation between normal and faulty samples, and the distance matrix indicated that the dynamics inside the data changed. Conversely, the "None" category was selected if the fault data showed no clusters

using the self-organising map, the distance matrix showed no change in the dynamics of the data, and the PCA plot showed no clear separation between normal and faulty samples. Finally, the "Partial" category was used when the fault data at least showed some partial clustering based on the self-organising map output, some limited separation between normal and faulty samples in the PCA plot, or some change in dynamics based on the resulting distance matrix.

The degree of distinctness for the cost matrix and node signature matrix data is summarised in Tables 3.7 and 3.8, respectively. Faults categorised as having "Apparent" distinctness were predicted as being easier to detect and classify compared to faults that were categorised as having "Partial" or "None" distinctness. Meanwhile, faults categorised as having "None" distinctness were predicted to be more difficult to detect and classify compared to faults belonging to other categories of distinctness. Although the results of the visual EDA techniques indicated that some faults did not show any distinctness, modern ML techniques (see Chapters 4 and 5) have the ability to apply multiple non-linear transformations to data [140]. These non-linear transformations make it possible to represent the data in a more distinct manner, thereby potentially making it easier to detect and classify faults that initially had no apparent distinctness.

Cost matrix data

Table 3.7: Degree of distinctness for cost matrix data.

Fault number	Apparent	Partial	None
1	✓		
2	✓		
3			✓
4	✓		
5	✓		
6	✓		
7		✓	
8		✓	
9			✓
10		✓	
11		✓	
12		✓	
13		✓	
14			✓
15			✓
16		✓	
17		✓	
18		✓	
19			✓
20		✓	

Node signature matrix data

Table 3.8: Degree of distinctness for node signature matrix data.

Fault number	Apparent	Partial	None
1	✓		
2		✓	
3	✓		
4	✓		
5	✓		
6	✓		
7		✓	
8		✓	
9		✓	
10		✓	
11		✓	
12		✓	
13		✓	
14			✓
15			✓
16		✓	
17		✓	
18		✓	
19			✓
20		✓	

3.7 Conclusion

In this chapter, EDA was applied to the data of both the cost and node signature matrices. It was discovered that for both the cost and node signature matrices, certain faults could be easily distinguishable from the system under normal operating conditions, whereas other faults could not be easily distinguished. Furthermore, when using the node signature matrix data, certain faults could be more easily distinguishable from the system under normal operating conditions compared to the cost matrix data and vice versa. Consequently, these results will prove to be valuable in later chapters, where phenomena such as why certain models may perform better than others and explanations why models might struggle with certain faults.

Chapter 4

Machine learning for the cost matrix data

4.1 Introduction

This chapter provides the results of applying ML techniques to the data of the cost matrices for fault detection and classification. Initially, a brief overview of the different ML techniques that were used, will be given, followed by the detection and classification results obtained by the respective ML techniques. The detection and classification performances of the different ML techniques will be compared against each other to identify the best performing ML technique. Furthermore, the best-performing ML technique for the cost matrix data will be used in the following chapters for comparison against the best-performing ML technique identified for the node signature matrix data.

4.2 The cost matrix structure

As mentioned in Section 2.5.2, the cost matrix is a square matrix that contains Euclidean data. The set of informative entries inside the cost matrix, used to detect and classify faults, may be distributed in several ways, and is indicated in Figure 4.1. The informative entries may be globally distributed and sparse (the red blocks indicated in Figure 4.1). It is also possible that the informative features are more locally grouped and span across a single row or column (orange and green blocks in Figure 4.1) or across multiple rows and columns (blue block in Figure 4.1). Various ML techniques that can exploit the different possible distributions of the informative features are discussed in the next section.

$$C = \begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1N} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2N} \\ C_{31} & C_{32} & C_{33} & \cdots & C_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{N1} & C_{N2} & C_{N3} & \cdots & C_{NN} \end{bmatrix}$$

Figure 4.1: Possible distributions of informative entries inside the cost matrix.

4.3 Machine learning techniques

Three ML techniques were identified to be used in the current study for the cost matrix data, keeping in mind the possible distributions of useful entries mentioned in Section 4.2. The three ML techniques include deep neural networks (DNN) and convolutional neural networks (CNN) comprising of 1D and 2D variants. The mentioned ML techniques were chosen because of their ability to process the data in the cost matrix in different ways [141]. CNNs have a superior ability to extract potential local features [142], while DNNs are more suited to extract more complex global features [143]. Note that the DNN and CNNs consist of several processing layers, allowing them to extract increasingly more abstract features with each subsequent layer from the input data [144].

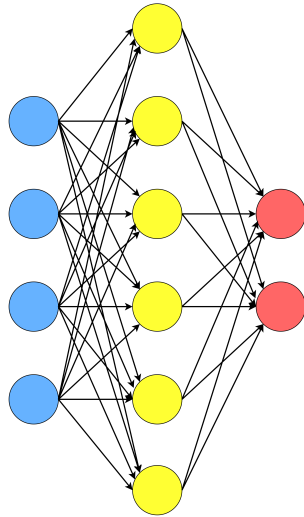
CNNs are a popular choice of ML techniques that have been widely used in the field of fault monitoring [145, 146, 147]. Furthermore, the grid-like nature of the cost matrix data also justifies the use of CNNs, since CNNs can exploit the spatial relationships present within the grid-like data [148]. DNNs on the other hand have also been widely used for fault monitoring activities and are typically used as a baseline against which other ML techniques can be compared [149, 150, 151].

4.3.1 Deep neural networks

DNNs are feed-forward type networks, where the output of one layer is used as input for the next layer. A DNN consists of an input layer, an output layer, and several hidden layers [152]. Unlike simple neural networks, deep neural networks contain at least two or more hidden layers (Figure 4.2), and in these hidden layers, the complex non-linear behaviour of the data can be extracted [153]. Furthermore, each different layer inside the

DNN (input, output and hidden layers) contains several neurons.

Simple neural network



Deep neural network

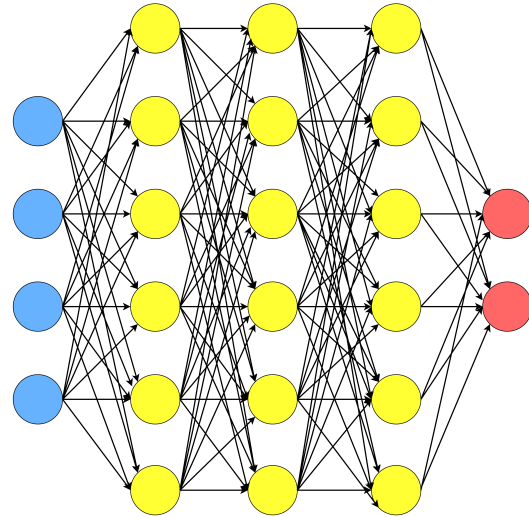


Figure 4.2: Architectures for a simple neural network (left) and a deep neural network (right) (Adapted from [154]).

The neurons inside the various layers of a DNN perform different tasks. The neurons inside the input layer accept the data that have to be processed and the neurons inside the output layer provide the output of the processed data of the DNN [152]. The neurons inside the hidden layers can be seen as small mathematical processing units, which when combined with other neurons in adjacent hidden layers, can be used to learn the complex relationship between the input features and the outputs [155]. A depiction of the structure of a neuron can be seen in Figure 4.3 [156].

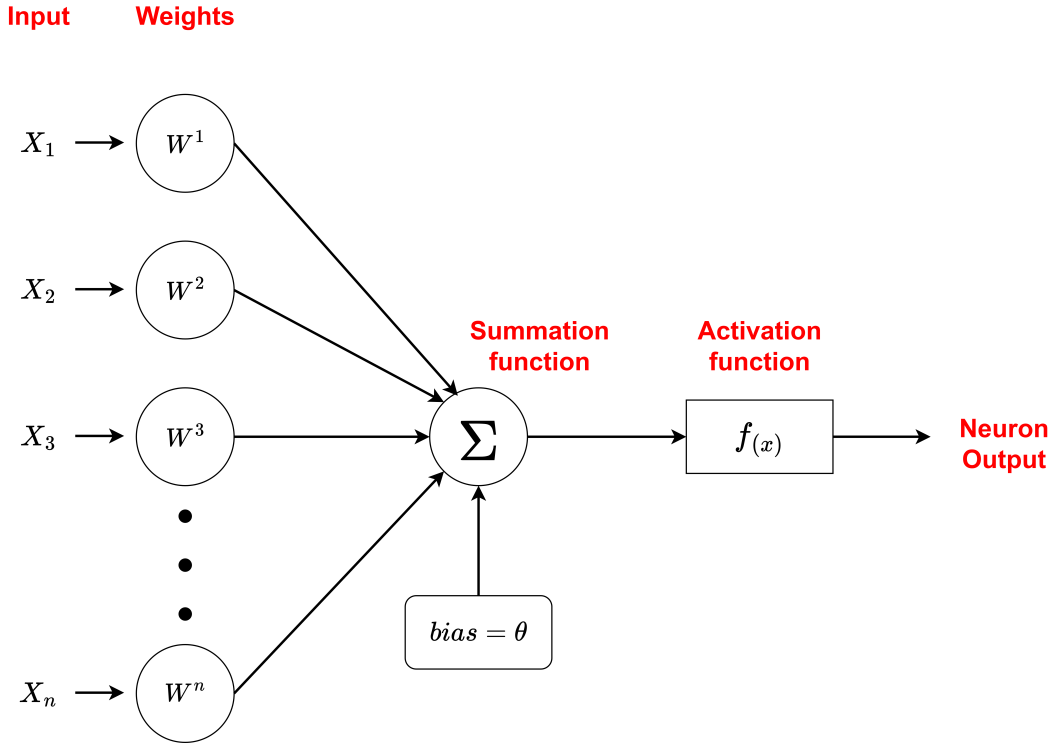


Figure 4.3: Structure of a neuron (Adapted from [156]).

A neuron accepts the output of the previous layer (denoted as X_1 to X_n in Figure 4.3), as input. Note that the outputs of the previous layer are each multiplied with a weight value denoted as W_1 to W_n . After multiplying each of the inputs (X_1 to X_n) with their respective weights, they are added together, and a bias term (denoted as θ) is also added. Once the weighted sum of the inputs and the bias term have been determined, the result is fed into an activation function (denoted as $f_{(x)}$ in Figure 4.3) [157]. An activation function is used to introduce non-linearity into the network, thereby allowing data that is not linearly separable to be non-linearly projected onto abstract spaces which might be more linearly separable [158]. Examples of commonly used activation functions are shown below in Figure 4.4 [159].

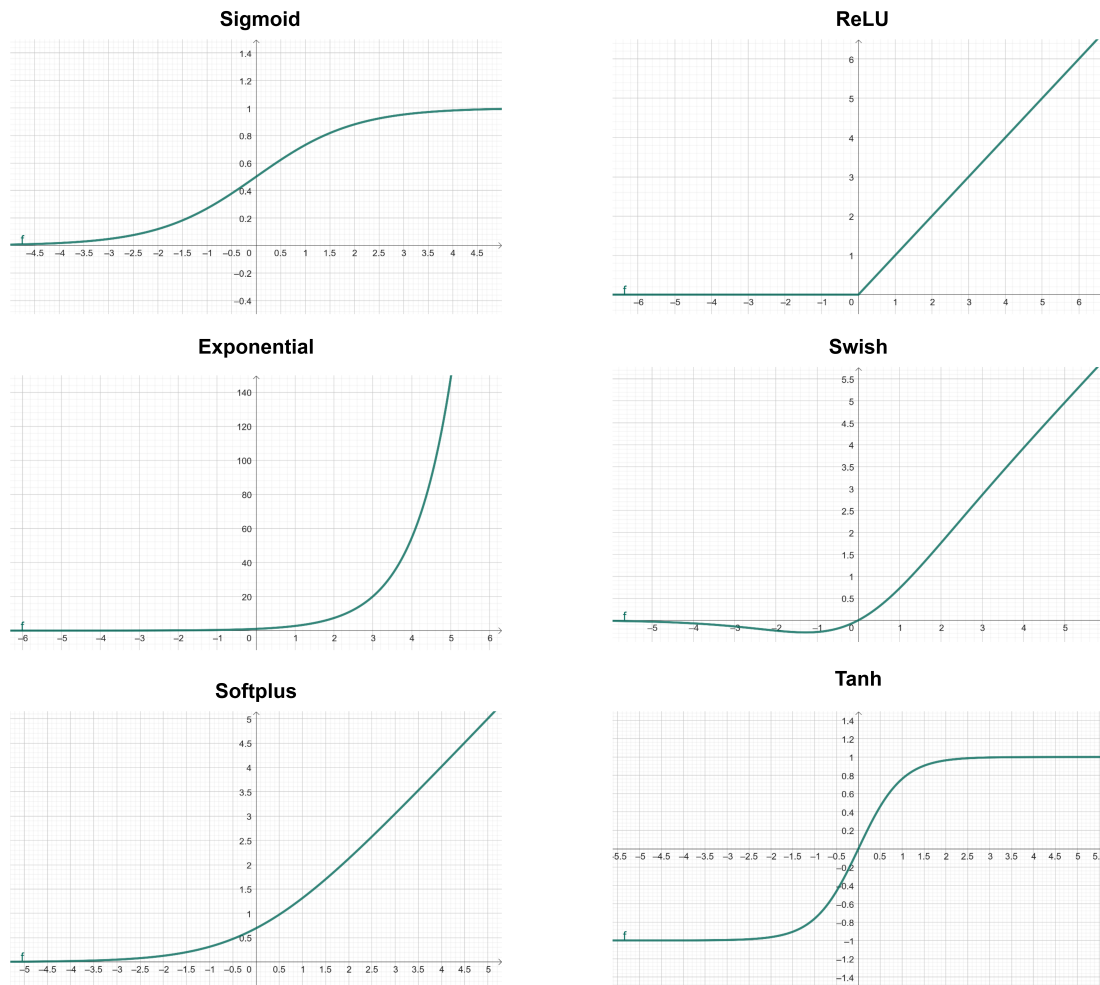


Figure 4.4: Examples of common activation functions (Adapted from [159]).

During supervised training (see Section 2.4.3 with regards to the difference between supervised, unsupervised and semi-supervised ML), the weights and biases inside the DNN are adjusted, to find a relatively good mapping between the inputs and the desired outputs [160]. A loss function is used to indicate how good the mapping is, by comparing the output of the DNN with the actual desired output [152]. During the training process, the gradients of the different weights and biases with respect to the loss function are calculated and used to adjust the respective weights and biases to minimise the loss function, thereby attempting to reduce the difference between the predictions of the DNN and the actual desired outputs. The backpropagation algorithm is used to determine the gradients of the weights and biases with respect to the loss function [161].

Another algorithm, called an optimizer, is further used to adjust the weights and biases in a specific way by using the gradients calculated with the backpropagation algorithm [162]. There are several optimizers, each with its own respective advantages and disadvantages [163]. Popular optimizers include Stochastic Gradient Descent (SGD), Adagrad, Adadelata,

RMSProp, and Adam [164].

During supervised training of a neural network, a phenomenon called overfitting may occur. Overfitting refers to when a neural network has memorised all the training data including the noise present in the data and is unable to generalise to unseen data [86]. Luckily a set of techniques, called regularisation techniques, have been proposed to combat the problem of overfitting [165]. One of the most popular regularisation techniques is Dropout, where some of the neurons inside the neural network are randomly switched off during training [166]. Randomly switching off some of the neurons, allow the remaining neurons to learn more robust features, and thus allowing them to generalise better to unseen data [167]. An example of the Dropout regularisation technique can be seen in Figure 4.5 [166].

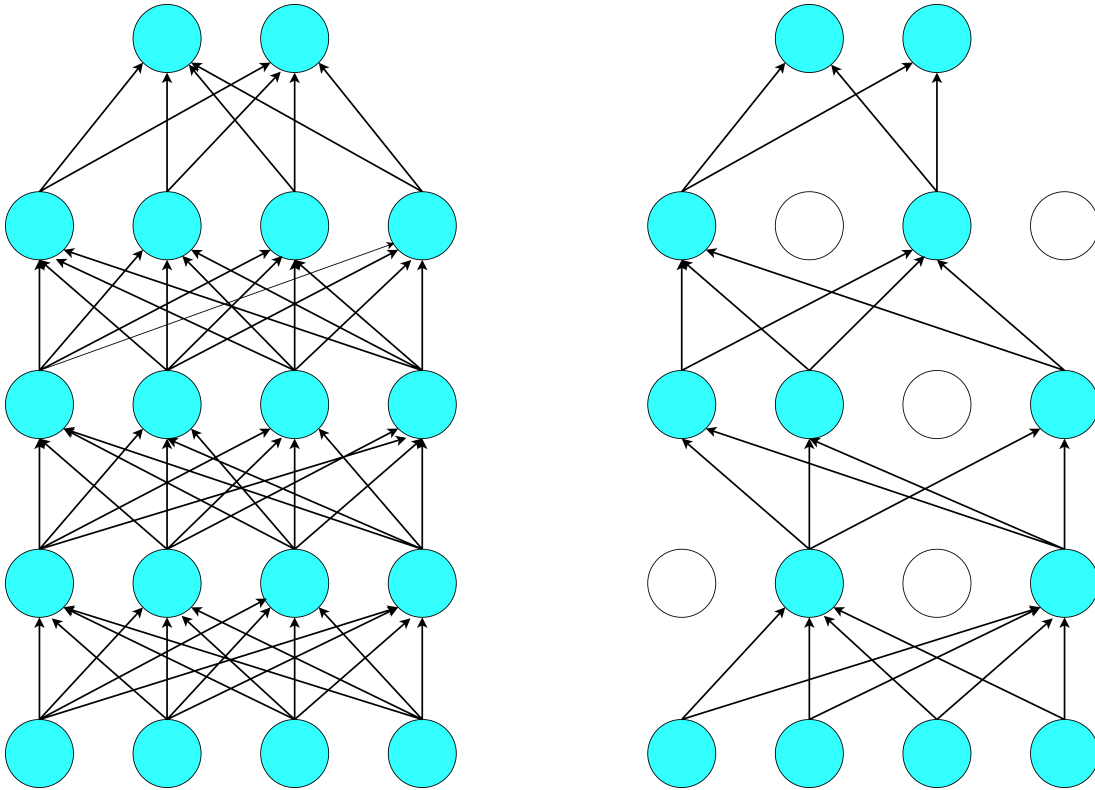


Figure 4.5: A neural network without Dropout (left) and a neural network with Dropout applied (right) (Adapted from [166]).

4.3.2 Convolutional neural networks

CNNs are a type of neural network designed to efficiently process data with well-defined grid-like structures such as 2D image data or 1D time series data [168]. In contrast to a DNN, which only contains hidden layers, a CNN contains additional layers called convolutional and pooling layers [169]. The basic structure of a CNN, consisting of convolutional,

pooling, and fully connected layers, is illustrated in Figure 4.6 [170]. The fully connected layers inside a CNN in Figure 4.6 are similar to the hidden layers of a DNN shown in Figure 4.2.

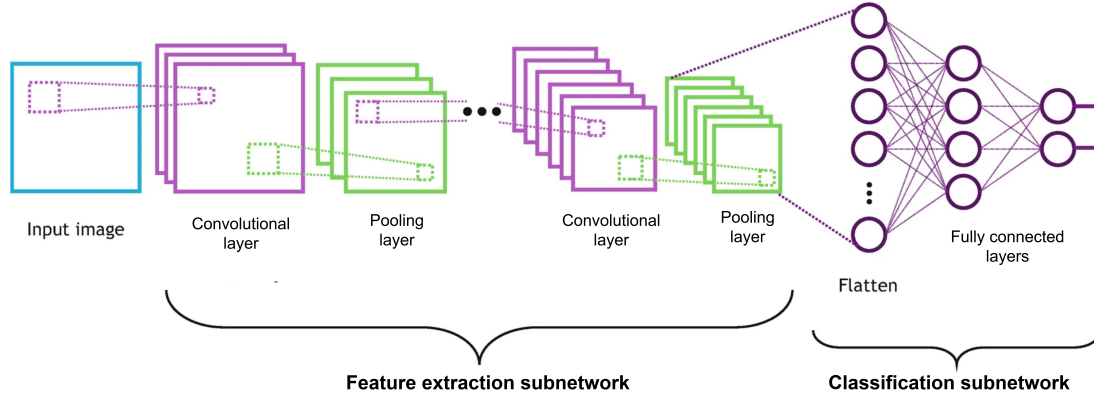


Figure 4.6: Structure of a CNN (Adapted from [170]).

A CNN consists of two subnetworks (Figure 4.6), namely a feature extraction network that learns the features present within the data and a classification network [171]. The feature extraction network consists of the convolutional and pooling layers, which extract features and reduce the dimensionality of the extracted features respectively. On the other hand, the classification network which consists of the fully connected layers is used to learn the mapping between the extracted features and the desired output [172]. Note that the extracted features produced from the feature extraction network are highly dimensional and need to be flattened before being fed into the classification network.

A convolutional layer consists of multiple small arrays, each containing numbers, and are called kernels [173]. These kernels are slid across the input, and as they are being slid across the input, the convolution operation is applied between each kernel and the patch of input data it covers. The convolution operation that takes place inside a convolutional layer requires computing the element-wise product between the entries within a kernel and the corresponding entries of the input data patch it covers followed by adding the result [169]. The output produced by the convolutional layer by applying a single kernel to the input is called a feature map or an activation map [172].

Examples of the convolution operation performed in two dimensions between input data and a single kernel can be seen in Figure 4.7 [174], where the two-dimensional input data is represented by the blue matrix, while the resulting feature map is represented by the green matrix. Note that the values inside the two-dimensional kernel are represented as the small numbers in the lower right corners of the shaded blue region, while the output of the corresponding convolution operation is indicated as the shaded green block of the output feature map.

Convolution process steps

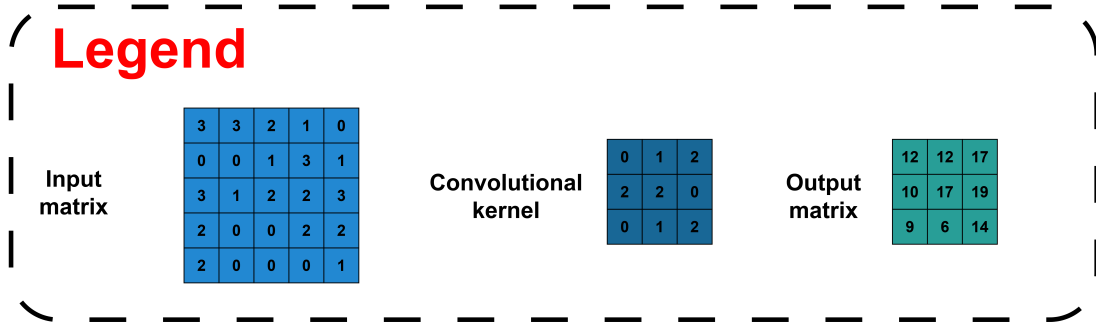
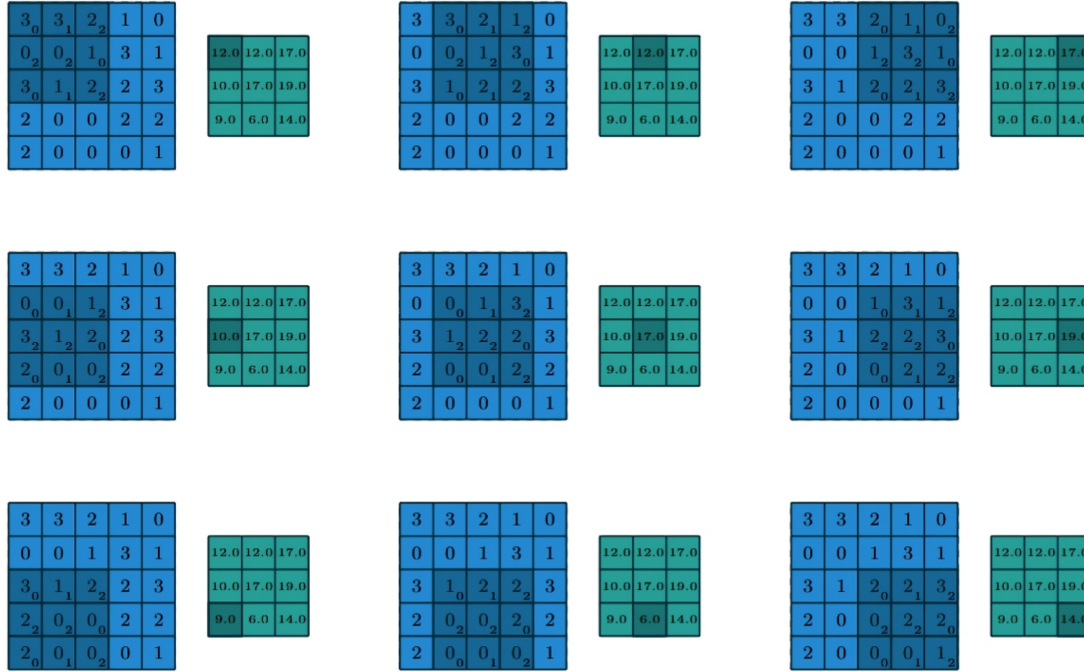


Figure 4.7: Two-dimensional convolution operation performed by convolutional layer (Adapted from [174]).

An example of the convolution operation performed in one dimension is shown in Figure 4.8 [175]. Note that the only differences between the convolution operations shown in Figures 4.7 and 4.8 are with respect to dimensionality, otherwise they function the same. In Figure 4.7 both the input data and the kernel are two-dimensional, with the kernel being able to slide across the input data in two dimensions. On the contrary, for the one-dimensional case illustrated in Figure 4.8, both the input data and the kernel are one-dimensional, and the kernel is only able to move in a single direction.

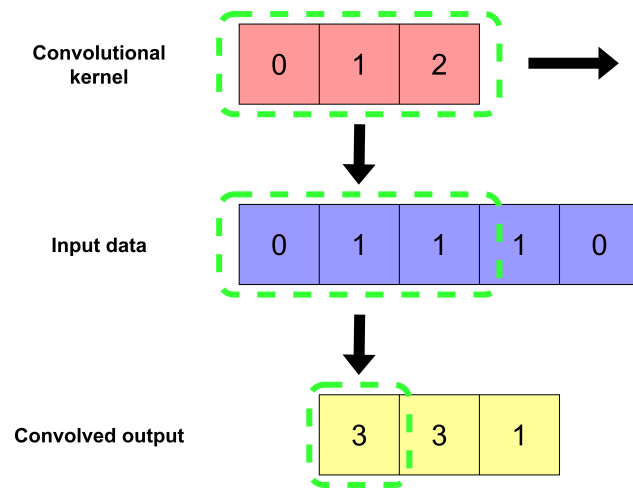


Figure 4.8: One-dimensional convolution operation performed by convolutional layer (Adapted from [175]).

The number of units by which a kernel is moved over the input data before the convolution operation is applied, is known as the stride of the kernel. An example of a convolutional kernel with a stride of 1 and a stride of 2 can be seen in Figure 4.9 [176]. With a stride of 1, the kernel is shifted only one unit before performing the convolution operation, while with a stride of 2, the kernel is shifted two units before performing the convolution operation. For 2D convolutional kernels, the stride parameter is applied in both dimensions, whereas, for 1D convolutional kernels, the stride parameter is only applied in a single direction.

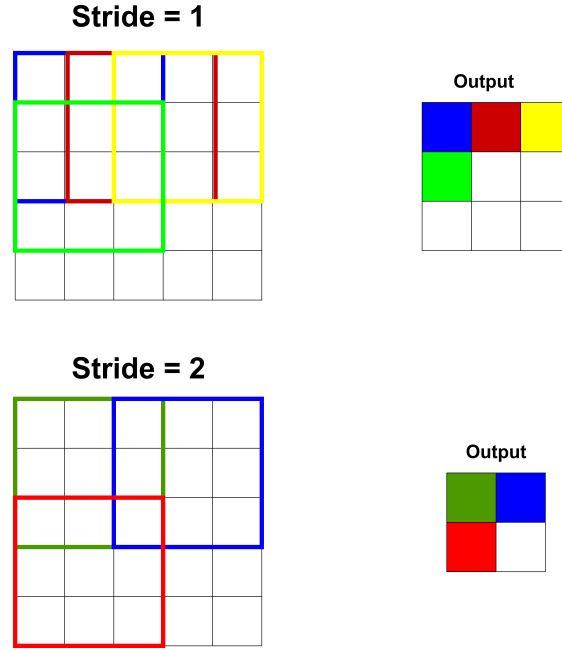


Figure 4.9: A convolutional kernel with a stride of 1 (top) and a convolutional kernel with a stride of 2 (bottom) (Adapted from [176]).

The kernels within the different convolutional layers can vary in size. Smaller kernels can extract more detailed local features from the input data, whereas larger kernels can extract more global features from the input data [177]. However, if the kernels are too small, the CNN will not be able to extract any useful features from the data, and if the kernel is too large the CNN might mix different local features, and thereby affecting the diagnostic capabilities of the CNN [178].

Each of the different kernels within a convolutional layer looks for a specific pattern that resides within the given input [179]. Therefore, the feature map produced by a single kernel will indicate whether a specific pattern was found within the input. Moreover, having multiple convolutional layers allows more complex and abstract patterns to be detected and extracted from the input data [180]. The kernels inside the initial convolutional layers can extract simple and basic features from the data, while the kernels in subsequent convolutional layers receive the output feature maps of previous convolutional layers as input, and are thus able to extract more complex and abstract features compared to previously extracted simple features.

After applying the convolution operation, the output feature maps are sent through an activation function [181]. Similar to the DNN, the activation function applied to the feature maps is used to introduce non-linearity, enabling subsequent convolutional layers to learn more complex features.

Convolutional layers are typically followed by pooling layers, which are used to reduce the

dimensionality of feature maps [182]. The pooling operation performed inside a pooling layer reduces the dimensionality of the feature maps, by calculating a summary statistic for specific patches within the feature maps and then replacing these patches with their corresponding summary statistic [183]. In addition, the pooling operation performed by pooling layers reduces the number of parameters inside the model and helps to avoid overfitting [184].

Two of the most commonly used pooling methods include max and average pooling [185]. With max pooling the maximum value inside of a patch is used to represent a patch, while with average pooling the average of the patch is calculated and used to represent the patch. Examples of max and average pooling can be observed in Figure 4.10 [148].

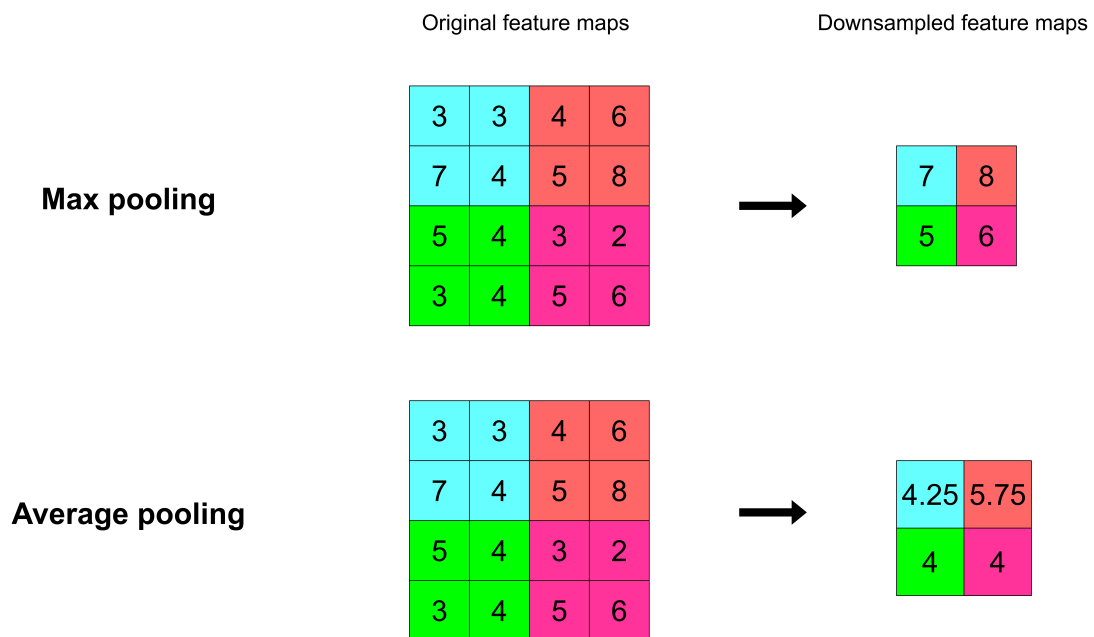


Figure 4.10: Max pooling (top) and average pooling (bottom) (Adapted from [148]).

The convolutional and pooling layers are followed by the flattening of the highly dimensional output feature maps before being passed onto the fully connected layers. As illustrated in Figure 4.6, the fully connected layers inside a CNN define the classification subnetwork. Within this classification subnetwork, fully connected layers are responsible for learning the mapping between the extracted feature maps and the desired output. Similar to the hidden layers of a DNN, mentioned in section 4.3.1, the fully connected layers of a CNN consist of trainable weights and biases as well as non-linear activation functions.

4.4 Training, validation and testing datasets description

The training dataset comprised 20 simulation runs for each fault condition as well as for normal operating conditions. Each simulation run represented 48 hours of plant operation, sampled at 3-minute intervals, with the respective fault condition being introduced after 8 hours of simulated time. Furthermore, 5% of the complete training dataset was reserved as a validation set for hyperparameter tuning. For both the training and validation datasets, stratification was used to ensure an equal representation of all operating conditions, thereby reducing the risk of the models developing a bias toward any specific condition during training.

The test dataset was constructed using an equal distribution of the various operating conditions and consisted of 2 simulation runs per condition, with each run again corresponding to 48 hours of simulated time sampled at 3-minute intervals. As in the training phase, the faults were introduced after 8 hours of simulation in the test runs. Distinct random seeds were used for all simulations in both the training and testing phases to ensure statistical independence between runs.

4.5 Architectures of machine learning models

Hyperparameter tuning was used to determine the optimal architectures for the three ML models mentioned in Section 4.3. The tuning of hyperparameters plays an important role, since the choice of hyperparameters significantly influences the performance of the ML algorithm [186]. The Optuna hyperparameter tuning framework was used to perform the hyperparameter tuning to find the optimal architectures [187].

Table 4.1 explains the symbols that will be used when describing the architectures for the different ML models. The optimal architectures for the three ML models, that were obtained with hyperparameter tuning, are summarised in Table 4.2. Additional aspects of the hyperparameter tuning, results which are not mentioned in Table 4.2 or in this section, can be found in Appendix C.

Table 4.1: Symbol legend for different types of layers.

Symbol	Description
Input	Refers to the input layer of the model.
FC	Refers to a fully connected layer. The succeeding number in parentheses refers to the number of neurons within the fully connected layer.
Conv	Refers to a convolutional layer, and depending on the architecture it is either a one-dimensional or two-dimensional convolutional layer. The succeeding number in parentheses refers to the number of kernels/filters within the convolutional layer.
Pool	Refers to a pooling layer. The succeeding number in parentheses refers to the size of the pooling kernel, i.e., how large of a neighbourhood is pooled. Furthermore, it also refers to the stride of the pooling kernel.
DP	Refers to Dropout. Additional information regarding the specific Dropout values used after each layer can be found in Appendix C.
Output	Refers to the output layer of the model. The number of neurons in the output layer must be equal to the number of different categories. Consequently, the number of neurons must be equal to 21, since the data can be classified as normal or as one of 20 different types of faults.

It is important to note that for both the DNN and CNN 1D models the cost matrix had to be flattened into a vector before being fed as input to the model, while the shape of the cost matrix for the CNN 2D model (, i.e. a two-dimensional 15×15 matrix), remained unchanged. During hyperparameter tuning it was discovered that the optimal kernel size for the CNN 1D model was 11, and for the CNN 2D model it was 7. Furthermore, the stride for the CNN 1D and CNN 2D convolutional kernels was 1. Note that a kernel size of 7 for the CNN 2D model refers to a kernel with dimensions 7×7 . Furthermore, it is important to note that the pooling layer used in the CNN 1D model is a max pooling layer and was found to perform better than an average pooling layer during hyperparameter tuning.

Table 4.2: Optimised architectures of traditional non-graphical ML techniques for the cost matrix data.

Model	Architecture
DNN	Input - FC(425) - DP - FC(375)-DP-FC(300) - DP - FC(225)-DP - Output (21)
CNN 1D	Input - Conv(32) - Conv(64) - Pool(2) - Flatten - FC(225) - DP - FC(225) - DP - Output (21)
CNN 2D	Input - Conv(64) - Conv(128) - Flatten - FC(175) - DP - Output (21)

The additional optimal hyperparameters for the different ML models obtained with hyperparameter tuning, not mentioned in this subsection, can be found in Appendix C.

4.6 Results

The fault detection and fault classification rates of the traditional non-graphical ML techniques using the cost matrix data, can be found in Tables 4.3 and 4.4 respectively. Note that these rates are expressed as percentages, with a 100 implying that the technique was able to detect or classify the fault accurately.

Table 4.3: Fault detection rates (%) for traditional non-graphical ML techniques using the cost matrix data.

Fault ID	DNN	CNN 1D	CNN 2D
1	100	100	100
2	98	98.5	98.38
3	100	100	100
4	100	100	100
5	100	100	100
6	100	100	100
7	42.81	43	47.13
8	95.94	95.69	96.31
9	92.25	96.25	96.88
10	87	86.88	88
11	97.5	97.5	98.13
12	99.56	99.5	99.44
13	98	97.56	97.31
14	46.88	77.5	83
15	9.06	15.94	23.06
16	24.5	31.25	42
17	90.19	90.81	91.5
18	79.06	78.54	79.69
19	8.88	10.19	15.25
20	60.69	63.43	70.13
Average	76.52	79.13	81.31

Table 4.4: Fault classification rates (%) for traditional non-graphical ML techniques using the cost matrix data.

Fault ID	DNN	CNN 1D	CNN 2D
0	96.04	96.69	94.03
1	99.69	99.75	99.38
2	97.88	98.5	98.13
3	100	100	100
4	99.88	100	100
5	99.81	99.94	99.19
6	99.58	100	98.75
7	35.75	35.75	33.06
8	82.44	80.19	83
9	84.13	89.13	91.81
10	82.56	81.56	81.88
11	80.88	91.38	92.69
12	93.56	93.44	93.38
13	86.81	90	88
14	42.06	74.75	78.81
15	2.44	6.94	11.19
16	14.50	20.31	31.75
17	82.56	86.81	88.94
18	68.44	64.79	62.08
19	0	0.56	1.19
20	50.88	55.63	60.50
Average	71.42	74.58	75.61

4.7 Discussion

Fault detection:

Initially, when comparing the average fault detection rates of the different techniques (DNN, CNN 1D, and CNN 2D) with their corresponding average fault classification rates, it was found that the average fault detection rates were higher than the average fault classification rates. However, this was expected, since fault detection is a much simpler task compared to fault classification. Fault classification is a multiclass classification problem which is considerably more complex than fault detection which in turn is a simpler binary classification problem.

The different techniques had high detection rates for most of the different step faults (faults

1 - 7, readers are referred to Table 2.1 for the different behaviours of each fault); except for fault 7 where all the techniques had low detection rates. The output of the self-organising map for fault 7 indicated the presence of distinct clusters within the data (see Table 3.5). However, the output of the PCA plot for fault 7 indicated that the “normal” samples were mixed with “faulty samples” inside one of the clusters. Furthermore, there was no clear separation between the “normal” and “faulty” samples in the PCA plot for fault 7 (see Table 3.1). Therefore, the fault detection rate for fault 7 appeared to be poor.

Faults that exhibit random variations (faults 8 - 12) were easily detectable with the different techniques, according to the relatively high detection rates. Among the random variation faults, fault 10 had the lowest fault detection rates across the different techniques. A possible reason for the lower fault detection rates for fault 10, might be found in the resulting distance matrices for the random variation faults. Although fault 10 indicated the presence of step changes (see Table 3.3), its step changes were small compared to the other random variation faults that had step changes. Furthermore, it appeared that the distance matrix for fault 10 was almost the same as the distance matrix of the system under normal operating conditions (Figure 3.6) and therefore the lower detection rate of fault 10 compared to the other random variation faults.

Fault 13, the only fault of a slow drift nature, could be easily detected with the different techniques. Faults that exhibited a sticking nature (faults 14 and 15) had mixed detection results. Both CNNs could detect fault 14 to a relatively high degree, while the DNN had a low detection rate for fault 14. All three techniques struggled to detect fault 15 as indicated by their low detection rates; however, as mentioned in Section 2.6.2, fault 15 is hard to detect and classify according to the literature. Furthermore, the PCA plot for fault 15 indicated that there was no distinct separation between ‘normal’ and ‘faulty’ samples (see Table 3.1), making it difficult to detect fault 15, as mentioned in Section 2.6.2. The conclusion reached above was further substantiated by the output of the self-organising map and its corresponding category in Table 3.5.

Faults 16 – 20, exhibiting unknown behaviour, had mixed detection results, with some faults being easily detectable and others difficult to detect. Fault 19 had the lowest detection rate among the different techniques. Unlike faults 3, 9, and 15 (see Section 2.6.2), fault 19 has not been reported to be difficult to detect and classify. Therefore, the poor fault detection performance of fault 19 might be attributed to the Simulink TEP model used to generate the exergy and energy data. Furthermore, the PCA plot for fault 19 indicated that there was no clear grouping or separation between the “normal” and “faulty” samples (see Table 3.1). In addition, the distance matrix and the output of the self-organising map for fault 19 also showed no discernible differences between the system being under normal operating conditions and the presence of fault 19 (see Tables 3.3) and 3.5.

Comparing the average detection rates, it was evident that the two-dimensional CNN had the highest detection rate amongst the different techniques, whilst the DNN had the lowest detection rate. Consequently, it appeared that the useful entries inside the cost matrix, used for fault detection, were not distributed globally or sparsely. Instead, it appeared that the entries were more locally grouped since the one-dimensional CNN had a higher fault detection rate than the DNN. Since the two-dimensional CNN had the highest detection rate, it was evident that locally grouped entries did not only span a single row or column, but instead span across multiple rows and columns.

Fault classification:

Similarly to the fault detection results, the fault classification results (Table 4.4) for faults with a step-like nature (faults 1 - 7), were relatively high among the different techniques, except for fault 7. Although the output of the self-organising map for fault 7 indicated the presence of several clusters within the data (see Table 3.5), the PCA plot for fault 7 showed that the ‘normal’ and ‘faulty’ samples were mixed within one of the clusters with no distinct separation (see Table 3.1). Therefore, the fault classification rate for fault 7 was low as the ‘faulty’ samples could not be easily discerned from the ‘normal’ samples.

Faults whose behaviour caused random variations (faults 8 - 12) had relatively high fault classification rates among the different techniques. Interestingly, for faults 10 and 12, the DNN had higher fault classification rates compared to the two CNNs. Consequently, it might be that the useful entries inside of the cost matrix, used to correctly classify faults 10 and 12, were not tightly grouped but instead are more outspread.

For fault 13, all three techniques had relatively high fault classification rates. Faults 14 and 15, which cause sticking within the system, had mixed fault classification rates. As seen in Table 4.4, fault 15 was significantly more difficult to classify compared to fault 14. However, as mentioned in Section 2.6.2, fault 15 is a difficult fault to detect and classify.

The fault classification rates were mixed for faults that cause unknown changes within the system (faults 16 - 20), with some faults being relatively easy to classify, while other faults were difficult to classify. Similar to the fault detection scenario, the fault classification rate for fault 19 was the lowest among the different techniques.

The average fault classification rates indicated that the two-dimensional CNN had a significantly better fault classification rate compared to the DNN, and a marginally better classification rate compared to the one-dimensional CNN, resulting in the best overall fault classification rate. Similar to the fault detection scenario, it appeared that the useful entries inside the cost matrix for fault classification were locally grouped and span multiple rows and columns. Unlike the fault detection scenario, the difference in overall fault classification capability between the one-dimensional and two-dimensional CNN was not

that large.

4.8 Conclusion

In this section, traditional non-graphical types of ML techniques, i.e. a DNN, a 1D CNN, and a 2D CNN, were applied to the cost matrix data for the purposes of fault detection and classification. For fault detection and fault classification, the 2D CNN had the best overall performance, while the DNN had the lowest overall performance compared to the other techniques. Consequently, since the 2D CNN had the best overall performance, it can be assumed that the useful entries inside the cost matrix, which are used for fault detection and fault classification, are grouped locally and span multiple rows and columns instead of being sparsely located. In the next chapter, a similar process is followed for the node signature matrix data, but with graphical ML techniques that can work with graph data structures.

Chapter 5

Machine learning for the node signature matrix data

5.1 Introduction

This chapter provides the results obtained with graph-compatible ML techniques on the node signature matrix data. Initially, a brief overview of the different graph compatible ML techniques used in this chapter is provided. In addition, the fault detection and classification rates for the different graph compatible ML techniques are provided and discussed. Finally, the best performing graph compatible ML technique for the node signature matrix data is identified and will be used for comparison in later chapters.

5.2 Graph neural networks

Graph neural networks (GNNs) refer to a family of ML techniques that can process data which are represented as graphs. In addition, GNNs accept graphs as input, process them, and are able to perform a variety of prediction tasks. These prediction tasks include graph-level, node-level, and edge-level prediction tasks [188]. With graph-level prediction tasks, a property of the entire graph has to be predicted, whereas, for node- and edge-level prediction tasks, the properties of the various nodes and edges within the graph have to be predicted. It is important to note that for this current study fault detection and classification were considered as graph-level prediction tasks.

Most GNNs use a mechanism, called message passing, to allow nodes to share information with their neighbouring nodes and update their representations. During message passing, the neighbouring nodes of a specific node are identified, and their representations are combined with that of the specific node, followed by a function that is applied to the combined

representations to create a new representation for the specific node [189]. Furthermore, this message passing mechanism is applied in parallel to each node, to allow each node to update their representation based on their current representation and the representations of their neighbouring nodes.

The umbrella term, GNNs, can be divided into four main groups, namely recurrent GNNs, convolutional GNNs, graph autoencoders and spatial temporal GNNs [190]. Figure 5.1 illustrates the classification scheme used to classify the different types of GNNs. Recurrent GNNs solely use the basic message passing mechanism and treat neighbouring nodes equally during each iteration, while convolutional GNNs can assign different weights to neighbouring nodes based on their importance. Graph autoencoders are used to learn low-dimensional latent representations of graphs, whereas spatial temporal GNNs have the added complexity of modelling node attributes as time series data, thereby allowing them to learn both the structural and temporal relationships between the different nodes.

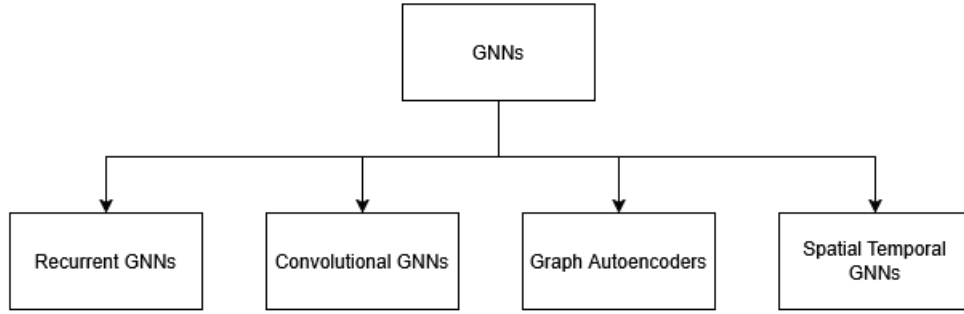


Figure 5.1: Classification scheme for GNNs.

In this current study, convolutional GNNs were selected to be applied to the node signature matrix data. Convolutional GNNs offer additional benefits over other types of GNNs, such as the ability to extract local features from neighbouring nodes and to extract hierarchical features with subsequent convolutional layers [191]. Further the current study will use the node signature matrix data on a sample-by-sample basis instead of a time series and no low-dimensional graph embedding is required. Consequently, graph autoencoders and spatial temporal GNNs were not considered.

Depending on how their convolution operator is applied, graph CNNs can be divided into either spectral or spatial methods [192]. Spectral methods are based on concepts within the field of graph signal theory and are based on the structure of the graph, while spatial methods are based on the convolution operation that CNNs typically perform on grid-like data [193]. In addition, spectral convolutions on graphs, within the field of graph signal

theory, are based on defining functions that use the eigenvalues of the Laplacian matrix of the graph [194].

Spectral methods have the advantage of being able to capture more global patterns, compared to spatial methods that focus more on local patterns [195]. However, spatial methods can generalise better to graphs with different structures, since they focus on local neighbourhoods of nodes instead of the entire graph structure, whereas spectral methods require graphs to have the same structure [196].

Figure 5.2 represents the typical structure for a graph CNN [197]. Initially, a graph is provided as input, after which several graph convolutional layers (represented as the message passing phase in Figure 5.2) are applied to the graph. Within the convolutional layers, the attributes/representations of the various nodes are updated using information from their neighbouring nodes. After applying the graph convolutional layers, the resulting graph is converted to a format needed to access the features of the graph to make predictions. This is illustrated as the readout phase in Figure 5.2. During the readout phase, a pooling function is required to summarise the graph (represented as the column vector between the pooling and multilayer perceptron stages in Figure 5.2); for example, the mean pooling function can be used to find the mean of the attributes/representations of the different nodes. After the graph has been summarised, it is used as input to a multilayered perceptron, which is responsible for learning a mapping between the summarised graph features and the desired output. Note that a multilayered perceptron is analogous to the fully connected layers described in Section 4.3.2.

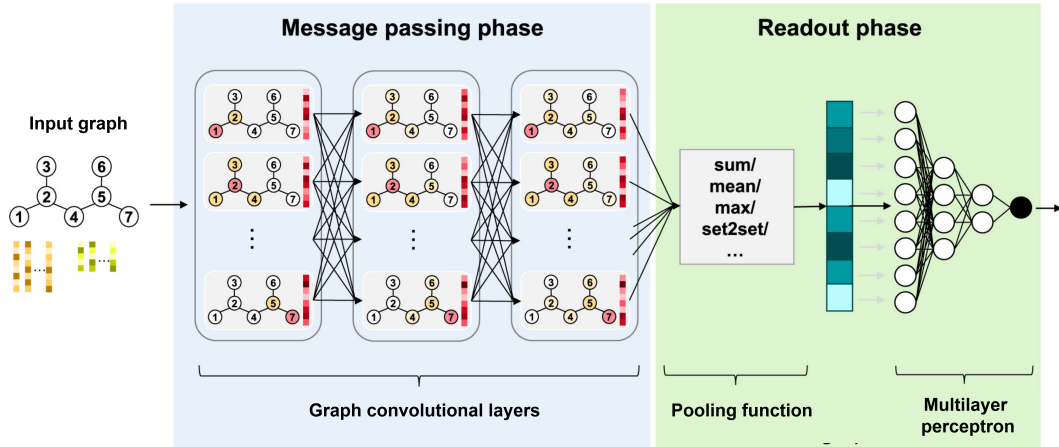


Figure 5.2: Typical structure of a graph CNN (Adapted from [197]).

In the following subsections, three different types of graph convolutional layers will be briefly discussed. These include GCN, ChebConv, and GATConv, which can be found in the PyTorch Geometric documentation [198]. The three different types of graph convolutional layers mentioned will be used in the current study to implement different convolutional GNNs.

5.2.1 GCN

The convolution operation performed within a GCN layer is a spectral method and was proposed by Kipf and Welling [199]. Further, the convolution operation is a first-order Chebyshev polynomial approximation of spectral graph convolution. The convolution operation is only capable of accessing the information stored within the one-hop neighbours of nodes [200]. Consequently, when the attributes of a node are updated, only the attributes of the immediate neighbours, which are one step away, can be used. For a more in-depth discussion regarding the convolution operation, readers are referred to the original paper by Kips and Welling [199]. Note that the convolutional GNN that was implemented with the GCN layer will be referred to as GCN-Net.

5.2.2 ChebConv

Defferrard *et al.* proposed the convolution operation that takes place within a ChebConv layer [201]. In contrast to the GCN layer, which is only a first-order Chebyshev polynomial approximation, the ChebConv layer can approximate spectral graph convolution using a K-th order Chebyshev polynomial. Furthermore, adjusting the order of the Chebyshev polynomial allows the ChebConv layer to consider K-hop neighbourhoods, implying that long-range dependencies between nodes can be learnt contrary to the GCN layer [202]. A more in-depth discussion of the ChebConv layer and its convolution operation can be found in the original paper by Defferrard *et al.* [201]. Note that the convolutional GNN that uses the ChebConv layer will be referred to as ChebConv-Net.

5.2.3 GATConv

The GATConv layer was proposed by Veličković *et al.* and, unlike the GCN and ChebConv layers, it is a spatial method [203]. Similar to the GCN layer, the GATConv layer only considers immediate neighbouring nodes, i.e. nodes that are one hop away. Unlike the other two types of convolutional layer, the GATConv layer uses an attention mechanism that allows it to assign different importances to neighbouring nodes.

Attention mechanisms allow neural networks to determine which information is the most relevant and give it more importance, in contrast to less relevant information, which is given less importance [204]. Moreover, the GATConv layer is designed to employ a multi-head attention mechanism, where each attention mechanism learns a different function between neighbouring nodes, resulting in each attention mechanism assigning different importance values between each set of neighbouring nodes. Veličković *et al.* also recommend that the different node features learnt with the different attention mechanisms should be concatenated in the initial GATConv layers, while being averaged in the last

GATConv layer [203]. A more in-depth discussion of the inner workings of the GATConv layer can be found in the original paper by Veličković *et al.* [203]. Note that the convolutional GNN that is implemented with the GATConv layer will be referred to as GATConv-Net.

5.3 Training, validation and testing datasets description

The training dataset comprised 20 simulation runs for each fault condition as well as for normal operating conditions. Each simulation run represented 48 hours of plant operation, sampled at 3-minute intervals, with the respective fault condition being introduced after 8 hours of simulated time. Furthermore, 5% of the complete training dataset was reserved as a validation set for hyperparameter tuning. For both the training and validation datasets, stratification was used to ensure an equal representation of all operating conditions, thereby reducing the risk of the models developing a bias toward any specific condition during training.

The test dataset was constructed using an equal distribution of the various operating conditions and consisted of 2 simulation runs per condition, with each run again corresponding to 48 hours of simulated time sampled at 3-minute intervals. As in the training phase, the faults were introduced after 8 hours of simulation in the test runs. Distinct random seeds were used for all simulations in both the training and testing phases to ensure statistical independence between runs.

5.4 Architectures of graphical machine learning models

Similar to the approach mentioned in Section 4.5, with the cost matrix data, hyperparameter tuning was also used to determine the optimal architectures for the graphical ML models with the node signature matrix data. The symbols used to describe the different layers within the architectures are explained in Table 5.1. Furthermore, the optimal architectures of the graphical ML techniques are summarised in Table 5.2. Additional aspects of the hyperparameter tuning results, which are not mentioned in Table 5.2 or this section, can be found in Appendix D.

Table 5.1: Symbol legend for different types of graphical layers.

Symbol	Description
Input	Refers to the input layer of the model.
GCNConv	Refers to a graph convolutional layer that implements the graph convolutional operation mentioned in Section 5.2.1. Note for GCNConv and all other subsequent graph convolution layers (ChebConv and GATConv), that the succeeding number in parentheses refers to the output dimensionality of the node representations.
ChebConv	Refers to a graph convolutional layer that implements the graph convolutional operation mentioned in Section 5.2.2.
GATConv	Refers to a graph convolutional layer that implements the graph convolutional operation mentioned in Section 5.2.3. Note that if the number of attention heads within the GATConv layer is greater than 1, then the output dimensionality of the node representations is timed with the number of attention heads since node representations are concatenated within all GATConv layers except the last one (see Section 5.2.3).
DP	Refers to Dropout (see Section 4.3.1). Additional information regarding the Dropout values used in the different graphical ML models can be found in Appendix D.
FC	Refers to a fully connected layer (see Section 4.3.2). The succeeding number in parentheses refers to the number of neurons within the fully connected layer.
Output	Refers to the output layer of the model. The number of neurons in the output layer must be equal to the number of different categories. Consequently, the number of neurons must be equal to 21 , since the data can be classified as normal or as one of 20 different types of faults.

All three architectures received the same input, namely the node attributes packed into a matrix and the adjacency matrix of the graph. The matrix containing the node attributes, consisted of the exergy and energy attributes for each node. Furthermore, the dimensionality of the node attribute matrix was 15×17 , with each node having 17 attributes (see Section 2.6.3). Therefore, each node representation had an initial dimensionality of 17. Note that the adjacency matrix of the graph was used to approximate spectral graph convolutions with the spectral approaches or to identify neighbouring nodes with the spatial approaches.

It is important to note that for the ChebConv-Net an optimal order of 3 was found for the Chebyshev polynomial using hyperparameter tuning. For the GATConv-Net the optimal number of attention heads was 2 using hyperparameter tuning. Furthermore, it was discovered that all three architectures, using hyperparameter tuning, showed that the optimal pooling function is the max pooling function, that is, a pooling function that returns the node representation with the largest value.

Table 5.2: Optimised architectures of graphical ML techniques for the node signature matrix data.

Model	Architecture
GCN-Net	Input - GCNConv(128) - GCNConv(256) - GCNConv(512) Maxpool - FC(500) -DP - Output(21)
ChebConv-Net	Input - ChebConv(128) - ChebConv(256) - ChebConv(512) Maxpool - FC(500) -DP - FC(200) - DP - Output(21)
GATConv-Net	Input - GATConv(128) - GATConv(512) - Maxpool - FC(500) - DP FC(425) - DP - Output(21)

The additional optimal hyperparameters for the different graphical ML models obtained with hyperparameter tuning, not mentioned in this subsection, can be found in Appendix D.

5.5 Results

Tables 5.3 and 5.4 represent the respective fault detection and fault classification rates for the graphical ML techniques using the data of the node signature matrix. Note that these rates are expressed as percentages, with 100 implying that the technique was able to detect or classify the fault accurately.

Table 5.3: Fault detection rates (%) for graphical ML models using the node signature matrix data.

Fault ID	GCN-Net	ChebConv-Net	GATConv-Net
1	100	100	100
2	99.38	99.06	99.44
3	100	100	100
4	100	100	100
5	100	100	100
6	100	100	100
7	45.63	39.38	42.81
8	95.69	95.31	95.13
9	98	97.5	97.19
10	92.88	92.69	92.31
11	99.06	98.63	98.88
12	99.5	99.38	99.5
13	97.63	97.56	97.56
14	84.88	83.5	82.81
15	10.31	16.75	13.94
16	44.75	44.25	46.38
17	92.19	92.13	92.06
18	78.85	78.65	77.71
19	16.19	12.44	11.75
20	64.69	71.13	64.75
Average	80.98	80.92	80.61

Table 5.4: Fault classification rates (%) for graphical ML models using the node signature matrix data.

Fault ID	GCN-Net	ChebConv-Net	GATConv-Net
0	95.59	97.54	97.06
1	99.25	99.63	99.50
2	98.56	98.5	99.06
3	100	100	100
4	100	100	100
5	99.81	99.81	99.69
6	99.17	100	100
7	33.63	34	34.75
8	84.69	83.81	83.81
9	95.25	94.75	94.50
10	90.06	88	88.44
11	96.25	95.88	94.56
12	96.38	93	96
13	90.63	90.68	90.88
14	80	80.81	78.19
15	2.06	9.5	8.13
16	37.63	39.38	40.69
17	90.25	91.75	90.88
18	72.40	75.52	68.85
19	4.25	2.44	2.31
20	53.75	63.56	55.44
Average	77.12	78.03	77.27

5.6 Discussion

Fault detection:

Similar to the results for the cost matrix data in Section 4.6, the average fault detection rates were higher than the average fault classification rates for the node signature matrix data using the different graphical techniques (GCN-Net, ChebConv-Net, and GATConv-Net). As mentioned in Section 4.7, this was expected, since fault detection is an inherently easier classification problem compared to fault classification. Fault detection can be viewed as a binary classification problem (indicating whether a fault is present or not), while fault classification can be viewed as a multiclass classification problem (indicating what type of fault is present).

Comparable to the results in Section 4.6, the graphical ML techniques had high detection rates for most of the faults with a step-like nature (faults 1 - 7), except for fault 7 (readers are referred to Table 2.1 for the different behaviours of the faults). Although the distance matrix for fault 7 indicated that fault 7 introduced some decaying oscillations into the system (see Table 3.4), it appeared that these oscillations were not large enough to adequately separate “faulty” samples from “normal” samples. Instead, the PCA plot for fault 7 showed that the “faulty” samples were tightly mixed with the “normal” samples, with no clear separation between them (see Table 3.2), and thus a poor fault detection rate.

The different graphical ML techniques had high and comparable fault detection rates for faults that caused random variations within the system (faults 8 - 12). Like the cost matrix results in Section 4.6, the fault detection rates for fault 10 were the lowest among the different graphical techniques. However, compared to the fault detection rates of fault 10 for the traditional non-graphical techniques in Section 4.6, the fault detection rates of the graphical techniques were significantly higher for fault 10. A possible reason for the higher detection rates for fault 10 using the node signature matrix data compared to using the cost matrix data might be in their respective PCA plots. Although both PCA plots indicated limited separation (see Tables 3.1 and 3.2), the PCA plot using the node signature matrix data indicated that the “normal” samples were more tightly grouped and more separated from the “faulty” samples, compared to the PCA plot for the cost matrix data. This greater degree of separation between the “normal” and “faulty” samples led to a higher fault detection rate.

Fault 13, causing a slow drift in the system, could be detected relatively accurately by the three techniques that showed comparable fault detection rates. Faults 14 and 15 (faults with a sticking nature) had inconsistent fault detection rates. Fault 14 had relatively high fault detection rates among the different techniques, while the techniques struggled to detect fault 15 and resulted in low fault detection rates. However, as mentioned in Section 2.6.2, fault 15 is a difficult fault to detect and classify. Furthermore, the distance matrix for fault 15 indicated no significant changes in the system variables (see Table 3.4), and was similar to the distance matrix when the system is operating normal without a fault present (Figure 3.7). Thus, it would be difficult to detect the presence of fault 15.

Faults exhibiting unknown behaviour (faults 16 - 20) also showed mixed fault detection rates. Like the results in Table 4.3, the graphical ML techniques also struggled to detect fault 19. Furthermore, the distance matrix for fault 19 indicated that fault 19 did not significantly change the system variables (see Table 3.4). Additionally, the PCA plot for fault 19 did not show a distinct separation between the “normal” and “faulty” samples (see Table 3.2). These randomly mixed samples indicated the difficulty to detect the presence of fault 19 within the system.

When comparing the average fault detection rates for the different graphical techniques,

no technique offered a significantly higher fault detection rate compared to the other graphical techniques. Furthermore, the graphical technique with the highest average fault detection rate, the GCN-Net, had a lower average fault detection rate compared to the traditional non-graphical technique with the highest average fault detection rate, the CNN 2D. However, comparing the other graphical techniques (ChebConv-Net and GATConv-Net) against the other traditional non-graphical techniques (DNN and CNN 1D), it was evident that overall the graphical techniques had higher fault detection rates compared to the traditional non-graphical techniques.

Fault classification:

The fault classification results (Table 5.4) showed a similar trend compared to the fault detection results (Table 5.3) for faults exhibiting a step-like nature (faults 1 - 7). Relatively high fault classification results were obtained for most of the faults across the different graphical techniques, except for fault 7. As mentioned previously, in the fault detection case, the PCA plot for fault 7 showed no clear separation between the “normal” and “faulty” samples (see Table 3.2), indicating a low fault classification rate since it is difficult to differentiate between “faulty” samples and “normal” samples when the system is operating normally.

For faults 8 – 12, causing random variations within the system, the fault classification rates were relatively high and comparable between the different graphical techniques. Unlike the fault detection results, the fault with the lowest fault classification rate, among the different graphical techniques, was fault 8. A possible reason for fault 8 having the lowest fault classification rate might be by comparing the results of the self-organising maps for faults 8 – 12. The output of the self-organising map for fault 8 indicated the presence of multiple small clusters (see Table 3.6). The presence of multiple small clusters in the output of the self-organising map for fault 8 makes it possible for the “faulty” samples to be mixed with the “normal” samples in one of the clusters. Furthermore, one of these several clusters could possibly contain a mixture of “faulty” and “normal” samples, and therefore no clear distinct separation. This was further supported by the PCA plot for fault 8 which contained limited separability (see Table 3.2), where the “faulty” and “normal” samples formed a cluster in the centre, indicating the poor fault classification rate for fault 8 compared to the rest of the random variation faults.

All graphical techniques could classify fault 13 with relatively high and comparable accuracies. Like the fault detection results, the fault classification results for faults 14 and 15 were inconsistent, with relatively high classification rates for fault 14, but extremely low classification rates for fault 15. Furthermore, comparing the fault classification rates of the graphical techniques against the traditional non-graphical techniques (Table 4.4) for fault 15, it was apparent that the traditional non-graphical techniques in general had

higher fault classification rates compared to the graphical techniques for fault 15.

For faults 16 – 20, the classification rates varied, where some faults had relatively high rates, while others, like fault 19, exhibited very low rates across the different graphical techniques. As mentioned previously, with fault detection, fault 19 did not induce noticeable changes within the system. Therefore, “faulty” samples were similar to “normal” samples and led to poor fault classification results.

Comparing the average fault classification rates of all the graphical techniques, ChebConv-Net offered the highest fault classification rate amongst the graphical techniques, followed by GATConv-Net having the second highest average fault classification rate, just slightly higher than the average fault classification rate of GCN-Net. Since ChebConv-Net had the best average fault classification rate and the average fault classification rates of GATConv-Net and GCN-Net being comparable, it was concluded that the spectral methods overall performed better than the spatial method.

Comparing the fault classification rate of ChebConv-Net with the fault classification rate of the best traditional non-graphical technique, CNN 2D, it was found that ChebConv-Net offered better fault classification. In addition, the overall average fault classification rates of the graphical techniques were higher compared to the average fault classification rates of the traditional non-graphical techniques, thereby indicating that the graphical techniques were able to exploit the topological information of the graph structure for more accurate fault classification [205].

5.7 Conclusion

In this section, different types of convolutional GNNs, i.e. GCN-Net, ChebConv-Net and GATConv-Net, were applied to the node signature matrix data for the purpose of fault detection and fault classification. For fault detection, it was discovered that all three types of convolutional GNN had similar overall fault detection rates. Further, it was shown that the best graphical technique for fault detection, GCN-Net, performed slightly worse than the best traditional non-graphical technique, CNN 2D. For fault classification, the overall fault classification rates for the graphical techniques were higher compared to the overall fault classification rates for the traditional non-graphical techniques. Consequently, it appears that the topological information contained within the graph, represented by the node signature matrix, allowed for more accurate fault classification compared to the traditional non-graphical approaches that used the cost matrix.

Chapter 6

Comparative study

6.1 Introduction

In this chapter, traditional benchmark methods for fault detection and fault classification will be investigated and briefly discussed. Further, the standard EGBV technique, that uses the singular value decomposition (SVD) of the cost matrix (see Section 2.5.3), will also be used as a benchmark and will be briefly discussed. The performances of the traditional benchmark methods and the standard EGBV technique will be compared with the results obtained in Chapters 4 and 5. The traditional benchmark methods will use the raw measurements from the Tennessee Eastman Process (TEP), while the standard EGBV technique will use the cost matrix data. Comparison of the results of the benchmark methods with the results obtained in Chapters 4 and 5 will give an indication whether applying ML techniques to the exergy and energy measurements of the TEP offers advantages over the traditional benchmark methods, that use the raw measurements, and the standard EGBV technique, not using any ML.

6.2 Traditional fault detection and classification

This section briefly describes the traditional benchmark techniques used for fault detection and classification. Furthermore, it is indicated whether a technique was used for fault detection, fault classification, or both. In addition to brief descriptions, implementation details of how the authors implemented the techniques are also provided. Note that the subsequent detection and classification techniques were implemented using the raw measurements generated by the Simulink Tennessee Eastman (TE) model being used in this current study (see Section 2.6.3).

The benchmark fault detection techniques used include multivariate statistical techniques

such as PCA (Principal Component Analysis), ICA (Independent Component Analysis), and PLS (Partial Least Squares), as well as a ML technique called one-class SVM (Support Vector Machine). Multivariate statistical techniques, that is, PCA, ICA and PLS, were implemented according to the paper by Yin *et al.* [99]. Furthermore, the one-class SVM algorithm was implemented according to the article by Mahadevan and Shah [206].

PCA, SVM, and an ANN (Artificial Neural Network) were used as benchmark techniques for fault classification. PCA is a statistical technique, while the SVM and ANN are ML techniques. The PCA and SVM fault classification techniques were implemented using the article by Jing and Hou [207]. Furthermore, the ANN technique was implemented using the article by Nashalji *et al.* [208].

6.2.1 PCA

Fault detection

PCA is used for statistical process monitoring, by projecting measurement variables onto a lower dimensional space and calculating statistics for the lower-dimensional projected data [209]. The statistics used in PCA for statistical process monitoring are the T^2 and Q statistics. If the T^2 and Q statistics exceed a predefined threshold, then there is a fault within the system; otherwise, the system is fault-free. Additional information on how the T^2 and Q statistics are calculated, as well as their respective thresholds, can be found in [99].

For their PCA fault detection implementation, Yin *et al.* only used 33 of the 52 variables from the raw TEP measurements [99]. Furthermore, these 33 variables comprised 22 measured variables and 11 manipulated variables (see Section 2.6.1 for the different types of variables within the TEP). Lastly Yin *et al.* used 9 principal components and a confidence interval of 95% when they calculated the thresholds for the T^2 and Q statistics.

Fault classification

The PCA fault classification technique, suggested by Jing and Hou, requires the calculation of loading vectors for a matrix that contains data from the different types of classes [207]. Furthermore, the average and covariance matrix for the data must be calculated for the different types of classes. A discriminant function is used, where the loading vectors, class average, and class covariance matrix are used to classify the class a data sample belongs to. Moreover, a discriminant value is calculated, with the discriminant function, for each class, by using the respective average and covariance matrix of the respective class. Finally, the class with the highest discriminant score is the class to which the data sample belongs.

Unlike the PCA fault detection technique, the PCA fault classification technique uses all 52 raw measurements from the TEP. Furthermore, Jing and Hou also use all the loading vectors, unlike the PCA fault detection technique, where only 9 loading vectors were used, for the PCA fault classification technique [207].

6.2.2 ICA

Fault detection

Like PCA, ICA is also a dimensionality reduction technique that can be used to project measurement variables onto lower dimensions. However, while PCA focusses on representing the data in terms of its variance (see Section 3.3), ICA focusses on representing the data in terms of statistically independent components [210]. The ICA process monitoring technique, as implemented by Yin *et al.*, calculates three statistics for the projected lower-dimensional data, namely I^2 , I_e^2 , and the Q statistic [99]. If one of the above-mentioned statistics exceeds a threshold, then a fault is present within the system. Information regarding the calculation of the statistics and their corresponding thresholds can be found in [99].

Yin *et al.* used 9 independent components as well as a confidence interval of 95% to determine the thresholds for the respective statistics [99]. Similar to the PCA technique, the ICA technique, of Yin *et al.*, also used only 33 of the 52 variables of the raw TEP measurements.

6.2.3 PLS

Fault detection

PLS is a projection technique that requires input variables and response variables to build a model. The aim of the model is to find the projected input variables that best describe the projected response variables [211]. In other words, a model is constructed that tries to learn how the projected input variables relate to the projected response variables. Similar to the other statistical techniques, PLS uses statistics, namely the T^2 and Q statistics, of the projected data and thresholds to detect if a fault is present in the system. Additional information on the calculation of the statistics and their corresponding thresholds can be found in [99].

The input variables used for the PLS model are the same 33 variables used for the PCA and ICA approaches, while the response/output variable is chosen to be the 35th measured variable (pure gas analysis for component G, see Appendix A) for the implementation by

Yin *et al.* [99]. Furthermore, the dimensionality of the projected data (latent variables) was selected to be 6 and a confidence interval of 95% was chosen for the thresholds.

6.2.4 One-class SVM

Fault detection

A one-class SVM (Support Vector Machine) is a ML algorithm that is used for outlier detection, meaning that it is fed training data from a single class and must be able to distinguish it from other classes [212]. The training data of a single class is projected onto a high-dimensional space using a kernel function, and the one-class SVM learns a boundary function, which contains most of the training data. When new unseen data is fed into the one-class SVM, it projects it onto the high-dimensional space and then checks whether the data sample lies within the boundary. If the data sample lies within the boundary, it is classified as being “normal” or belonging to the class of the training data. However, if it falls outside the boundary, it is classified as an outlier and belongs to another class.

Mahadevan and Shah used a radial basis kernel function with their one-class SVM implementation [206]. They also used all the 52 raw measured variables from the TEP when training the one-class SVM. Note that the authors trained the one-class SVM with ‘normal’ data when the system was operating under normal conditions and no faults were present.

6.2.5 SVM

Fault classification

An SVM is a supervised ML technique that attempts to learn a hyperplane that separates the data of two classes [213]. Furthermore, an SVM can use kernel functions to project non-linear data onto a higher dimensional space, thereby achieving linear separability of the projected data. Since a SVM is mainly used for binary classification (two classes), several techniques have been developed to adapt it for multi-class classification. Two of the most common approaches to adapt an SVM for multi-class classification, are the one-versus-rest and one-versus-one approaches [207]. With the one-versus-rest approach, if there are n categories, n SVMs are constructed where each category represents one class, while the other combined categories represent the other class. In contrast, for the one-versus-one approach, if there are n categories, then an SVM is constructed between each pair of possible categories, resulting in $n(n-1)/2$ different SVMs.

For their SVM implementation, Jing and Hou used the one-versus-one approach for multi-class classification [207]. They used all 52 raw measurements from the TEP, and used PCA to reduce the dimensionality of the data before constructing the SVMs. Moreover, the authors used a radial basis kernel function for their SVMs. A more detailed discussion of their implementation can be found in [207].

6.2.6 ANN

Fault classification

A discussion regarding how ANNs (Artificial Neural Networks) work can be found in Section 4.3.1. Essentially, an ANN consists of several adjustable non-linear processing layers. The aim of an ANN is to adjust these non-linear processing layers to find a mapping between input data and a desired output. Furthermore, an ANN adjusts these processing layers by trying to minimise the difference between the desired output and the predictions of the ANN.

The ANN implementation by Nashalji *et al.* used the hyperbolic tangent function as an activation function [208]. Furthermore, they used the Levenberg-Marquardt learning algorithm to train the ANN. Additional information regarding the ANN architectures and implementation details can be found in [208].

6.3 EGBV SVD

The standard EGBV technique, which uses the SVD of the cost matrix, can be used for both fault detection and fault classification. For fault detection, the SVD results of the cost matrix are used in conjunction with an exponential weighted moving average (EWMA) control chart, while qualitative fault signatures are used for fault classification [214]. The following subsections provide a more in-depth discussion regarding how the SVD results of the cost matrix is used for the purpose of fault detection and fault classification.

6.3.1 Fault detection

The cost matrix data of the system, while under normal operating conditions, is required to determine thresholds for the EWMA control charts for fault detection [214]. More specifically, EWMA control charts and thresholds are used and determined for each of the resulting SVD elements, namely the singular values and the singular vectors. The upper and lower thresholds for each EWMA control chart are empirically chosen to achieve a

confidence interval of 95%, which means that 95% of the “normal” data for the specific SVD element fall within the upper and lower thresholds.

Once the different thresholds for the various SVD elements have been chosen, fault detection can take place. During the fault detection process, SVD is applied to the current cost matrix of the system and the resulting SVD elements of the current cost matrix are compared with their respective thresholds. If any of the SVD elements, which include the singular values and the entries inside of the corresponding singular vectors, exceed either one of their respective thresholds, a fault is present within the system. Additional information regarding the fault detection procedure, using the SVD of the cost matrix, can be found in [214].

6.3.2 Fault classification

The fault classification procedure, using the SVD of the cost matrix, can be viewed as an extension of the fault detection technique, mentioned in the preceding subsection. The fault classification procedure requires the thresholds that were determined for the SVD elements during the fault detection procedure. In contrast to the fault detection procedure, which only required ‘normal’ cost matrix data, the fault classification procedure requires additional cost matrix data for each type of fault.

The thresholds of the SVD elements and the cost matrix data for the different types of faults are used to create qualitative fault signatures for each type of fault. The qualitative fault signatures use three symbols, namely, +, -, and 0, and these symbols are assigned based on how the averages of the SVD elements of the ‘faulty’ cost matrices change [214]. Moreover, the fault signature is a matrix whose dimensionality corresponds to the combined dimensionality of the SVD elements, namely the singular values and their corresponding singular vectors. If the average of an SVD element is above or below its respective upper or lower thresholds, while the specific fault is present within the system, the corresponding entry inside the fault signature will be marked with a + or -, respectively. Meanwhile, if the average of the SVD element remains between its respective upper and lower thresholds, the corresponding entry inside the fault signature will be marked with a 0.

After a fault has been detected, a signature matrix is constructed for the current cost matrix. The signature matrix for the current cost matrix will be populated, similar to the process used to create the fault signatures mentioned in the preceding paragraph. Once the signature matrix is constructed, it is compared to the different fault signatures for similarity. The fault signature with the highest similarity to the signature matrix will indicate the type of fault present within the system. It is important for fault signatures to be as distinct as possible, to ensure that faults can be uniquely classified [215, 93, 90].

Additional information regarding the fault classification procedure of the EGBV SVD technique can be found in [214].

6.4 Results

The fault detection and fault classification rates of the various benchmark detection and classification techniques mentioned in Sections 6.2 and 6.3, are summarised in Tables 6.1 and 6.2, respectively. Note that the fault detection and fault classification rates are expressed as percentages, with 100 indicating that the technique was able to detect or classify the fault perfectly.

Table 6.1: Fault detection rates (%) for benchmark methods.

Fault ID	PCA	ICA	PLS	One-class SVM	EGBV SVD
1	99.88	100	99.63	100	100
2	98.25	98.25	98.50	98.50	100
3	2.88	3.75	1.88	5.75	100
4	100	100	100	100	100
5	100	100	99.88	100	100
6	100	100	100	100	100
7	100	100	100	100	96.19
8	97.63	98	97.50	98	99.94
9	1.88	3.88	1.25	3.75	98.38
10	58.25	92.50	26.75	92.63	99.06
11	99.13	99.50	96.50	99.50	100
12	98.75	98.75	98.63	98.75	100
13	94.13	95.38	94.13	95.25	99.88
14	33.13	84.88	8.88	85.13	86.13
15	6.38	10.38	3	11.88	86.88
16	36.88	96.75	10.63	96.88	92.19
17	96.50	96.63	93.75	96.50	96.69
18	86.76	87.37	85.08	87.57	93.02
19	18.13	96.88	10.38	97	86
20	60.88	97.25	45.38	97.25	94.88
Average	69.47	83.01	63.59	83.22	96.46

Table 6.2: Fault classification rates (%) for benchmark methods.

Fault ID	PCA	SVM	ANN	EGBV SVD
0	37.04	29.38	38.20	0
1	79.92	96.75	98.13	100
2	78.56	98	98	99.13
3	24.04	24	35.71	97.38
4	81.37	87.5	98.63	100
5	78.46	99.25	97	96
6	0	0	0	99.38
7	83.25	100	99.88	1.69
8	81.69	58.13	58.55	1.75
9	25.70	17.75	17.73	0
10	74.82	19.88	73.78	39.44
11	79.08	66	88.76	11.75
12	76.48	86.50	82.52	1.81
13	78.25	62.13	60.92	12.56
14	60.98	8.13	76.90	0
15	35.48	15.75	23.60	0
16	77.73	27.75	85.27	0
17	77.21	86.63	90.14	27.38
18	75.93	48.81	68.92	13.54
19	80.54	22.38	61.67	0
20	80.75	76	88.89	0
Average	65.11	53.84	68.72	33.42

6.5 Discussion

Fault detection:

Comparing the fault detection rates of the different techniques for faults 1 – 7, that is, faults with a step-like nature (readers are referred to Table 2.1 for the different behaviours of each fault), it is evident that the fault detection rates are mostly high for most faults except fault 3. Among the different techniques, only the EGBV SVD technique was able to detect fault 3 perfectly, while the other techniques had extremely poor detection rates. However, this was expected since fault 3 is notoriously hard to detect using the raw measurements, as mentioned in Section 2.6.2. Therefore, it appears that the energy and exergy information of the cost matrix data makes it possible to detect faults that are hard to detect with traditional techniques using the raw measurements.

For faults that cause random variations within the system (faults 8 - 12), most detection techniques had poor detection rates for fault 9, except the EGBV SVD technique. Similar to fault 3, fault 9 is also notoriously hard to detect using the raw measurements, as mentioned in Section 2.6.2. Furthermore, for fault 10, both the PCA and PLS techniques had low detection rates, while the other techniques had much higher detection rates.

All the detection techniques had high detection rates for fault 13, which causes a slow drift within the system. Furthermore, the EGBV SVD technique had the best detection rate among the different techniques. The detection rates for the other techniques were slightly lower and comparable with each other.

Faults that exhibit a sticking nature, faults 14 and 15, had mixed detection rates for the non-exergy and energy methods. For fault 14, the ICA and one-class SVM techniques had much higher detection rates, compared to the PCA and PLS techniques. Furthermore, all the non-exergy and energy techniques had poor detection rates for fault 15, which was expected, since it is a difficult fault to detect with the raw measurements, as mentioned in Section 2.6.2. The EGBV SVD technique had high detection rates for faults 14 and 15, but the detection rates for faults 14 and 15 were not as extremely high compared to previous faults. Moreover, the difference between the detection rates of the EGBV technique and the more accurate non-exergy and energy methods, the ICA and one-class SVM techniques, was not that large.

The PCA and PLS techniques had poorer detection rates for several unknown faults (faults 16 - 20), specifically faults 16, 19 and 20, compared to the other techniques. Furthermore, all the techniques had relatively high fault detection rates for the remaining unknown faults. Lastly, the EGBV SVD technique was outperformed by both the ICA and one-class SVM techniques for detecting faults 16, 19 and 20.

Comparing the average fault detection rates, it was evident that the EGBV SVD technique overall had a significantly higher fault detection rate compared to the rest of the techniques. Consequently, it appeared that the exergy and energy data, in the form of the cost matrix, utilised by the EGBV SVD offered more accurate fault detection compared to the non-exergy and energy methods which use raw measurements. Furthermore, the average fault detection rates for the non-exergy and energy methods, indicated that the ICA and one-class SVM techniques greatly outperformed the PCA and PLS techniques. In addition, the one-class SVM technique had a slightly higher average fault detection rate compared to the ICA technique. The ICA and one-class SVM techniques had better average fault detection rates and could be attributed due to the data possibly not following a Gaussian distribution, and that PCA and PLS techniques are not able to work effectively with non-Gaussian data [216, 217].

Fault classification:

Fault classification for “normal” samples (fault 0) seemed to be difficult for all the techniques (Table 6.2), especially the EGBV SVD technique. A possible explanation for the poor performance of the EGBV SVD technique to classify “normal” samples could be on how the fault signature matrices are constructed. Using the process outlined in Section 6.3.2, the fault signature matrix for normal conditions will contain only zeros, since all SVD elements must be within the limits. However, several incipient faults, faults that do not cause noticeable changes within the system, will also have fault signature matrices containing zeros, and thus the SVD elements will mostly stay within the thresholds and not have noticeable shifts. Consequently, the fault signature for normal conditions will not be unique, making it impossible to accurately classify it.

For faults that exhibited a step-like nature (faults 1 - 7), all non-exergy and energy techniques struggled with the classification of faults 3 and 6. The low fault classification rate for fault 3 was expected (see Section 2.6.2); however, the non-existent fault classification rate for fault 6 was surprising. The poor fault classification rate for fault 6 could possibly be attributed to the Simulink TE model used to generate the raw measurement data. Meanwhile, the EGBV SVD technique had high fault classification rates for most of the step-like faults, except for fault 7, while the non-exergy and energy methods had significantly higher fault classification rates for fault 7.

Faults 8 – 12, faults causing random variations within the system, had mixed classification results among the different techniques. For fault 8, the PCA technique had a far higher fault classification rate compared to the other techniques. All the techniques struggled to classify fault 9; however, this was no surprise since fault 9 is difficult to classify, as mentioned in Section 2.6.2. The SVM technique struggled to classify fault 10 and had a significantly lower classification rate compared to the other techniques. Furthermore, the EGBV SVD technique had extremely poor fault classification rates for the random variation faults, and was outperformed by the other non-exergy and energy methods.

Regarding fault 13, which causes a slow drift within the system, the PCA technique had a significantly higher classification rate compared to the other techniques. The SVM and ANN techniques had lower but relatively still high classification rates compared to the PCA technique. The EGBV SVD technique had the lowest fault classification rate among the different techniques and had a significantly low classification rate.

Faults 14 and 15, exhibiting a sticking nature, had varying degrees of success in fault classification rates among the different techniques. The EGBV SVD technique had very poor classification rates and was unable to classify faults 14 and 15. For the non-exergy and energy methods, the SVM technique had a poor fault classification rate for fault 14, while the ANN technique had a higher fault classification rate compared to the other

techniques. Furthermore, all non-exergy and energy techniques had relatively low fault classification rates for fault 15, which was expected since it is a notoriously difficult fault to classify, as mentioned in Section 2.6.2.

The PCA and ANN techniques had relatively high fault classification rates for the different unknown faults (faults 16 - 20). Meanwhile, the SVM technique had mixed fault classification rates, and had especially poor classification rates for faults 16 and 19. Furthermore, the EGBV SVD technique had the lowest classification rates among the different techniques for the various unknown faults.

Comparing the average fault classification rates of the different techniques, it appears that the PCA and ANN techniques had significantly better fault classification rates compared to the other techniques. Furthermore, the ANN technique had the best average fault classification rate, slightly higher than the average fault classification rate of the PCA technique. The EGBV SVD technique had the lowest average fault classification rate, and was significantly lower than the average fault classification rates of the other techniques. Unlike the fault detection case, it appeared that the fault classification procedure implemented by the EGBV SVD technique did not offer more accurate fault classification, compared to the other non-exergy and energy methods. It can be argued that the poor performance of the EGBV SVD technique might be attributed to the fact that the fault signatures are not distinct enough to ensure that a fault can be uniquely identified (see Section 6.3.2).

6.6 Comparison with ML techniques

In this section, the fault detection and fault classification rates of all the techniques used in this study, including benchmark techniques, were compared for their ability to detect and classify the different types of faults. Furthermore, bar charts were used to visualise the comparative performance of the different techniques for the different types of faults. Additionally, the best-performing technique(s) for each type of fault were tabulated for both fault detection and classification.

6.6.1 Results

Fault detection

The fault detection rates of the different techniques per fault are illustrated in Figure 6.1. Note that the fault detection rates are expressed as percentages, and the accompanying colour legend on the right shows the different techniques. If a technique had a fault detection rate of 0 for a specific fault, there will be no coloured bar on the bar chart for

that specific fault.



Figure 6.1: Fault detection rates (%) for all techniques.

The best-performing fault detection technique(s) per fault are tabulated in Table 6.3. Note that the fault detection rates, inside the “Performance” column, are expressed as percentages. Furthermore, if multiple techniques have the best fault detection rate for a specific fault, then all the respective techniques are listed. In addition, the type of behaviour of each fault (see Section 2.6.2) is also listed in Table 6.3.

Table 6.3: Best fault detection rate (%) per fault.

Fault ID	Performance	Technique	Behaviour type
1	100	DNN CNN 1D CNN 2D GCN-Net ChebConv-Net GATConv-Net ICA One-class SVM EGBV SVD	Step
2	100	EGBV SVD	Step
3	100	DNN CNN 1D CNN 2D GCN-Net ChebConv-Net GATConv-Net EGBV SVD	Step
4	100	CNN 1D CNN 2D GCN-Net ChebConv-Net GATConv-Net PCA ICA PLS One-class SVM EGBV SVD	Step
5	100	DNN CNN 1D CNN 2D GCN-Net ChebConv-Net GATConv-Net PCA ICA One-class SVM EGBV SVD	Step

6	100	DNN CNN 1D CNN 2D GCN-Net ChebConv-Net GATConv-Net PCA ICA PLS One-class SVM EGBV SVD	Step
7	100	PCA ICA PLS One-class SVM	Step
8	99.94	EGBV SVD	Random variation
9	98.38	EGBV SVD	Random variation
10	99.06	EGBV SVD	Random variation
11	100	EGBV SVD	Random variation
12	100	EGBV SVD	Random variation
13	99.88	EGBV SVD	Slow drift
14	86.13	EGBV SVD	Sticking
15	86.88	EGBV SVD	Sticking
16	96.88	One-class SVM	Unknown
17	96.69	EGBV SVD	Unknown
18	93.02	EGBV SVD	Unknown
19	97	One-class SVM	Unknown
20	97.25	ICA One-class SVM	Unknown

Fault classification

The fault classification rates of the different techniques per fault are shown in Figure 6.2. Note that the fault classification rates are expressed as percentages, and the accompanying colour legend on the right shows the different techniques. If a technique had a fault classification rate of 0 for a specific fault, no coloured bar will show on the bar chart for that specific fault.

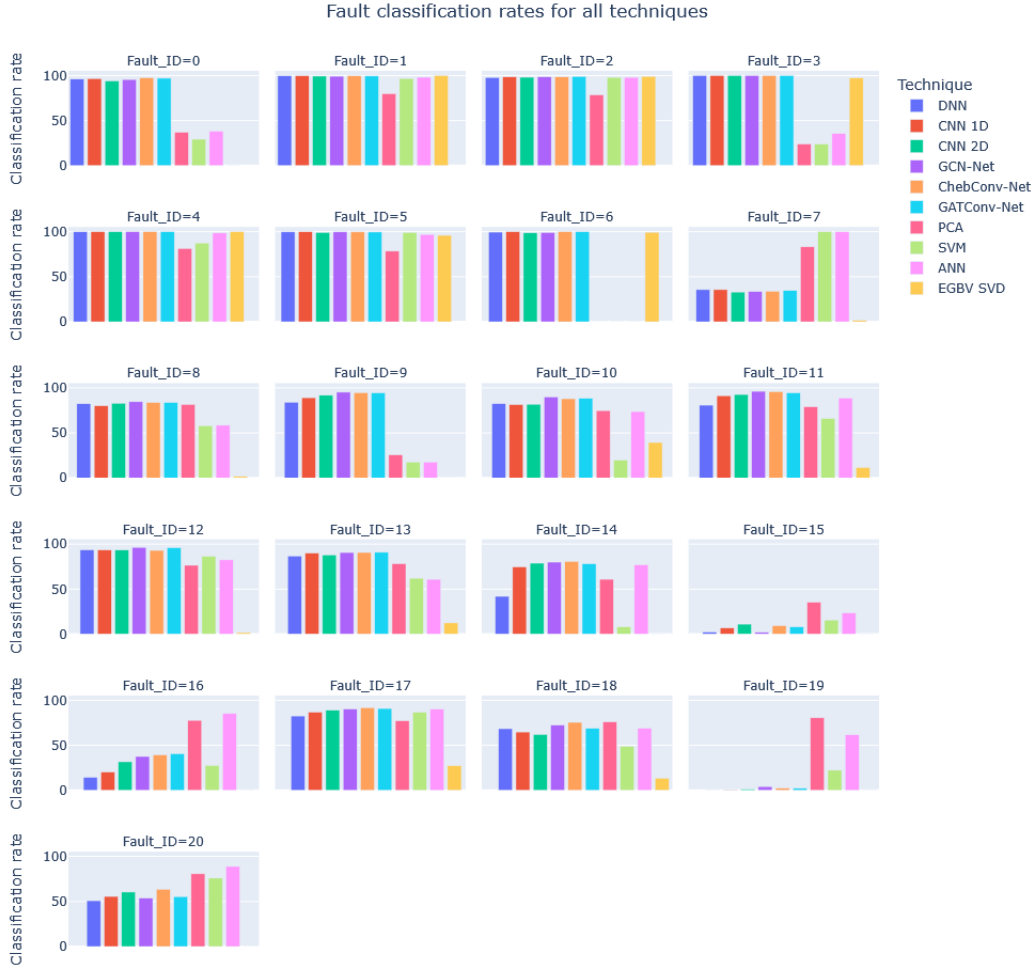


Figure 6.2: Fault classification rates (%) for all techniques.

Table 6.4 summarises the best performing fault classification technique(s) per fault. Note that the values within the “Performance” column represent the fault classification rates, which are expressed in percentages. If several techniques had the best fault classification rate for a specific fault, then all the techniques were listed. Furthermore, the type of behaviour of each fault (see Section 2.6.2) was also included in Table 6.4.

Table 6.4: Best fault classification rate (%) per fault.

Fault ID	Performance	Technique	Behaviour type
0	97.54	ChebConv-Net	Step
1	100	EGBV SVD	Step
2	99.13	EGBV SVD	Step
3	100	DNN CNN 1D CNN 2D GCN-Net ChebConv-Net GATConv-Net	Step
4	100	CNN 1D CNN 2D GCN-Net ChebConv-Net GATConv-Net EGBV SVD	Step
5	99.94	CNN 1D	Step
6	100	CNN 1D ChebConv-Net GATConv-Net	Step
7	100	SVM	Step
8	84.69	GCN-Net	Random variation
9	95.25	GCN-Net	Random variation
10	90.06	GCN-Net	Random variation
11	96.25	GCN-Net	Random variation
12	96.38	GCN-Net	Random variation
13	90.88	GATConv-Net	Slow drift
14	80.81	ChebConv-Net	Sticking
15	35.48	PCA	Sticking
16	85.27	ANN	Unknown
17	91.75	ChebConv-Net	Unknown
18	75.93	PCA	Unknown
19	80.54	PCA	Unknown
20	88.89	ANN	Unknown

6.6.2 Discussion

Fault detection:

Comparing the fault detection rates for the faults exhibiting a step-like behaviour (faults 1 - 7), it was apparent from the bar charts for faults 3 and 7 (see Figure 6.1) that there were large variations in the performance of the different techniques. The bar chart for fault 3 showed that the non-exergy and energy benchmark methods severely underperformed compared to the exergy and energy methods. However, as mentioned in Section 2.6.2, fault 3 is difficult to detect using the raw measurements. For fault 7 the non-exergy and energy benchmark methods performed better compared to the exergy and energy methods. Furthermore, for the exergy and energy methods the EGBV SVD benchmark method had a far superior detection rate for fault 7 compared to the exergy and energy methods that used ML. As mentioned in Chapters 4 and 5, the poor fault detection rates of the exergy and energy methods that use ML may be due to the data of fault 7 not having a clear distinction between “normal” and “faulty” samples, as seen with the results of the EDA.

For faults 8 – 12 (random variation faults), only the bar charts for faults 9 and 10 showed drastic variations in fault detection performance between the different techniques. Non-exergy and energy benchmark methods had poor detection rates compared to the exergy and energy methods for fault 9. Similar to fault 3, fault 9 is also difficult to detect when only using the raw measurements as mentioned in Section 2.6.2. For fault 10, the PCA and PLS methods had the lowest detection rates, while the rest of the methods had relatively high detection rates.

Fault 13, exhibiting a slow drift nature, had relatively consistent fault detection results between the different techniques. Figure 6.1, indicates that the fault detection rates of the different techniques for fault 13 were high and comparable with each other. Although the different techniques had comparable fault detection rates, the technique with the best fault detection rate for fault 13 was the EGBV SVD technique.

The bar charts for faults 14 and 15 (sticking nature) showed large variations in the fault detection rates between the different techniques (Figure 6.1). Similar to fault 10, the PCA and PLS techniques had the lowest detection rates for fault 14. Furthermore, it seemed that the exergy and energy ML methods that used the node signature matrix data (GCN-Net, ChebConv-Net, and GATConv-Net) had overall better detection rates for fault 14 when compared to the exergy and energy ML methods that used the cost matrix data (DNN, CNN 1D, and CNN 2D). For fault 15, the EGBV SVD technique showed the highest fault detection rate, while the other techniques had extremely low detection rates. As mentioned in Section 2.6.2 fault 15 is difficult to detect.

For the unknown faults (faults 16 - 20), the ML exergy and energy methods were out-

performed by several of the benchmark methods. As seen in Table 6.3, each of the best performing fault detection techniques for faults 16 – 20 was a benchmark technique. Furthermore, the results of Table 6.3 indicated that the one-class SVM technique was overall the most accurate technique for detecting unknown faults due to the number of times it occurred, followed by the EGBV SVD technique.

The best fault detection technique was the EGBV SVD technique and was identified as the best detection technique for sixteen out of the twenty faults, as shown in Table 6.3. The four faults where the EGBV SVD technique was not the best technique, include faults 7, 16, 19 and 20. However, the bar charts for the four faults mentioned (Figure 6.1) illustrated that the EGBV SVD technique was not far behind in terms of detection rate compared to the best technique(s) of the four mentioned faults. Furthermore, three of the four faults were unknown faults and thus it seemed that the EGBV SVD technique struggled to detect unknown faults compared to the other benchmark techniques.

Fault classification:

The exergy and energy methods using ML had no trouble to classify “normal” samples (fault 0) correctly, as shown in the bar chart depicted in Figure 6.2. On the contrary, the benchmark methods struggled to classify “normal” samples, resulting in low fault classification rates. Among the benchmark methods, the EGBV SVD technique had the lowest fault classification rate, which could be attributed to the way the fault signature matrices are constructed, as mentioned in Section 6.3.2.

Faults with a step-like behaviour (faults 1 - 7), had high fault classification rates among the ML exergy and energy methods for most of the faults, except for fault 7 which was similar to the fault detection results. Furthermore, the non-exergy and energy benchmark methods, except the PCA technique, had relatively high fault classification rates for most of the step-like faults. However, the non-exergy and energy benchmark methods had poor classification rates for faults 3 and 6. The poor fault classification rates for fault 3 were expected, as fault 3 is difficult to classify using raw measurements, as mentioned in Section 2.6.2. The extremely poor fault classification rates for fault 6 might be attributed to the TE Simulink model, as mentioned in Section 6.5. The EGBV SVD technique had excellent fault classification rates for most of the step-like faults, except fault 7. The poor fault classification rate for fault 7 using the EGBV SVD technique could be attributed to the fault signature matrix not being distinct enough, as mentioned in Section 6.3.2.

Most of the ML exergy and energy techniques outperformed the benchmark techniques for the random variation faults (faults 8 - 12); except for faults 8 and 11, where the CNN 1D technique had a lower classification rate compared to the PCA technique and the DNN technique a lower classification rate compared to the ANN technique respectively

(see Figure 6.2). The non-exergy and energy benchmark techniques struggled to classify fault 9 correctly. However, as mentioned in Section 2.6.2, fault 9 is a difficult fault to classify using the raw measurements. Among the benchmark techniques, the EGBV SVD technique overall had the lowest classification rates for the random variation faults.

The benchmark techniques had much lower fault classification rates for fault 13 (slow drift), compared to the ML exergy and energy methods. The ML exergy and energy techniques that use the node signature matrix data (GCN-Net, ChebConv-Net, and GATConv-Net) had slightly better classification rates compared to the other ML exergy and energy techniques (DNN, CNN 1D, and CNN 2D). Furthermore, among the benchmark techniques, the EGBV SVD technique had the lowest fault classification rate for fault 13.

Faults with a sticking nature (faults 14 and 15) showed mixed classification rate results. For fault 14, most of the ML exergy and energy techniques, except for the DNN and CNN 1D techniques, performed better than the benchmark techniques. Conversely, the non-exergy and energy benchmark techniques outperformed the rest of the exergy and energy techniques for fault 15. Although the non-exergy and energy benchmark techniques outperformed the rest of the techniques, their fault classification rates for fault 15 were relatively low. As mentioned in Section 2.6.2, it is difficult to accurately classify fault 15 using the raw measurements of the TEP.

For some of the unknown faults (faults 16 - 20), the non-exergy and energy benchmark techniques significantly outperformed the other techniques, especially for faults 16, 19 and 20. The remaining faults, faults 17 and 18, indicated that the ML exergy and energy methods had better or comparable fault classification rates compared to some of the non-exergy and energy benchmark methods. Overall, based on the results in Table 6.4, the non-exergy and energy benchmark techniques seemed to be better equipped to classify unknown faults compared to the exergy and energy techniques.

To summarise, the best performing technique(s) in Table 6.4 were the ML exergy and energy techniques that used the node signature matrix data. Therefore, it appeared that using the node signature matrix data offered better fault classification instead of the cost matrix data. Furthermore, using ML techniques for the exergy and energy data seemed to allow for more accurate fault classification, compared to the statistical fault signature approach used in the EGBV SVD technique.

6.7 Conclusion

In this chapter, different benchmark techniques for fault detection and fault classification were investigated and compared to the ML-based techniques in Chapters 4 and 5. The benchmark techniques included statistical, ML, and exergy and energy-based techniques.

Furthermore, it was discovered that the benchmark exergy and energy technique, the EGBV SVD technique, had superior fault detection rates compared to the other techniques. However, it had extremely poor fault classification performance compared to the other techniques. For fault classification, it was observed that the ML techniques investigated in Chapter 5, using the node signature matrix data, offered better fault classification compared to the rest of the techniques. Finally, it was concluded that some of the non-exergy and energy benchmark methods outperformed the exergy and energy techniques on several faults for both fault detection and classification, especially for the unknown faults.

Chapter 7

Conclusion

7.1 Summary and conclusions

Various ML techniques, were applied to the different levels of abstraction within the EGBV technique, and were compared to benchmark FDI techniques to highlight the merits and shortcomings of the different techniques. The four research objectives outlined in Chapter 1 have been achieved, consequently answering the research question.

7.1.1 Exploratory data analysis of cost matrix and node signature matrix data

Exploratory data analysis (EDA) was performed on the data from the cost and node signature matrices in Chapter 3. PCA, distance matrices, and self-organising maps were used to explore the data for the cost and node signature matrices. The techniques mentioned indicated that it may be inherently more difficult to detect and classify certain faults compared to others. As mentioned in Section 3.1, EDA is useful for generating hypotheses. Therefore, the EDA served as a basis for potentially explaining why certain faults were difficult to detect and classify. However, the hypotheses generated with the EDA could only be confirmed or dismissed once ML techniques were applied to the data and their results were analysed. Examples, where hypotheses generated by the EDA were confirmed, include faults 15 and 19 (see Tables 3.7 and 3.8), which had extremely poor fault detection and classification rates. On the other hand, hypotheses that were dismissed include faults 3 and 9 for the cost matrix data (see Table 3.7) and fault 14 for the node signature matrix data (see Table 3.8), which had unexpectedly high fault detection and classification rates.

7.1.2 Application and evaluation of machine learning techniques for the cost matrix data

Three traditional non-graphical ML techniques, able to process Euclidean data, were investigated and applied to the cost matrix data. The three techniques included a DNN, a one-dimensional CNN and a two-dimensional CNN. These three mentioned techniques were chosen, since they can process Euclidean data in different ways.

The fault detection and fault classification rates indicated that the two-dimensional CNN performed the best among the different techniques. Since the two-dimensional CNN performed the best, it appeared that the useful entries inside the cost matrix, which are used for fault detection and classification, were grouped locally and spanned across multiple rows and columns.

7.1.3 Application and evaluation of machine learning techniques for the node signature matrix data

Graphical ML techniques that can process non-Euclidean data structures, such as graphs, were investigated and applied to the node signature matrix data. Graph neural networks (GNNs) were considered as potential graphical ML techniques, and graph CNNs were the preferred choice. Three different types of graph CNNs, GCN-Net, ChebConv-Net, and GATConv-Net, were investigated and applied. Furthermore, the three above-mentioned graph CNNs consisted of spectral and spatial methods.

GCN-Net, ChebConv-Net, and GATConv-Net exhibited comparable fault detection rates, while ChebConv-Net had the highest overall fault classification rate. Therefore, it appeared that spectral convolutional GNNs were more suitable for the node signature matrix data compared to spatial convolutional GNNs. Compared to the two-dimensional CNN, the graph CNNs had overall lower fault detection rates. However, ChebConv-Net had the highest overall fault classification rate compared to the two-dimensional CNN. Therefore, the node signature matrix offers more accurate fault classification compared to the cost matrix.

7.1.4 Comparison between machine learning techniques and benchmark techniques

Several benchmark FDI techniques including the standard EGBV technique were implemented. The benchmark FDI techniques, except for the standard EGBV technique, used raw process measurements instead of exergy and energy measurements. The fault detection and fault classification rates of the benchmark techniques were compared with the

results obtained in Chapters 4 and 5 with the ML techniques.

The comparison indicated that the standard EGBV technique had superior fault detection capabilities and significantly outperformed the other techniques. Therefore, it appears that the cost matrix data offer the most accurate fault detection results overall. However, the standard EGBV technique had a poor overall fault classification performance, while the ChebConv-Net had the highest overall fault classification rate among the various techniques. Consequently, it appears that applying ML to the data of the node signature matrix offers the most accurate fault classification results overall.

7.2 Future work and recommendations

During the current study, several aspects were identified that require further investigation and improvement. The aspects are briefly mentioned and described in the subsequent subsections.

7.2.1 Feature selection for cost and node signature matrices

Several of the fault detection benchmark techniques in Chapter 6 namely, PCA, ICA, and PLS, used only a subset of the raw measurements/variables for fault detection purposes [99]. The performance of a data-driven FDI technique is affected by the choice of variables/features used [218]. Redundant and uninformative variables/features will cause the performance of the data-driven FDI technique to degrade. Conversely, using only informative variables/features will allow the performance of the data-driven FDI technique to improve.

The ML techniques implemented in Chapters 4 and 5, used all the variables/features inside the cost and node signature matrices. Some of the variables/features may have been redundant or uninformative, potentially causing the ML techniques to achieve lower fault detection and classification results with certain faults. Consequently, feature selection techniques that can determine the optimal features/variables inside of the cost and node signature matrices for the purposes of fault detection and classification should be investigated.

7.2.2 Alternative methods of fault signature generation with standard EGBV technique

The standard EGBV technique had superior fault detection capabilities compared to the EGBV technique that used ML techniques. However, the fault classification performance

of the standard EGBV technique was extremely poor. The poor fault classification performance of the standard EGBV technique was due to the fact that the fault signatures were not distinct enough or equidistant from the signature of the current sample.

Fault signatures are used by the standard EGBV technique to classify faults by comparing each fault signature against the signature of the current sample. The fault signature that is most similar to the signature of the current sample is indicative of the type of fault. Therefore, if the fault signatures are too similar to each other or equidistant from the signature of the current sample, the current sample cannot be uniquely identified, leading to poor fault classification results. Therefore, alternative methods for generating more distinct fault signatures for the standard EGBV technique should be investigated to potentially improve the fault classification capabilities of the standard EGBV technique.

7.2.3 Comparison with temporal machine learning techniques

All the ML techniques implemented in Chapters 4 and 5 do not consider the temporal relationships between samples, meaning that they consider each sample independently. On the contrary, temporal ML techniques can consider the temporal relationship between samples, which makes them ideal for fault diagnosis purposes in systems with changing dynamics [219]. Representing the exergy and energy information of the system as a time series and implementing temporal ML techniques might be an interesting comparison against the sample-by-sample ML techniques of Chapters 4 and 5.

7.2.4 Hybrid FDI techniques

The cost matrix data in conjunction with the standard EGBV technique achieved excellent fault detection. Meanwhile, the node signature matrix data paired with the graph convolution neural networks resulted in the highest overall fault classification rates. Consequently, it appears that the data inside the cost matrix and node signature matrices have their respective strengths for fault detection and fault classification, respectively. Therefore, future work where hybrid FDI techniques use the combined data from both the cost and node signature matrices for fault detection and classification may warrant further investigation.

7.2.5 Explainable AI

The ML techniques used in Chapters 4 and 5 are regarded as black box models, meaning that the inner workings of the model are not easily interpretable by humans. The lack of interpretability makes it difficult for these ML techniques to be adopted in real-world

settings where safety is vital and human operators need to trust the techniques [220]. In addition, the ability to interpret the predictions made by ML models makes it possible to identify the root causes resulting in certain incorrect predictions [221]. Consequently, future work is required to improve the interpretability of ML models applied within the EGBV context.

7.3 Closure

In this study, the objective of comparing the ML techniques used within the EGBV context against other benchmark FDI techniques was achieved. The use of ML techniques within the EGBV context did not necessarily improve the fault detection ability compared to the benchmark FDI techniques. However, a significant improvement was achieved in terms of fault classification ability when ML techniques were used within the EGBV context, compared to the FDI benchmark techniques.

Bibliography

- [1] S. Yin, S. X. Ding, X. Xie, and H. Luo, “A review on basic data-driven approaches for industrial process monitoring,” *IEEE Transactions on Industrial Electronics*, vol. 61, pp. 6418–6428, 11 2014.
- [2] Y.-J. Park, S.-K. S. Fan, and C.-Y. Hsu, “A review on fault detection and process diagnostics in industrial processes,” *Processes*, vol. 8, p. 1123, 9 2020.
- [3] V. Venkatasubramanian, “Process fault detection and diagnosis: Past, present and future,” *IFAC Proceedings Volumes*, vol. 34, pp. 1–13, 6 2001.
- [4] J.-E. YANG, “Fukushima dai-ichi accident: Lessons learned and future actions from the risk perspectives,” *Nuclear Engineering and Technology*, vol. 46, pp. 27–38, 2 2014.
- [5] A. Labib and R. Champaneri, “The bhopal disaster-learning from failures and evaluating risk,” *Maintenance Engineering*, vol. 27, pp. 41–47, 2012.
- [6] I. Nimmo, “Adequately address abnormal operations,” *Chemical engineering progress*, vol. 91, no. 9, 1995.
- [7] S. Liu, F. Lei, D. Zhao, and Q. Liu, “Abnormal situation management in chemical processes: Recent research progress and future prospects,” *Processes*, vol. 11, p. 1608, 5 2023.
- [8] A. R. Sahu, S. K. Palei, and A. Mishra, “Data-driven fault diagnosis approaches for industrial equipment: A review,” *Expert Systems*, vol. 41, p. e13360, 2 2024. [Online]. Available: <https://doi.org/10.1111/exsy.13360>
- [9] D. Leite, E. Andrade, D. Rativa, and A. M. A. Maciel, “Fault detection and diagnosis in industry 4.0: A review on challenges and opportunities,” *Sensors*, vol. 25, p. 60, 12 2024.
- [10] R. J. Fickelscherer, *Artificial Intelligence in Process Fault Diagnosis: Methods for Plant Surveillance*. John Wiley & Sons, 2024.
- [11] W. Yan, J. Wang, S. Lu, M. Zhou, and X. Peng, “A review of real-time fault diagnosis methods for industrial smart manufacturing,” *Processes*, vol. 11, p. 369, 1 2023.

- [12] T. Escobet, A. Bregon, B. Pulido, and V. Puig, *Fault diagnosis of dynamic systems*. Springer, 2019.
- [13] I. P. Colo, C. S. Sueldo, M. De Paula, and G. G. Acosta, “Intelligent approach for the industrialization of deep learning solutions applied to fault detection,” *Expert Systems with Applications*, vol. 233, p. 120959, 2023.
- [14] G. Van Schoor, K. R. Uren, M. A. van Wyk, P. A. van Vuuren, and C. P. du Rand, “An energy perspective on modelling, supervision, and control of large-scale industrial systems: Survey and framework,” *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 6692–6703, 2014.
- [15] H. Marais, G. van Schoor, and K. R. Uren, “The merits of exergy-based fault detection in petrochemical processes,” *Journal of Process Control*, vol. 74, pp. 110–119, 2 2019.
- [16] K. R. Uren and G. van Schoor, “Reflection on the energy graph-based visualisation approach to fdi of large-scale industrial systems,” *IFAC-PapersOnLine*, vol. 56, pp. 4094–4107, 2023.
- [17] C. Reinartz, M. Kulahci, and O. Ravn, “An extended tennessee eastman simulation dataset for fault-detection and decision support systems,” *Computers & Chemical Engineering*, vol. 149, p. 107281, 6 2021.
- [18] H. Khorasgani, A. Hasanzadeh, A. Farahat, and C. Gupta, “Fault detection and isolation in industrial networks using graph convolutional neural networks,” in *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*. IEEE, 2019, pp. 1–7.
- [19] L. Shuaiyi, K. Wang, L. Zhang, and B. Wang, “Global-local integration for gnn-based anomalous device state detection in industrial control systems,” *Expert Systems with Applications*, vol. 209, p. 118345, 2022.
- [20] T. Wu, A. Scaglione, and D. Arnold, “Complex-value spatiotemporal graph convolutional neural networks and its applications to electric power systems ai,” *IEEE Transactions on Smart Grid*, vol. 15, pp. 3193–3207, 5 2024.
- [21] K. He, D. Yu, D. Wang, M. Chai, S. Lei, and C. Zhou, “Graph attention network-based fault detection for uavs with multivariant time series flight data,” *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–13, 2022.
- [22] Z. Li, J. Ma, R. Fan, Y. Zhao, J. Ai, and Y. Dong, “Aircraft sensor fault diagnosis based on graphsage and attention mechanism,” *Sensors*, vol. 25, p. 809, 1 2025.
- [23] D. Chen, R. Liu, Q. Hu, and S. X. Ding, “Interaction-aware graph neural networks for fault diagnosis of complex industrial processes,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, pp. 6015–6028, 9 2023.

- [24] W. H. Choi, J. Kim, and J. Y. Lee, “Development of fault diagnosis models based on predicting energy consumption of a machine tool spindle,” *Procedia Manufacturing*, vol. 51, pp. 353–358, 2020.
- [25] D. J. R. Orozco, O. J. Venturini, J. C. E. Palacio, and O. A. del Olmo, “A new methodology of thermodynamic diagnosis, using the thermoeconomic method together with an artificial neural network (ann): A case study of an externally fired gas turbine (efgt),” *Energy*, vol. 123, pp. 20–35, 3 2017.
- [26] L. Wang, P. Fu, N. Wang, T. Morosuk, Y. Yang, and G. Tsatsaronis, “Malfunction diagnosis of thermal power plants based on advanced exergy analysis: The case with multiple malfunctions occurring simultaneously,” *Energy Conversion and Management*, vol. 148, pp. 1453–1467, 9 2017.
- [27] Y. Cao, X. Lv, G. Han, J. Luan, and S. Li, “Research on gas-path fault-diagnosis method of marine gas turbine based on exergy loss and probabilistic neural network,” *Energies*, vol. 12, p. 4701, 12 2019.
- [28] Z. Wang, B. Yang, Q. Ma, H. Wang, R. Carriveau, D. S. K. Ting, and W. Xiong, “Facilitating energy monitoring and fault diagnosis of pneumatic cylinders with exergy and machine learning,” *International Journal of Fluid Power*, pp. 643–682, 2023.
- [29] A. Santolamazza, V. Cesarotti, and V. Introna, “Anomaly detection in energy consumption for condition-based maintenance of compressed air generation systems: an approach based on artificial neural networks,” *IFAC-PapersOnLine*, vol. 51, pp. 1131–1136, 2018.
- [30] A. Picallo-Perez, J. M. Sala-Lizarraga, E. Iribar-Solabarrieta, M. Odriozola-Maritorena, and L. Portillo-Valdés, “Application of the malfunction thermoeconomic diagnosis to a dynamic heating and dhw facility for fault detection,” *Energy and Buildings*, vol. 135, pp. 385–397, 2017.
- [31] N. Xu, L. Yang, Y. Guo, L. Chang, G. Zhang, and J. Zhang, “An improved thermoeconomic diagnosis method: Applying to marine diesel engines,” *Journal of Marine Science and Engineering*, vol. 13, p. 244, 1 2025.
- [32] Z. An, S. Li, J. Wang, and X. Jiang, “A novel bearing intelligent fault diagnosis framework under time-varying working conditions using recurrent neural network,” *ISA Transactions*, vol. 100, pp. 155–170, 5 2020.
- [33] L. Tightiz, M. A. Nasab, H. Yang, and A. Addeh, “An intelligent system based on optimized anfis and association rules for power transformer fault diagnosis,” *ISA Transactions*, vol. 103, pp. 63–74, 8 2020.

- [34] Y. H. Chae, S. G. Kim, H. Kim, J. T. Kim, and P. H. Seong, “A methodology for diagnosing fac induced pipe thinning using accelerometers and deep learning models,” *Annals of Nuclear Energy*, vol. 143, p. 107501, 8 2020.
- [35] W. Cai, W. Zhang, X. Hu, and Y. Liu, “A hybrid information model based on long short-term memory network for tool condition monitoring,” *Journal of Intelligent Manufacturing*, vol. 31, pp. 1497–1510, 8 2020.
- [36] T. Ince, S. Kiranyaz, L. Eren, M. Askar, and M. Gabbouj, “Real-time motor fault detection by 1-d convolutional neural networks,” *IEEE Transactions on Industrial Electronics*, vol. 63, pp. 7067–7075, 11 2016.
- [37] L. Wen, X. Li, L. Gao, and Y. Zhang, “A new convolutional neural network-based data-driven fault diagnosis method,” *IEEE Transactions on Industrial Electronics*, vol. 65, pp. 5990–5998, 7 2018.
- [38] C. Che, H. Wang, X. Ni, and Q. Fu, “Domain adaptive deep belief network for rolling bearing fault diagnosis,” *Computers & Industrial Engineering*, vol. 143, p. 106427, 5 2020.
- [39] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, “Do deep neural networks contribute to multivariate time series anomaly detection?” *Pattern Recognition*, vol. 132, p. 108945, 12 2022.
- [40] L. H. Chiang, E. L. Russell, and R. D. Braatz, *Fault Detection and Diagnosis in Industrial Systems*. Springer London, 2001.
- [41] R. Isermann, *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. Springer Science & Business Media, 2006.
- [42] R. Isermann and P. Ballé, “Trends in the application of model based fault detection and diagnosis of technical processes,” *IFAC Proceedings Volumes*, vol. 29, pp. 6325–6336, 6 1996.
- [43] C. Aldrich and L. Auret, *Unsupervised Process Monitoring and Fault Diagnosis with Machine Learning Methods*. Springer London, 2013.
- [44] R. Isermann, “Supervision, fault-detection and fault-diagnosis methods — an introduction,” *Control Engineering Practice*, vol. 5, pp. 639–652, 5 1997.
- [45] —, *Fault-Diagnosis Applications*. Springer Berlin Heidelberg, 2011.
- [46] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. N. Kavuri, “A review of process fault detection and diagnosis: Part i: Quantitative model-based methods,” *Computers & Chemical Engineering*, vol. 27, pp. 293–311, 3 2003.
- [47] J. Korbicz, J. M. Koscielny, Z. Kowalczyk, and W. Cholewa, *Fault diagnosis: models, artificial intelligence, applications*. Springer Science & Business Media, 2012.

- [48] K. Zhong, M. Han, and B. Han, “Data-driven based fault prognosis for industrial systems: a concise overview,” *IEEE/CAA Journal of Automatica Sinica*, vol. 7, pp. 330–345, 3 2020.
- [49] S. A. A. Taqvi, H. Zabiri, L. D. Tufa, F. Uddin, S. A. Fatima, and A. S. Maulud, “A review on data-driven learning approaches for fault detection and diagnosis in chemical processes,” *ChemBioEng Reviews*, vol. 8, pp. 239–259, 6 2021.
- [50] N. Bayar, S. Darmoul, S. Hajri-Gabouj, and H. Pierreval, “Fault detection, diagnosis and recovery using artificial immune systems: A review,” *Engineering Applications of Artificial Intelligence*, vol. 46, pp. 43–57, 11 2015.
- [51] J. Wang, J. Zhou, and X. Chen, *Data-Driven Fault Detection and Reasoning for Industrial Monitoring*. Springer Singapore, 2022, vol. 3.
- [52] S. X. Ding, *Data-driven Design of Fault Diagnosis and Fault-tolerant Control Systems*. Springer London, 2014.
- [53] S. Sankararaman, “Significance, interpretation, and quantification of uncertainty in prognostics and remaining useful life prediction,” *Mechanical Systems and Signal Processing*, vol. 52-53, pp. 228–247, 2 2015.
- [54] E. Zio, “Prognostics and health management (phm): Where are we and where do we (need to) go in theory and practice,” *Reliability Engineering & System Safety*, vol. 218, p. 108119, 2 2022.
- [55] K. Tidriri, N. Chatti, S. Verron, and T. Tiplica, “Bridging data-driven and model-based approaches for process fault diagnosis and health monitoring: A review of researches and future challenges,” *Annual Reviews in Control*, vol. 42, pp. 63–81, 2016.
- [56] X. Dai and Z. Gao, “From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis,” *IEEE Transactions on Industrial Informatics*, vol. 9, pp. 2226–2238, 11 2013.
- [57] M. Thirumarimurugan, N. Bagyalakshmi, and P. Paarkavi, “Comparison of fault detection and isolation methods: A review,” in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*. IEEE, 2016, pp. 1–6.
- [58] A. H. Sabry and U. A. B. U. Amirulddin, “A review on fault detection and diagnosis of industrial robots and multi-axis machines,” *Results in Engineering*, vol. 23, p. 102397, 9 2024.
- [59] V. Pashazadeh, F. R. Salmasi, and B. N. Araabi, “Data driven sensor and actuator fault detection and isolation in wind turbine using classifier fusion,” *Renewable Energy*, vol. 116, pp. 99–106, 2 2018.

- [60] K. Worden and J. M. Dulieu-Barton, “An overview of intelligent fault detection in systems and structures,” *Structural Health Monitoring*, vol. 3, pp. 85–98, 3 2004.
- [61] M. A. Djeziri, B. O. Bouamama, R. Merzouki, and G. Dauphin-Tanguy, “Optimal sensor placement for fault diagnosis,” in *2009 IEEE International Conference on Mechatronics*. IEEE, 2009, pp. 1–6.
- [62] D. J. Hill and B. S. Minsker, “Anomaly detection in streaming environmental sensor data: A data-driven modeling approach,” *Environmental Modelling & Software*, vol. 25, pp. 1014–1022, 9 2010.
- [63] A. Abid, M. T. Khan, and J. Iqbal, “A review on fault detection and diagnosis techniques: basics and beyond,” *Artificial Intelligence Review*, vol. 54, pp. 3639–3664, 6 2021.
- [64] K. Severson, P. Chaiwatanodom, and R. D. Braatz, “Perspectives on process monitoring of industrial systems,” *Annual Reviews in Control*, vol. 42, pp. 190–200, 2016.
- [65] M. Y. Aalsalem, W. Z. Khan, W. Gharibi, M. K. Khan, and Q. Arshad, “Wireless sensor networks in oil and gas industry: Recent advances, taxonomy, requirements, and open challenges,” *Journal of Network and Computer Applications*, vol. 113, pp. 87–97, 7 2018.
- [66] T. Muhammed and R. A. Shaikh, “An analysis of fault detection strategies in wireless sensor networks,” *Journal of Network and Computer Applications*, vol. 78, pp. 267–287, 1 2017.
- [67] P. Kumar and R. Tiwari, “Additive fault diagnosis techniques in rotor systems: a state-of-the-art review,” *Sāadhanā*, vol. 49, p. 207, 6 2024.
- [68] Z. Zhao, Y. Yang, S. X. Ding, and L. Li, “Fault-tolerant control for systems with model uncertainty and multiplicative faults,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, pp. 514–524, 2 2020.
- [69] B. Cai, Y. Liu, and M. Xie, “A dynamic-bayesian-network-based fault diagnosis methodology considering transient and intermittent faults,” *IEEE Transactions on Automation Science and Engineering*, vol. 14, pp. 276–285, 1 2017.
- [70] I. Koren and C. M. Krishna, *Fault-tolerant systems*. Morgan Kaufmann, 2020.
- [71] D. Zhou, Y. Zhao, Z. Wang, X. He, and M. Gao, “Review on diagnosis techniques for intermittent faults in dynamic systems,” *IEEE Transactions on Industrial Electronics*, vol. 67, pp. 2337–2347, 3 2020.
- [72] H. Safaeipour, M. Forouzanfar, and A. Casavola, “A survey and classification of incipient fault diagnosis approaches,” *Journal of Process Control*, vol. 97, pp. 1–16, 1 2021.

- [73] S. Khanna and S. Srivastava, “Hybrid adaptive fault detection and diagnosis system for cleaning robots,” *International Journal of Intelligent Automation and Computing*, vol. 7, pp. 1–14, 2024.
- [74] V. Venkatasubramanian, R. Rengaswamy, and S. N. Kavuri, “A review of process fault detection and diagnosis: Part ii: Qualitative models and search strategies,” *Computers & Chemical Engineering*, vol. 27, pp. 313–326, 3 2003.
- [75] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, and K. Yin, “A review of process fault detection and diagnosis: Part iii: Process history based methods,” *Computers & Chemical Engineering*, vol. 27, pp. 327–346, 3 2003.
- [76] W. Li, H. Li, S. Gu, and T. Chen, “Process fault diagnosis with model- and knowledge-based approaches: Advances and opportunities,” *Control Engineering Practice*, vol. 105, p. 104637, 12 2020.
- [77] F. Calabrese, A. Regattieri, M. Bortolini, and F. G. Galizia, “Data-driven fault detection and diagnosis: Challenges and opportunities in real-world scenarios,” *Applied Sciences*, vol. 12, p. 9212, 9 2022.
- [78] S. R. Saufi, Z. A. B. Ahmad, M. S. Leong, and M. H. Lim, “Challenges and opportunities of deep learning models for machinery fault detection and diagnosis: A review,” *IEEE Access*, vol. 7, pp. 122 644–122 662, 2019.
- [79] K. Zope, K. Singh, S. H. Nistala, A. Basak, P. Rathore, and V. Runkana, “Anomaly detection and diagnosis in manufacturing systems: A comparative study of statistical, machine learning and deep learning techniques,” in *Annu. Conf. PHM Soc*, vol. 11, 2019.
- [80] J. Zheng, C. Zhao, and F. Gao, “Retrospective comparison of several typical linear dynamic latent variable models for industrial process monitoring,” *Computers & Chemical Engineering*, vol. 157, p. 107587, 1 2022.
- [81] Y. Lei, B. Yang, X. Jiang, F. Jia, N. Li, and A. K. Nandi, “Applications of machine learning to machine fault diagnosis: A review and roadmap,” *Mechanical Systems and Signal Processing*, vol. 138, p. 106587, 4 2020.
- [82] M. Fernandes, J. M. Corchado, and G. Marreiros, “Machine learning techniques applied to mechanical fault diagnosis and fault prognosis in the context of real industrial manufacturing use-cases: a systematic literature review,” *Applied Intelligence*, vol. 52, pp. 14 246–14 280, 2022. [Online]. Available: <https://doi.org/10.1007/s10489-022-03344-3>
- [83] R. Isermann, “Model-based fault-detection and diagnosis – status and applications,” *Annual Reviews in Control*, vol. 29, pp. 71–85, 1 2005.

- [84] R. J. Patton, P. M. Frank, and R. N. Clark, *Issues of fault diagnosis for dynamic systems*. Springer Science & Business Media, 2013.
- [85] J. Marzat, H. Piet-Lahanier, F. Damongeot, and E. Walter, “Model-based fault diagnosis for aerospace systems: a survey,” *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 226, pp. 1329–1360, 10 2012.
- [86] X. Ying, “An overview of overfitting and its solutions,” *Journal of Physics: Conference Series*, vol. 1168, p. 022022, 2 2019.
- [87] R. Iqbal, T. Maniak, F. Doctor, and C. Karyotis, “Fault detection and isolation in industrial processes using deep learning approaches,” *IEEE Transactions on Industrial Informatics*, vol. 15, pp. 3077–3084, 5 2019.
- [88] P. Mercorelli, “Recent advances in intelligent algorithms for fault detection and diagnosis,” *Sensors*, vol. 24, p. 2656, 4 2024.
- [89] J. Yu and Y. Zhang, “Challenges and opportunities of deep learning-based process fault detection and diagnosis: a review,” *Neural Computing and Applications*, vol. 35, pp. 211–252, 1 2023.
- [90] G. van Schoor and K. Uren, “A vision of energy-based visualisation of large scale industrial systems for the purpose of condition monitoring.” *31st Conference on Condition Monitoring and Diagnostic Engineering Management, Rustenburg, South Africa*, pp. 337–346, 7 2018.
- [91] M. J. Moran and E. Sciubba, “Exergy analysis: Principles and practice,” *Journal of Engineering for Gas Turbines and Power*, vol. 116, pp. 285–290, 4 1994.
- [92] J. Szargut, *Exergy method: technical and ecological applications*. WIT press, 2005, vol. 18.
- [93] H. Naser, G. van Schoor, and K. Uren, “Energy-based fault detection and isolation of a brayton cycle-based htgr power conversion unit – a comparative study,” *Annals of Nuclear Energy*, vol. 164, p. 108616, 12 2021.
- [94] S. Van Graan, G. Van Schoor, and K. Uren, “Graph matching as a means to energy-visualisation of a counter-flow heat exchanger for the purpose of fault diagnosis,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 2842–2847, 2017.
- [95] K. R. Uren, G. van Schoor, M. van Eldik, and J. J. A. de Bruin, “An energy graph-based approach to fault diagnosis of a transcritical co2 heat pump,” *Energies*, vol. 13, p. 1783, 4 2020.
- [96] S. Greyling, G. van Schoor, K. R. Uren, and H. Marais, “An energy graph eigen-decomposition approach to fault detection and isolation applied to a gas-to-liquids process,” *Computers & Chemical Engineering*, vol. 168, p. 108040, 12 2022.

- [97] —, “Exergy graph-based fault detection and isolation of a gas-to-liquids process,” *IFAC-PapersOnLine*, vol. 53, pp. 13 674–13 681, 2020.
- [98] J. Downs and E. Vogel, “A plant-wide industrial process problem control,” *Comput Chem Eng*, vol. 17, pp. 245–255, 1993.
- [99] S. Yin, S. X. Ding, A. Haghani, H. Hao, and P. Zhang, “A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark tennessee eastman process,” *Journal of Process Control*, vol. 22, pp. 1567–1581, 10 2012.
- [100] E. B. Andersen, I. A. Udugama, K. V. Gernaey, A. R. Khan, C. Bayer, and M. Kulaici, “An easy to use gui for simulating big data using tennessee eastman process,” *Quality and Reliability Engineering International*, vol. 38, pp. 264–282, 2 2022.
- [101] C. Martin-Villalba, A. Urquia, and G. Shao, “Implementations of the tennessee eastman process in modelica,” *IFAC-PapersOnLine*, vol. 51, no. 2, pp. 619–624, 2018.
- [102] N. L. Ricker, “Decentralized control of the tennessee eastman challenge process,” *Journal of process control*, vol. 6, no. 4, pp. 205–221, 1996.
- [103] D. R. Vinson, C. Georgakis, and J. Fossy, “Studies in plant-wide controllability using the tennessee eastman challenge problem, the case for multivariable control,” in *Proceedings of 1995 American Control Conference-ACC’95*, vol. 1. IEEE, 1995, pp. 250–254.
- [104] E. Ramírez and R. Gani, “Methodology for the design and analysis of reaction-separation systems with recycle. 2. design and control integration,” *Industrial & engineering chemistry research*, vol. 46, no. 24, pp. 8084–8100, 2007.
- [105] T. McAvoy and N. Ye, “Base control for the tennessee eastman problem,” *Computers & Chemical Engineering*, vol. 18, pp. 383–413, 5 1994.
- [106] A. Banerjee and Y. Arkun, “Control configuration design applied to the tennessee eastman plant-wide control problem,” *Computers & Chemical Engineering*, vol. 19, pp. 453–480, 4 1995.
- [107] W. L. Luyben, “Simple regulatory control of the eastman process,” *Industrial & Engineering Chemistry Research*, vol. 35, pp. 3280–3289, 1 1996.
- [108] K.-L. Wu and C.-C. Yu, “Operability for processes with recycles: Interaction between design and operation with application to the tennessee eastman challenge process,” *Industrial & Engineering Chemistry Research*, vol. 36, pp. 2239–2251, 6 1997.

- [109] G. R. Srinivas and Y. Arkun, "Control of the tennessee eastman process using input-output models," *Journal of Process Control*, vol. 7, no. 5, pp. 387–400, 1997.
- [110] P. Lyman and C. Georgakis, "Plant-wide control of the tennessee eastman problem," *Computers & Chemical Engineering*, vol. 19, pp. 321–331, 3 1995.
- [111] T. Larsson, K. Hestetun, E. Hovland, and S. Skogestad, "Self-optimizing control of a large-scale plant: The tennessee eastman process," *Industrial & Engineering Chemistry Research*, vol. 40, pp. 4889–4901, 10 2001.
- [112] S. Heo and J. H. Lee, "Fault detection and classification using artificial neural networks," *IFAC-PapersOnLine*, vol. 51, pp. 470–475, 2018.
- [113] P. Hajihosseini, M. M. Anzehaee, and B. Behnam, "Fault detection and isolation in the challenging tennessee eastman process by using image processing techniques," *ISA Transactions*, vol. 79, pp. 137–146, 8 2018.
- [114] K. Liu, Z. Fei, B. Yue, J. Liang, and H. Lin, "Adaptive sparse principal component analysis for enhanced process monitoring and fault isolation," *Chemometrics and Intelligent Laboratory Systems*, vol. 146, pp. 426–436, 8 2015.
- [115] Y. Zhang, "Enhanced statistical analysis of nonlinear processes using kpca, kica and svm," *Chemical Engineering Science*, vol. 64, pp. 801–811, 3 2009.
- [116] L. H. Chiang, M. E. Kotanchek, and A. K. Kordon, "Fault diagnosis based on fisher discriminant analysis and support vector machines," *Computers & Chemical Engineering*, vol. 28, pp. 1389–1401, 7 2004.
- [117] A. Kulkarni, V. Jayaraman, and B. Kulkarni, "Knowledge incorporated support vector machines to detect faults in tennessee eastman process," *Computers & Chemical Engineering*, vol. 29, pp. 2128–2133, 9 2005.
- [118] A. Bakdi and A. Kouadri, "An improved plant-wide fault detection scheme based on pca and adaptive threshold for reliable process monitoring: Application on the new revised model of tennessee eastman process," *Journal of Chemometrics*, vol. 32, p. e2978, 2018.
- [119] A. Bathelt, N. L. Ricker, and M. Jelali, "Revision of the tennessee eastman process model," *IFAC-PapersOnLine*, vol. 48, pp. 309–314, 2015.
- [120] J. Vosloo, K. Uren, G. van Schoor, L. Auret, and H. Marais, "Exergy-based fault detection on the tennessee eastman process," *IFAC-PapersOnLine*, vol. 53, pp. 13 713–13 720, 2020.
- [121] A. T. Jebb, S. Parrigon, and S. E. Woo, "Exploratory data analysis as a foundation of inductive research," *Human Resource Management Review*, vol. 27, pp. 265–276, 6 2017.

- [122] A. Bezerra, I. Silva, L. A. Guedes, D. Silva, G. Leitão, and K. Saito, “Extracting value from industrial alarms and events: A data-driven approach based on exploratory data analysis,” *Sensors*, vol. 19, p. 2772, 6 2019.
- [123] K. Pearson, “Liii. on lines and planes of closest fit to systems of points in space,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, pp. 559–572, 11 1901.
- [124] M. Greenacre, P. J. F. Groenen, T. Hastie, A. I. D’Enza, A. Markos, and E. Tuzhilina, “Principal component analysis,” *Nature Reviews Methods Primers*, vol. 2, p. 100, 12 2022.
- [125] M. C. Thomas, W. Zhu, and J. A. Romagnoli, “Data mining and clustering in chemical process databases for monitoring and knowledge discovery,” *Journal of Process Control*, vol. 67, pp. 160–175, 7 2018.
- [126] S. Zheng and J. Zhao, “A new unsupervised data mining method based on the stacked autoencoder for chemical process fault diagnosis,” *Computers & Chemical Engineering*, vol. 135, p. 106755, 4 2020.
- [127] M. Alabsi, Y. Liao, and A.-A. Nabulsi, “Bearing fault diagnosis using deep learning techniques coupled with handcrafted feature extraction: A comparative study,” *Journal of Vibration and Control*, vol. 27, pp. 404–414, 2 2021.
- [128] R. Redondo, Álvaro Herrero, E. Corchado, and J. Sedano, “A decision-making tool based on exploratory visualization for the automotive industry,” *Applied Sciences*, vol. 10, p. 4355, 6 2020.
- [129] M. A. Khan, B. Asad, K. Kudelina, T. Vaimann, and A. Kallaste, “The bearing faults detection methods for electrical machines—the state of the art,” *Energies*, vol. 16, p. 296, 12 2022.
- [130] A. Behnia, H. K. Chai, M. GhasemiGol, A. Sepehrinezhad, and A. A. Mousa, “Advanced damage detection technique by integration of unsupervised clustering into acoustic emission,” *Engineering Fracture Mechanics*, vol. 210, pp. 212–227, 4 2019.
- [131] W. Zhu, W. Sun, and J. Romagnoli, “Adaptive k-nearest-neighbor method for process monitoring,” *Industrial & Engineering Chemistry Research*, vol. 57, pp. 2574–2586, 2 2018.
- [132] A. Melo, F. F. Fadel, M. M. Câmara, and J. C. Pinto, “Distance matrix patterns for visual and interpretable process data analytics,” *Industrial & Engineering Chemistry Research*, vol. 62, pp. 13 889–13 901, 9 2023.
- [133] T. Kohonen, “The self-organizing map,” *Proceedings of the IEEE*, vol. 78, pp. 1464–1480, 1990.

- [134] R. Mordia and A. K. Verma, “Visual techniques for defects detection in steel products: A comparative study,” *Engineering Failure Analysis*, vol. 134, p. 106047, 4 2022.
- [135] I. S. Andrades, J. J. C. Aguilar, J. M. V. García, J. A. C. Carrillo, and M. S. Lozano, “Low-cost road-surface classification system based on self-organizing maps,” *Sensors*, vol. 20, p. 6009, 10 2020.
- [136] X. Chen and X. Yan, “Using improved self-organizing map for fault diagnosis in chemical industry process,” *Chemical Engineering Research and Design*, vol. 90, pp. 2262–2277, 12 2012.
- [137] S. Schwartz, J. J. M. Jimenez, M. Salaün, and R. Vingerhoeds, “A fault mode identification methodology based on self-organizing map,” *Neural Computing and Applications*, vol. 32, pp. 13 405–13 423, 9 2020.
- [138] B. Zhong, J. Wang, H. Wu, J. Zhou, and Q. Jin, “Som-based visualization monitoring and fault diagnosis for chemical process,” *2016 Chinese Control and Decision Conference (CCDC)*, pp. 5844–5849, 5 2016.
- [139] T. Kohonen, “Essentials of the self-organizing map,” *Neural Networks*, vol. 37, pp. 52–65, 1 2013.
- [140] S. Qiu, X. Cui, Z. Ping, N. Shan, Z. Li, X. Bao, and X. Xu, “Deep learning techniques in intelligent fault diagnosis and prognosis for industrial systems: A review,” *Sensors*, vol. 23, p. 1305, 1 2023.
- [141] A. Botalb, M. Moinuddin, U. Al-Saggaf, and S. S. Ali, “Contrasting convolutional neural network (cnn) with multi-layer perceptron (mlp) for big data analysis,” in *2018 International conference on intelligent and advanced system (ICIAS)*. IEEE, 2018, pp. 1–5.
- [142] Z. Zhou, H. Yu, L. Yang, Z. Li, and C. Wen, “Fault diagnosis method for chemical processes based on variable correlation-guided convolutional neural networks,” *The Canadian Journal of Chemical Engineering*, 3 2025.
- [143] P. Patro and K. Netti, “Comparison of deep learning models (cnn and dnn) for multivariate time series dataset,” in *International Conference on Information and Communication Technology for Intelligent Systems*. Springer, 2024, pp. 249–261.
- [144] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, “Review of deep learning: concepts, cnn architectures, challenges, applications, future directions,” *Journal of Big Data*, vol. 8, p. 53, 3 2021.

- [145] Z. Zhu, Y. Lei, G. Qi, Y. Chai, N. Mazur, Y. An, and X. Huang, “A review of the application of deep learning in intelligent fault diagnosis of rotating machinery,” *Measurement*, vol. 206, p. 112346, 1 2023.
- [146] F. Zhang, N. Saeed, and P. Sadeghian, “Deep learning in fault detection and diagnosis of building hvac systems: A systematic review with meta analysis,” *Energy and AI*, vol. 12, p. 100235, 4 2023.
- [147] M. Ahang, T. Charter, O. Ogunfowora, M. Khadivi, M. Abbasi, and H. Najjaran, “Intelligent condition monitoring of industrial plants: An overview of methodologies and uncertainty management strategies,” *arXiv preprint arXiv:2401.10266*, 2024.
- [148] O. A. M. López, A. M. López, and J. Crossa, *Convolutional Neural Networks*. Springer International Publishing, 2022, pp. 533–577. [Online]. Available: https://doi.org/10.1007/978-3-030-89010-0_13
- [149] Z. Zhao, T. Li, J. Wu, C. Sun, S. Wang, R. Yan, and X. Chen, “Deep learning algorithms for rotating machinery intelligent diagnosis: An open source benchmark study,” *ISA Transactions*, vol. 107, pp. 224–255, 12 2020.
- [150] D. D. Patil, S. Bindu *et al.*, “Arc fault detection in dc microgrid using deep neural network,” in *2021 4th Biennial International Conference on Nascent Technologies in Engineering (ICNTE)*. IEEE, 2021, pp. 1–6.
- [151] A. K. Ozcanli, F. Yaprakdal, and M. Baysal, “Deep learning methods and applications for electrical power systems: A comprehensive review,” *International Journal of Energy Research*, vol. 44, pp. 7136–7157, 7 2020.
- [152] O. A. Montesinos López, A. Montesinos López, and J. Crossa, *Fundamentals of Artificial Neural Networks and Deep Learning*. Cham: Springer International Publishing, 2022, pp. 379–425. [Online]. Available: https://doi.org/10.1007/978-3-030-89010-0_10
- [153] J. Patterson and A. Gibson, *Deep learning: A practitioner’s approach*. ” O’Reilly Media, Inc.”, 2017.
- [154] J. Thanaki, *Python natural language processing*. Packt Publishing Ltd, 2017.
- [155] A. I. Georgevici and M. Terblanche, “Neural networks and deep learning: a brief introduction,” *Intensive Care Medicine*, vol. 45, pp. 712–714, 2019. [Online]. Available: <https://doi.org/10.1007/s00134-019-05537-w>
- [156] J. A. Yacim and D. G. B. Boshoff, “Impact of artificial neural networks training algorithms on accurate prediction of property values,” *Journal of Real Estate Research*, vol. 40, pp. 375–418, 7 2018.

- [157] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” *Proceedings of the IEEE*, vol. 105, pp. 2295–2329, 12 2017.
- [158] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, “Activation functions in deep learning: A comprehensive survey and benchmark,” *Neurocomputing*, vol. 503, pp. 92–108, 9 2022.
- [159] O. Pantalé, “Comparing activation functions in machine learning for finite element simulations in thermomechanical forming,” *Algorithms*, vol. 16, no. 12, p. 537, 2023.
- [160] S.-C. Huang and T.-H. Le, *Neural networks*. Elsevier, 2021, pp. 27–55.
- [161] M. A. Nielsen, *Neural networks and deep learning*. Determination press San Francisco, CA, USA, 2015, vol. 25.
- [162] H. Raj, A. Kumar, A. Kodipalli, and T. Rao, “Machine failure detection using artificial neural networks and a comparison of various optimizers,” in *2024 4th Asian Conference on Innovation in Technology (ASIANCON)*. IEEE, 2024, pp. 1–7.
- [163] A. Omar, F. Mohamed, M. Mohammed, and B. Fouad, “The commonly used algorithms to optimize a neural network in supervised learning: overview, and comparative study,” in *2021 International conference on digital age & technological advances for sustainable development (ICDATA)*. IEEE, 2021, pp. 31–38.
- [164] R. Zaheer and H. Shaziya, “A study of the optimization algorithms in deep learning,” in *2019 third international conference on inventive systems and control (ICISC)*. IEEE, 2019, pp. 536–539.
- [165] R. Moradi, R. Berangi, and B. Minaei, “A survey of regularization strategies for deep models,” *Artificial Intelligence Review*, vol. 53, pp. 3947–3986, 8 2020.
- [166] I. Salehin and D.-K. Kang, “A review on dropout regularization approaches for deep neural networks within the scholarly domain,” *Electronics*, vol. 12, p. 3106, 7 2023.
- [167] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, pp. 1929–1958, 2014.
- [168] M. R. Bhuiyan and J. Uddin, “Deep transfer learning models for industrial fault diagnosis using vibration and acoustic sensors data: A review,” *Vibration*, vol. 6, pp. 218–238, 2 2023.
- [169] K. O’shea and R. Nash, “An introduction to convolutional neural networks,” *arXiv preprint arXiv:1511.08458*, 2015.

- [170] M. Vakalopoulou, S. Christodoulidis, N. Burgos, O. Colliot, and V. Lepetit, *Deep Learning: Basics and Convolutional Neural Networks (CNNs)*. Springer US, 2023, pp. 77–115. [Online]. Available: https://doi.org/10.1007/978-1-0716-3195-9_3
- [171] H. Izadkhah, *Medical image processing: an insight to convolutional neural networks*. Elsevier, 2022, pp. 175–213.
- [172] A. Ajit, K. Acharya, and A. Samanta, “A review of convolutional neural networks,” in *2020 international conference on emerging trends in information technology and engineering (ic-ETITE)*. IEEE, 2020, pp. 1–5.
- [173] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights into Imaging*, vol. 9, pp. 611–629, 8 2018.
- [174] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *arXiv preprint arXiv:1603.07285*, 2016.
- [175] Y. Zhu, H. chen, W. Meng, Q. Xiong, and Y. Li, “A wide kernel cnn-lstm-based transfer learning method with domain adaptability for rolling bearing fault diagnosis with a small dataset,” *Advances in Mechanical Engineering*, vol. 14, p. 168781322211357, 11 2022.
- [176] J. Yopez and S.-B. Ko, “Stride 2 1-d, 2-d, and 3-d winograd for convolutional neural networks,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, pp. 853–863, 4 2020.
- [177] S. Han, P. Niu, S. Luo, Y. Li, D. Zhen, G. Feng, and S. Sun, “A novel deep convolutional neural network combining global feature extraction and detailed feature extraction for bearing compound fault diagnosis,” *Sensors*, vol. 23, p. 8060, 9 2023.
- [178] D. Ruan, J. Wang, J. Yan, and C. Gühmann, “Cnn parameter design based on fault signal analysis and its application in bearing fault diagnosis,” *Advanced Engineering Informatics*, vol. 55, p. 101877, 1 2023.
- [179] M. Krichen, “Convolutional neural networks: A survey,” *Computers*, vol. 12, p. 151, 7 2023.
- [180] M. M. Taye, “Theoretical understanding of convolutional neural network: Concepts, architectures, applications, future directions,” *Computation*, vol. 11, p. 52, 3 2023.
- [181] A. Patil and M. Rane, “Convolutional neural networks: an overview and its applications in pattern recognition,” *Information and Communication Technology for Intelligent Systems: Proceedings of ICTIS 2020, Volume 1*, pp. 21–30, 2021.
- [182] H. Gholamalinezhad and H. Khosravi, “Pooling methods in deep neural networks, a review,” *arXiv preprint arXiv:2009.07485*, 2020.

- [183] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu, “Towards better analysis of deep convolutional neural networks,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, pp. 91–100, 1 2017.
- [184] A. Zafar, M. Aamir, N. M. Nawi, A. Arshad, S. Riaz, A. Alruban, A. K. Dutta, and S. Almotairi, “A comparison of pooling methods for convolutional neural networks,” *Applied Sciences*, vol. 12, p. 8643, 8 2022.
- [185] M. Sahu and R. Dash, “A survey on deep learning: convolution neural network (cnn),” in *Intelligent and Cloud Computing: Proceedings of ICICC 2019, Volume 2*. Springer, 2021, pp. 317–325.
- [186] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, and A. Boulesteix, “Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 13, p. e1484, 2023.
- [187] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.
- [188] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko, “A gentle introduction to graph neural networks,” *Distill*, vol. 6, no. 9, p. e33, 2021.
- [189] Z. Liu and J. Zhou, *Introduction to Graph Neural Networks*. Springer International Publishing, 2020.
- [190] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, pp. 4–24, 1 2021.
- [191] M. Procaccini, A. Sahebi, and R. Giorgi, “A survey of graph convolutional networks (gcns) in fpga-based accelerators,” *Journal of Big Data*, vol. 11, p. 163, 2024. [Online]. Available: <https://doi.org/10.1186/s40537-024-01022-4>
- [192] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, “Graph neural networks: A review of methods and applications,” *AI Open*, vol. 1, pp. 57–81, 2020.
- [193] F. Liang, C. Qian, W. Yu, D. Griffith, and N. Golmie, “Survey of graph neural networks and applications,” *Wireless Communications and Mobile Computing*, vol. 2022, pp. 1–18, 7 2022.
- [194] B. Nica, *A brief introduction to spectral graph theory*. European Mathematical Society Zürich, 2018, vol. 3.

- [195] O. Wieder, S. Kohlbacher, M. Kuenemann, A. Garon, P. Ducrot, T. Seidel, and T. Langer, “A compact review of molecular property prediction with graph neural networks,” *Drug Discovery Today: Technologies*, vol. 37, pp. 1–12, 12 2020.
- [196] N. Heidari, L. Hedegaard, and A. Iosifidis, *Graph convolutional networks*. Elsevier, 2022, pp. 71–99.
- [197] A. M. Schweidtmann, J. G. Rittig, J. M. Weber, M. Grohe, M. Dahmen, K. Leonhard, and A. Mitsos, “Physical pooling functions in graph neural networks for molecular property prediction,” *Computers & Chemical Engineering*, vol. 172, p. 108202, 4 2023.
- [198] M. Fey and J. E. Lenssen, “Fast graph representation learning with pytorch geometric,” *arXiv preprint arXiv:1903.02428*, 2019.
- [199] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.
- [200] Z. Sun, C. Wang, W. Hu, M. Chen, J. Dai, W. Zhang, and Y. Qu, “Knowledge graph alignment network with gated multi-hop neighborhood aggregation,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 01, 2020, pp. 222–229.
- [201] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” *Advances in neural information processing systems*, vol. 29, 2016.
- [202] A. Hariri, Álvaro Arroyo, A. Gravina, M. Eliasof, C.-B. Schönlieb, D. Bacciu, K. Azizzadenesheli, X. Dong, and P. Vandergheynst, “Return of chebnet: Understanding and improving an overlooked gnn on long range tasks,” *arXiv preprint arXiv:2506.07624*, 2025.
- [203] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *arXiv preprint arXiv:1710.10903*, 2017.
- [204] A. de Santana Correia and E. L. Colombini, “Attention, please! a survey of neural attention models in deep learning,” *Artificial Intelligence Review*, vol. 55, no. 8, pp. 6037–6124, 2022.
- [205] F. Errica, M. Podda, D. Bacciu, and A. Micheli, “A fair comparison of graph neural networks for graph classification,” *arXiv preprint arXiv:1912.09893*, 2019.
- [206] S. Mahadevan and S. L. Shah, “Fault detection and diagnosis in process data using one-class support vector machines,” *Journal of Process Control*, vol. 19, pp. 1627–1639, 12 2009.
- [207] C. Jing and J. Hou, “Svm and pca based fault classification approaches for complicated industrial process,” *Neurocomputing*, vol. 167, pp. 636–642, 11 2015.

- [208] M. N. Nashalji, S. Arvand, and M. Norouzifard, "Integration of principal component analysis and neural classifier for fault detection and diagnosis of tennessee eastman process," in *2014 4th International Conference on Engineering Technology and Technopreneuship (ICE2T)*. IEEE, 2014, pp. 166–170.
- [209] M. B. Shams, H. Budman, and T. Duever, "Fault detection, identification and diagnosis using cusum based pca," *Chemical Engineering Science*, vol. 66, no. 20, pp. 4488–4498, 2011.
- [210] M. Kano, S. Tanaka, S. Hasebe, I. Hashimoto, and H. Ohno, "Monitoring independent components for fault detection," *AIChE Journal*, vol. 49, pp. 969–976, 4 2003.
- [211] F. Harrou, M. N. Nounou, H. N. Nounou, and M. Madakyaru, "Pls-based ewma fault detection strategy for process monitoring," *Journal of Loss Prevention in the Process Industries*, vol. 36, pp. 108–119, 7 2015.
- [212] A. Bounsiar and M. G. Madden, "One-class support vector machines revisited," in *2014 International Conference on Information Science & Applications (ICISA)*. IEEE, 2014, pp. 1–4.
- [213] Z. Yin and J. Hou, "Recent advances on svm based fault diagnosis and process monitoring in complicated industrial processes," *Neurocomputing*, vol. 174, pp. 643–650, 1 2016.
- [214] W. Wolmarans, "A comparison of pca-and energy-based fault detection and isolation in a physical heated twotank process," Ph.D. dissertation, North-West University (South Africa)., 2022.
- [215] K. R. Uren, G. van Schoor, and L. Auret, "An energy-attributed graph approach for the purposes of fdi in a heated two-tank system," *IFAC-PapersOnLine*, vol. 52, pp. 159–164, 2019.
- [216] M. Aljunaid, H. Shi, and Y. Tao, "Quality-related fault detection based on improved independent component regression for non-gaussian processes," *IEEE Access*, vol. 7, pp. 158 594–158 602, 2019.
- [217] J.-M. Lee, C. Yoo, and I.-B. Lee, "Statistical process monitoring with independent component analysis," *Journal of Process Control*, vol. 14, pp. 467–485, 8 2004.
- [218] M. Ardakani, M. Askarian, A. Shokry, G. Escudero, M. Graells, and A. Espuña, *Optimal features selection for designing a fault diagnosis system*. Elsevier, 2016, vol. 38, pp. 1111–1116.
- [219] H. Zhao, S. Sun, and B. Jin, "Sequential fault diagnosis based on lstm neural network," *IEEE Access*, vol. 6, pp. 12 929–12 939, 2018.

- [220] A. Maged, S. Haridy, and H. Shen, “Explainable artificial intelligence techniques for accurate fault detection and diagnosis: A review,” *arXiv preprint arXiv:2404.11597*, 2024.
- [221] V. Hassija, V. Chamola, A. Mahapatra, A. Singal, D. Goel, K. Huang, S. Scardapane, I. Spinelli, M. Mahmud, and A. Hussain, “Interpreting black-box models: A review on explainable artificial intelligence,” *Cognitive Computation*, vol. 16, pp. 45–74, 1 2024.

Appendix A

Tennessee Eastman Process variables

The manipulated and measurable variables within the TEP are summarised in Table A.1 and Table A.2, respectively [64]. Note that the four reactants are indicated as A, C, D, and E, the two products as G, and H, the inert product as B, and the byproduct as F. Furthermore, the last manipulated variable (agitator speed) remains constant during the operation of the TEP and is usually ignored, resulting in a total of 52 variables (41 measurable variables and 11 manipulated variables).

Table A.1: Manipulated variables within the Tennessee Eastman process.

Variable number	Description	Units
1	D feed flow	kg/h
2	E feed flow	kg/h
3	A feed flow	kscmh ^a
4	A and C feed flow	kscmh
5	Compressor recycle valve	%
6	Purge valve	%
7	Separator pot liquid flow	m ³ /h
8	Stripper liquid product flow	m ³ /h
9	Stripper steam valve	%
10	Reactor cooling water flow	m ³ /h
11	Condenser cooling water flow	m ³ /h
12	Agitator speed	rpm

^a kscmh = kilo standard cubic meters per hour

Table A.2: Measured variables within the Tennessee Eastman process.

Variable number	Description	Units
1	A feed	kscmh
2	D feed	kg/h
3	E feed	kg/h
4	A and C feed	kscmh
5	Recycle flow	kscmh
6	Reactor feed rate	kscmh
7	Reactor pressure	kPa gauge
8	Reactor level	%
9	Reactor temperature	°C
10	Purge rate	kscmh
11	Product separator temperature	°C
12	Product separator level	%
13	Product separator pressure	kPa gauge
14	Product separator underflow	m ³ /h
15	Stripper level	%
16	Stripper pressure	kPa gauge
17	Stripper underflow	m ³ /h
18	Stripper temperature	°C
19	Stripper steam flow	kg/h
20	Compressor work	kW
21	Reactor cooling water outlet temperature	°C
22	Separator cooling water outlet temperature	°C
23	Reactor feed analysis (Component A)	mol%
24	Reactor feed analysis (Component B)	mol%
25	Reactor feed analysis (Component C)	mol%
26	Reactor feed analysis (Component D)	mol%
27	Reactor feed analysis (Component E)	mol%
28	Reactor feed analysis (Component F)	mol%
29	Purge gas analysis (Component A)	mol%
30	Purge gas analysis (Component B)	mol%
31	Purge gas analysis (Component C)	mol%
32	Purge gas analysis (Component D)	mol%
33	Purge gas analysis (Component E)	mol%
34	Purge gas analysis (Component F)	mol%
35	Purge gas analysis (Component G)	mol%
36	Purge gas analysis (Component H)	mol%
37	Product analysis (Component D)	mol%
38	Product analysis (Component E)	mol%
39	Product analysis (Component F)	mol%
40	Product analysis (Component G)	mol%
41	Product analysis (Component H)	mol%

Appendix B

Exploratory data analysis

Links to the results of the EDA techniques can be found in the following subsections. Furthermore, each subsection briefly describes how to interpret the results obtained. Note that the EDA results were obtained using 48-hour simulation runs, where the respective faults were introduced after 8 hours. In addition, the 48-hour simulation runs were sampled at 3-minute intervals.

B.1 Principal component analysis results

The resulting PCA plots for both the cost and node signature matrices data for all the faults can be found by following the link listed below. Note for the PCA plots that the blue dots represent normal samples, while the red dots represent faulty samples. Normal samples refer to samples where the system was still operating normally, and faulty samples refer to when a specific fault was induced and present within the system.

<https://doi.org/10.25388/nwu.c.7977635>

B.2 Distance matrices results

The distance matrices for both the cost and node signature matrices data for all faults can be found by navigating to the link listed below. Note that in the distance matrices lighter colours represent larger distances between data points, while darker colours represent smaller distances between data points. Lastly, note that the axes of the distance matrices correspond to the sample numbers of the data used.

<https://doi.org/10.25388/nwu.c.7977647>

B.3 Self-organising map results

The output of the self-organising maps for the cost and node signature matrices data for all the faults can be found by using the link listed below. Note that the colour legend accompanying the output of

each self-organising map represents how closely the weights of a neuron correspond to the weights of neighbouring neurons. Darker colours indicate that the neuron's weight differs greatly from the weights of neighbouring neurons, and lighter colours indicate that the neuron's weight more closely resembles the weights of neighbouring neurons.

<https://doi.org/10.25388/nwu.c.7977653>

Appendix C

Machine learning for the cost matrix data

The optimal hyperparameters of the traditional non-graphical machine learning techniques, obtained with hyperparameter tuning, are summarised in the subsequent tables. With hyperparameter tuning it was also discovered that the best optimizer to use with each model is the ADAM optimizer. Note that all models used the ReLU activation function.

Hyperparameter tuning was also used to identify the optimal type of dataset to use with each model during training. The types of datasets can be mainly divided into normalised or standardised datasets. Furthermore, the scaling constants used when normalising or standardising the datasets could be determined using either the entire data set or only the NOC data. Consequently, hyperparameter tuning was used to determine whether the data should be normalised or standardised, as well as how the scaling constants should be calculated.

Note that for the one-dimensional CNN, the search space for the optimal type of dataset was enlarged by considering whether the cost matrix should be flattened row-wise or column-wise. Flattening the cost matrix row-wise or column-wise would affect the relative positions of entries within the resulting vector, thereby also influencing the performance of the one-dimensional which depends on relative position of entries. Consequently, for the one-dimensional CNN dataset, it is indicated whether the cost matrix was flattened row-wise or column-wise.

Table C.1: Batch sizes used for the traditional non-graphical machine learning technique for the cost matrix data.

Model	Batch size
DNN	64
CNN 1D	32
CNN 2D	32

Table C.2: Learning rates used for the traditional non-graphical machine learning technique for the cost matrix data.

Model	Learning rate
DNN	0.00040
CNN 1D	0.00040
CNN 2D	0.00025

Table C.3: Datasets used for the traditional non-graphical machine learning technique for the cost matrix data.

Model	Dataset
DNN	Standardized all
CNN 1D	Normalized NOC (column)
CNN 2D	Standardized all

The Dropout values used in the layers of the different machine learning techniques are summarised in the subsequent tables. Readers are referred to Table 4.2, to see the positions of the different layers, mentioned in the subsequent tables, with respect to their model architectures.

Table C.4: Dropout values used for the layers of the DNN.

Layer	Dropout value
FC(425)	0.323
FC(375)	0.252
FC(300)	0.235
FC(225)	0.209

Table C.5: Dropout values used for the layers of the 1D CNN.

Layer	Dropout value
FC(225)	0.233
FC(225)	0.336

Table C.6: Dropout values used for the layers of the 2D CNN.

Layer	Dropout value
FC(175)	0.250

Appendix D

Machine learning for the node signature matrix data

The optimal hyperparameters of the graphical machine learning techniques, that were not mentioned in Section 5.4, are summarised in Tables D.1 - D.6. With hyperparameter tuning it was also discovered that the best optimizer to use with each model was the ADAM optimizer. Note that all models used the ReLU activation function.

Hyperparameter tuning was also used to identify the optimal type of dataset to use with each model during training. The types of datasets can be mainly divided into normalised or standardised datasets. Furthermore, the scaling constants used when normalising or standardising the datasets could be determined using either the entire data set or only the NOC data. Consequently, hyperparameter tuning was used to determine whether the data should be normalised or standardised, as well as how the scaling constants should be calculated.

Table D.1: Batch sizes used for the graphical machine learning technique for the node signature matrix data.

Model	Batch size
GCN-Net	128
ChebConv-Net	64
GATConv-Net	32

Table D.2: Learning rates used for the graphical machine learning technique for the node signature matrix data.

Model	Learning rate
GCN-Net	0.00187
ChebConv-Net	0.00110
GATConv-Net	0.00063

Table D.3: Datasets used for the graphical machine learning technique for the node signature matrix data.

Model	Dataset
GCN-Net	Standardized all
ChebConv-Net	Standardized all
GATConv-Net	Standardized all

The Dropout values used in the layers of the different graphical machine learning techniques are summarised in the subsequent tables. Readers are referred to Table 5.2, to see the positions of the different layers, mentioned in the subsequent tables, with respect to their model architectures.

Table D.4: Dropout values used in the layers of the GCN-Net.

Layer	Dropout value
FC(500)	0.484

Table D.5: Dropout values used in the ChebConv-Net.

Layer	Dropout value
FC(500)	0.258
FC(200)	0.215

Table D.6: Dropout values used in the GATConv-Net.

Layer	Dropout value
FC(500)	0.2375
FC(425)	0.4837