

Data-Driven Diagnosis for Cyber-Physical Systems

Von der Fakultät für Maschinenbau und Bauingenieurwesen der
Helmut-Schmidt-Universität / Universität der Bundeswehr Hamburg zur
Erlangung des akademischen Grades eines Doktor-Ingenieurs genehmigte

DISSERTATION

von

Henrik Sebastian Steude

aus Kiel

Hamburg 2025

Erstgutachter:

Prof. Dr. rer. nat. Oliver Niggemann

Helmut-Schmidt-Universität / Universität der Bundeswehr Hamburg

22043 Hamburg

Zweitgutachter:

Prof. Dr.-Ing. Alexander Fay

Ruhr-Universität Bochum

44780 Bochum

Tag der mündlichen Prüfung: 24.09.2025

Abstract

Rapid recovery from failures in safety-critical systems requires accurate and timely diagnosis, which is a task that is increasingly challenging due to the growing complexity of modern cyber-physical systems. These systems generate large amounts of data describing operational metrics, sensor readings, and performance indicators across various subsystems. The complexity is also driven by various discrete system modes, complex interactions between subsystems, and external influence factors. The example of such a complex system that has motivated this work is the Environmental Control and Life Support System of the International Space Station's Columbus module. Due to this volume of data, in this particular case thousands of signals, differentiating nominal from abnormal system states alone becomes a non-trivial task. However, the challenge of identifying root causes of abnormal behavior proves to be even more complex and often requires system models whose creation demands substantial domain expertise.

This thesis presents a novel approach to these fault diagnosis challenges that minimizes the need for extensive prior knowledge. The research focuses on developing a method that detects anomalies at the subsystem level, which is an aggregated level between individual sensors and the overall system, and leverages the detected anomalies for system diagnosis.

The main contributions of this thesis are as follows: *(i)* A novel neural network architecture that was specifically designed for detecting anomalies in subsystems of cyber-physical systems. *(ii)* A new graph-based diagnostic algorithm which uses basic causal relationships between subsystems and the output of the anomaly detection model to identify root causes of failures. *(iii)* A methodology that combines the components above into a comprehensive diagnostic framework. *(iv)* An implementation of these methods using state-of-the-art machine learning operations tools, that addresses the challenges typically involved in deploying and maintaining machine learning models in production environments.

The proposed framework was evaluated through a set of experiments using both simulated and real-world datasets. The results provide evidence that the proposed approach can identify subsystem-level anomalies and find the subsystems that caused the system failure.

This thesis makes contributions to the fields of anomaly detection, fault diagnosis, and machine learning operations in the context of cyber-physical systems. It provides a largely data-driven solution for the diagnosis challenge in complex technical systems, which traditionally requires extensive manual and labor-intensive modeling. The findings have implications for various domains including aerospace, manufacturing, and other critical infrastructure systems.

Acknowledgements

Dear reader, I would first like to thank you for taking the time to read this thesis. I hope you will find something valuable for your own work in these pages.

Writing this thesis was only possible thanks to the tremendous support of my doctoral supervisor, Prof. Dr. rer. nat. Oliver Niggemann, who not only accompanied me through the sometimes difficult path through the PhD studies, but also contributed significantly to the scientific results of this work through regular technical discussions, reviews, and ideas.

From the start of working on the topics in this thesis, I have learned a lot and am grateful for the opportunity to do so in such a friendly and motivated environment. My sincere thanks therefore also go to my colleagues who created this atmosphere and, in particular, to the co-authors of the publications related to this work. This does not only include the staff at the Professorship Computer Science in Mechanical Engineering at the Helmut Schmidt University, but also the entire team of the (K)ISS project—the project that motivated this research. In this regard, I would like to mention Lukas Moddemann, Dr. rer. nat. Christian Geier, Dr. rer. nat. Alexander Diedrich, and Alexander Windmann, who played a significant role in helping me learn so much and truly enjoy the work in research and the (K)ISS project.

Finally, these lines would not have been written without the wonderful support of my wife Martina Steude. She not only challenged this work and all the papers with her non-technical perspective and acted as a rubber duck during programming, but also supported me throughout the whole process, especially during the months when I withdrew from social activities in order to finalize this thesis. Thank you! Last but not least, I must not forget to mention my one-year-old dog Lotta, who made sure I got the fresh air and exercise I needed during the hottest phases of the writing process. You deserve a treat.

Contents

List of Figures	xii
List of Tables	xiii
List of Algorithms	xv
Acronyms	xvii
List of Mathematical Symbols	xix
1 Introduction	1
1.1 Motivation	1
1.1.1 Anomaly Detection	3
1.1.2 Diagnosis	3
1.1.3 Combining Anomaly Detection and Diagnosis	4
1.1.4 Machine Learning Operations	5
1.2 Research Questions	6
1.3 Contributions	9
1.4 Outline of this Thesis	10
1.5 List of Publications	11
I Background	15
2 Foundations	17
2.1 Time Series Analysis with Neural Networks	17
2.1.1 Convolutional Neural Networks Overview	18
2.1.2 Temporal Convolutional Networks (TCNs)	19
2.1.3 Autoencoders for Dimensionality Reduction	22
2.1.4 Autoencoders for Anomaly Detection	23
2.1.5 Variational Autoencoders for Anomaly Detection	23
2.2 Graph-Based Algorithms for Diagnostic Reasoning	27
2.2.1 Fundamentals of Graph Theory	27
2.2.2 Graph Algorithms	28
2.3 Machine Learning Operations (MLOps)	29
2.3.1 The ML Development Lifecycle and CRISP-DM	29
2.3.2 Containerization and Cloud-Native Infrastructure	32

3	State of the Art	33
3.1	Anomaly Detection in Cyber-Physical Systems	33
3.1.1	Modern Anomaly Detection Methods	33
3.1.2	Subsystem-Level and Structurally-Informed Approaches	36
3.2	Machine Learning Approaches for Diagnosis in Cyber-Physical Systems	37
3.2.1	Supervised Learning for Fault Classification	38
3.2.2	Diagnostic Insights from Unsupervised Anomaly Detection Models	38
3.2.3	Integrating Data-driven Methods with Diagnostic Reasoning	39
3.2.4	The Role of Abstraction in Diagnostic Methods	41
3.3	State of the Art in MLOps	41
3.3.1	Evolution of MLOps Frameworks and Platforms	41
3.3.2	Cloud-Native Infrastructure and MLOps Platforms	42
3.3.3	Challenges for MLOps in Diagnostic Applications	43
3.4	Research Gaps and Opportunities	43
II	Formal Definition and Solution	45
4	Formal Definition	47
4.1	Basic Notation and Definitions	47
4.2	Problem Statement	48
4.2.1	Required Inputs	48
4.2.2	Expected Outputs	49
4.3	Relation to Research Questions	50
5	Solution	53
5.1	Overview	53
5.2	Symptoms Generator	55
5.2.1	Neural Network Architecture	55
5.2.2	Residual Binarizer	58
5.3	Graph Diagnosis Algorithm	60
5.3.1	Motivation and Background	60
5.3.2	Algorithm Design Principles and Evaluation Criteria	60
5.3.3	Algorithm Description	63
5.3.4	Computational Complexity and Trade-offs	63
5.4	Machine Learning Operations Implementation	66
5.4.1	Implementation Strategy	66
5.4.2	MLOps Components and Workflow Integration	67
5.4.3	Implementation Challenges and Solutions	69

III	Evaluation	71
6	Experimental Results	73
6.1	Evaluation Criteria & Hypotheses	73
6.2	Evaluation of the Symptoms Generator	74
6.2.1	Experimental Setup	74
6.2.2	Results and Analysis	77
6.3	Evaluation of the Graph Diagnosis Algorithm	79
6.3.1	Tennessee Eastman Process	80
6.3.2	Controlled Graph Scenarios	81
6.4	Evaluation of the Integrated Diagnosis Framework	83
6.4.1	Evaluation with the SWaT Dataset	83
6.4.2	Randomly generated Dynamical Systems	86
6.5	Evaluation of the MLOps Implementation	90
6.5.1	Use Case	90
6.5.2	Pipeline Implementation and Integration	90
7	Theoretical Results	93
7.1	Soundness and Completeness	93
7.1.1	Soundness Analysis	93
7.1.2	Completeness Considerations	94
7.2	Complexity Analysis	94
8	Discussion	97
8.1	Subsystem-Level Anomaly Detection Architecture	97
8.2	Graph-Based Diagnostic Algorithm	98
8.3	Integrated Diagnostic Framework	98
8.4	MLOps Implementation	99
IV	Conclusion and Outlook	101
9	Conclusion	103
9.1	Addressing the Research Questions	103
9.2	Key Strengths	104
9.3	Limitations	105
10	Outlook	107
10.1	Advancing the Subsystem-Level Anomaly Detection Architecture	107
10.2	Enhancing the Graph-Based Diagnostic Algorithm	107
10.3	MLOps Advancements	108
10.4	Long-Term Vision	108
	Bibliography	111

Contents

Curriculum Vitae

131

List of Figures

1.1	Possible fault detection and resolution workflow in cyber-physical systems	2
1.2	System hierarchy illustrating subsystem and observation levels	7
1.3	Illustrative fault propagation graph for a hydraulic system.	8
2.1	Illustration of a multi-channel one-dimensional convolution.	19
2.2	Visualization of dilated causal convolutions	20
2.3	Neural architecture overview of a Temporal Convolutional Network.	21
2.4	Basic architecture of an autoencoder with a sample of the Modified National Institute of Standards and Technology dataset	22
2.5	Architecture of a Variational Autoencoder illustrated with an MNIST digit sample	24
2.6	Comparison of latent spaces in Autoencoder and Variational Autoencoder	26
2.7	Different representations of a directed cyclic graph	28
2.8	The Cross-Industry Standard Process for Data Mining methodology and corresponding Machine Learning Operations features	31
4.1	System hierarchy including subsystems and signals	48
5.1	Overview of the three phases of the proposed diagnosis framework	54
5.2	Neural network architecture with composite latent space	56
5.3	Detailed Temporal Convolutional Network-Variational Autoencoder architecture	57
5.4	Example fault propagation graph with symptomatic subsystems	61
5.5	Overview of the Machine Learning Operations reference implementation	68
6.1	Healthy data sample with causal and derived signals	75
6.2	Different approaches to modeling subsystem-level anomalies	76
6.3	Distribution of anomaly scores across different fault scenarios	79
6.4	Tennessee Eastman Process causal subsystem graph	80
6.5	Evaluation of the graph diagnosis algorithm	82
6.6	SWaT process diagram	84
6.7	Time series data generation process for random dynamic system experiment	87
6.8	Experimental trial for random dynamical system experiment	88
6.9	Result summary of the random dynamic system experiment	89

List of Figures

6.10 Microservice architecture for the diagnostic system 92

List of Tables

6.1	Label allocation in the test set.	77
6.2	Anomaly Detection Performance Across Subsystems.	78
6.3	Comparison of true causal subsystems, observed symptomatic subsystems, and predicted root causes for the four fault scenarios of the Tennessee Eastman Process that were described in detail by Chiang et al. [30].	81
6.4	Attacked subsystems (ground truth), detected symptoms, and diagnosed root cause candidates for the SWaT experiment.	85

List of Algorithms

1	Overall Diagnostic Process (phases b and c only)	55
2	Graph Diagnosis Algorithm	65

Acronyms

AE	Autoencoder
AI	Artificial Intelligence
API	Application Programming Interface
AutoML	Automated Machine Learning
BFS	Breadth-First Search
CBD	Consistency-Based Diagnosis
CFA	Cabin Fan Assembly
CHX	Condensate Heat Exchangers
CNN	Convolutional Neural Network
CPS	Cyber-Physical System
CPU	Central Processing Unit
CRISP-DM	Cross-Industry Standard Process for Data Mining
DAG	Directed Acyclic Graph
DDP	Distributed Data Parallel
DevOps	Development and Operations
DFS	Depth-First Search
DL	Deep Learning
ECLSS	Environmental Control and Life Support System
ELBO	Evidence Lower Bound
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
GNN	Graph Neural Network
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
IoT	Internet of Things
ISFA	Inter-module Ventilation Supply Fan Assembly
ISS	International Space Station
KFP	Kubeflow Pipelines
KL	Kullback-Leibler
LLM	Large Language Model
LSTM	Long Short-Term Memory
ML	Machine Learning
MLOps	Machine Learning Operations
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NN	Neural Network

Acronyms

PCA	Principal Component Analysis
PINN	Physics-Informed Neural Network
ReLU	Rectified Linear Unit
RIM	Recurrent Independent Mechanisms
RNN	Recurrent Neural Network
SDK	Software Development Kit
TCN	Temporal Convolutional Network
TCS	Thermal Control System
TCV	Temperature Control Valves
TEP	Tennessee Eastman Process
VAE	Variational Autoencoder

List of Mathematical Symbols

Basic Mathematical Notation

x, y, z, t, \dots	Scalar quantities, represented in non-bold notation
$\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{w}, \dots$	Vectors, denoted by bold lowercase letters
$\mathbf{X}, \mathbf{Y}, \mathbf{W}, \dots$	Matrices, denoted by bold uppercase letters
u, v, \dots	Typical vertex identifiers used in graph theory
A, B, S, P, \dots	Sets, represented by non-bold uppercase letters
$\hat{x}, \hat{\mathbf{x}}, \hat{\mathbf{X}}, \dots$	Model predictions or estimates, denoted by hat notation
i, j, k, l, m, \dots	Index variables typically used for iteration, counting, or referencing elements in vectors, sequences or matrices
\mathbb{R}	Set of real numbers
\mathbb{N}	Set of natural numbers
f, g, h, \dots	General functions or mappings
\sum	Summation operator
\prod	Product operator
\int	Integration operator
max, min	Maximum and minimum functions
\odot	Hadamard product (element-wise product of two matrices)
$\mathbb{1}_A(x)$	Indicator function (1 if $x \in A$, 0 otherwise)
\mathbf{I}	Identity matrix
$O(\dots)$	Big-O notation
$p(x)$	Probability distribution
$q(x)$	Approximate or variational probability distributions
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and variance σ^2
$D_{KL}(p\ q)$	Kullback-Leibler divergence between distributions p and q
σ^2	Variance parameter
ϵ	Random noise variable, sampled from a standard normal distribution

Neural Networks and Deep Learning

l	Index variable denoting a specific layer in a neural network
L_{NN}	Number of layers in a neural network
C_{in}	Number of input channels in a convolutional layer

List of Mathematical Symbols

C_{out}	Number of output filters/channels in a convolutional layer
L	Length of an input sequence
L_{out}	Length of an output sequence after convolution
n	Dimensionality of the input space in autoencoders
m	Dimensionality of the latent space in autoencoders
K	Kernel size in convolutional operations
d	Dilation rate in temporal convolutional networks
p	Window size in pooling operations
ρ	Receptive field size in a Temporal Convolutional Network, representing the number of input time steps that influence a single output
\mathbf{W}	Filter/kernel weights in convolutional layers
\mathbf{b}	Bias term in neural network layers
Conv	Convolution operation in neural networks
MaxPool	Max pooling operation that extracts maximum values from local regions
$\sigma(x)$	General notation for activation functions
ReLU	Rectified Linear Unit activation function, defined as $\max(0, x)$
r	Residual value or error between prediction and ground truth
E_i	Encoder network for subsystem i
D	Decoder network
\mathbf{z}_i	Latent space vector for subsystem i
\mathbf{z}_c	Composite latent space from all subsystems
$\boldsymbol{\theta}$	Trainable parameters of a neural network
$\boldsymbol{\theta}_{\text{block}}$	Trainable parameters within a residual block
ϕ	Parameters of the encoder network
\mathcal{L}	Loss function for training neural networks
MSE	Mean Squared Error, a common loss function
β	Weighting factor in β -VAE to balance reconstruction and KL divergence

CPS, Graph Theory and Diagnosis

$S = \{s_1, \dots, s_n\}$	Set of subsystems in a CPS
$P = \{p_1, \dots, p_m\}$	Set of available signals/sensors
T	Time interval for time series analysis
t	Time point within interval T
Δt_w	Window size for time series analysis
$\mathbf{X} \in \mathbb{R}^{ T \times P }$	Time series data matrix where each row corresponds to a time point and each column to a signal
\mathbf{X}_t	Matrix of sensor values in the current time window

$G = (S, E)$	Graph with node set S and edge set E
$(s_i, s_j) \in E$	Directed edge from node s_i to node s_j
V_k	Set of nodes at exactly distance k from a given source node u , forming the k -th shell in a graph traversal
shells[d]	Set of nodes at exactly distance d from a candidate node in the causal graph
shortest_path(c, u)	Function that determines the length of the shortest path between node c and node u in graph G
path($v \rightarrow s$)	Predicate that is true if there exists a directed path from node v to node s in graph G
\mathcal{G}	Function that generates health states from time series data
\mathcal{M}	Function that maps subsystems to their associated signals
\mathcal{A}	Diagnostic algorithm that identifies causal subsystems from health states and a causal graph
\mathbf{h}	Vector of health states for all subsystems (time-dependent)
$h_s \in \{0, 1\}$	Health state of subsystem s (0 for "OK", 1 for "not OK", time-dependent)
S_{sym}	Set of symptomatic subsystems (time-dependent)
S_{causal}	Set of subsystems identified as root causes of observed symptoms (time-dependent)
$S_{\text{reach}}(c, U)$	Set of symptomatic nodes reachable from candidate node c
C	Set of candidate nodes that could be potential root causes for observed symptoms
U	Set of symptomatic subsystems that remain unexplained during the diagnostic process
B	Set of candidate nodes whose score exceeds the threshold and are selected as root causes
$\sigma_r(c)$	Reachability score for candidate node c
$\sigma_d(c)$	Distance score for candidate node c
$\sigma_a(c)$	Anomaly score for candidate node c
$\sigma_c(c)$	Chain score for candidate node c
$\sigma(c)$	Total score for candidate node c
σ_{max}	Maximum score among all candidate nodes in a given iteration of the diagnostic algorithm
$\mathbf{w} = (w_1, w_2, w_3, w_4)$	Weight vector for the different evaluation criteria
θ_{thr}	Threshold parameter in the diagnosis algorithm
θ	Sensitivity parameter in the diagnostic algorithm that determines which candidate nodes are considered as root causes
$\alpha(\mathbf{x})$	Anomaly score quantifying the degree of abnormality of an observation
α_τ	Sensitivity parameter for statistical threshold determination

List of Mathematical Symbols

τ	Anomaly detection threshold applied to reconstruction error or anomaly scores
$\delta(\mathbf{x})$	Binary anomaly detection decision (1 for anomalous, 0 for normal)
$\text{paths}(c)$	The set of all directed paths in graph G that start from node c
P^t	Time-wise precision in anomaly detection, the ratio of correctly identified anomalous time points to all time points classified as anomalous
R^e	Event-wise recall in anomaly detection, the ratio of detected anomalous events to all actual anomalous events
F_1^c	Composite F1 score, the harmonic mean of time-wise precision and event-wise recall
$r_s(t)$	Reconstruction error for subsystem s at time t , quantifying the deviation between original and reconstructed signals
A	State matrix in a linear system model, representing the dynamics of the system
B	Input matrix in a linear system model, representing how control inputs affect the system states
y	State vector containing the internal states of a dynamic system
u	Control input vector affecting the dynamics of the system
s_{true}	Label for the actual causal subsystem in experimental scenarios

1 Introduction

Chapter Outline This chapter first describes the motivation for this thesis. After a brief overview of the historical development and the current state of the art in relevant research areas, the research questions are discussed in Section 1.2. Section 1.3 presents the contributions of this thesis, followed by Section 1.4, which describes the structure of the thesis. Finally, Section 1.5 contains the list of publications and places them in the context of the thesis.

1.1 Motivation

In the life cycle of most technical systems, component failures are inevitable. When such failures occur, they must be identified and remedied quickly. A timely recovery is particularly important in safety-critical applications [162] and can in many cases prevent cascading failures [3]. However, since modern Cyber-Physical Systems (CPSs) are becoming increasingly complex, fault diagnosis is a non-trivial task. CPSs are systems where computational elements are tightly integrated with physical components [86]. They are characterized by many, sometimes unobservable system variables, complex interactions between subsystems, and different system modes with discrete state transitions [84].

The Environmental Control and Life Support System (ECLSS) [136] of the Columbus module on the International Space Station (ISS) is an example of such a complex CPS. The development of an automated diagnostic system for the ECLSS motivated this work and will serve as an example application throughout this thesis. The system records more than a thousand signals and is only one of many systems installed in the Columbus module. Apart from this data volume, the ECLSS operates in various discrete system modes, such as condensate removal procedures and switching between redundant subsystems. Furthermore, the system is exposed to numerous external influences. These include, for example, those resulting from the orbital dynamics of the ISS and control variables such as temperature and humidity setpoints [178].

This example illustrates the complexity of many modern CPSs. The mere detection of anomalies, which is a necessary first step in troubleshooting [50], can therefore become a technical challenge. Despite some contributions that question the superiority of Deep Learning (DL) methods over traditional statistical approaches [5, 147, 156], recent literature reviews identify DL as the state-of-the-art for this task [61, 86, 134, 208]. These methods learn a model of the system behavior in a purely data-driven manner and identify anomalies as deviations between observations and model predictions [131].

1 Introduction

However, identifying anomalies is not enough to be able to get a system back to a healthy state. The cause of the anomalies must also be known, which turns out to be a significantly greater challenge. As shown in Figure 1.1, anomaly detection

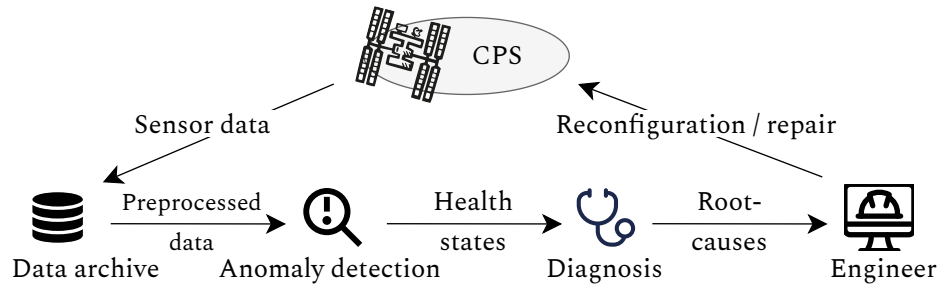


Figure 1.1: Possible fault detection and resolution workflow in CPS. The process shows how outputs of the anomaly detection step serve as input for the diagnostic reasoning step, which determines root causes that engineers can address through reconfiguration or repair actions.

is only one step in the overall diagnostic process. While it determines that something is wrong, diagnosis must determine the root cause of the problem. In current research, this diagnostic challenge is addressed by various methods, including supervised learning approaches [59, 121, 207], classical symbolic Artificial Intelligence (AI) techniques [41, 50, 142], and hybrid methods [89, 126, 145]. However, what all of these approaches have in common is that they require significantly more structured prior knowledge than purely data-driven anomaly detection, either in the form of anomaly labels or formalized system models.

In the case of the ECLSS module, for example, despite detailed error reports and maintenance tickets, no suitable labels are available for training supervised learning models for diagnosis. Therefore, the creation of training data would require manual labeling of the time series. And even if these labels were available, supervised learning models would only be able to recognize faults that have already occurred in the time period described by the training dataset, which is not acceptable for most applications, including the ECLSS example application. The development of system models for diagnostic methods based on symbolic AI, however, is also associated with considerable effort due to the system size, complexity and expertise required [17, 41].

Based on these observations, the main objective of this work is to present an automated diagnostic approach that minimizes the required prior knowledge while still providing useful diagnoses. The proposed methodology combines a novel subsystem-level anomaly detection model with a graph-based diagnostic reasoning algorithm to identify which subsystems are the main causes of observed anomalies. Motivated by the implementation of the proposed solutions for the ECLSS example system, this thesis also addresses the challenges associated with the deployment and operation of modern Machine Learning (ML) solutions in production envi-

ronments. In preparation for the research questions presented in Section 1.2, the following subsections introduce the topics of anomaly detection, diagnosis, their combination and Machine Learning Operations (MLOps).

1.1.1 Anomaly Detection

Modern CPS typically record a large number of signals that are used for continuous monitoring. For this reason, anomaly detection in the field of CPS usually means identifying unusual patterns in multivariate time series [208]. According to Hawkins [75], an anomaly refers to a deviation from a general data distribution. These deviations can be categorized as (i) point anomalies, which affect individual measurements, (ii) contextual anomalies, which are only abnormal in a specific context (such as time or place), and (iii) collective anomalies, in which the observations in a sequence together form an anomalous pattern [27]. Research into this modeling of data distributions and the corresponding identification of deviations from these distributions has been ongoing since the 1960s [71]. Since then, new methods have been continuously developed in various communities [5]. The focus of research has shifted from classical statistical and stochastic process-based methods [14] to traditional ML approaches [27], which has eventually led to today's focus on DL techniques [208].

Despite these developments, the basic concept remained unchanged during this period: historical data is used to learn a model of the normal behavior of the system, which is then used to evaluate how well new observations match this model [131]. The greater the deviation between the observed data and the model predictions, the higher the probability that a sample will be classified as anomalous. This approach offers two advantages over rule-based or supervised learning-based methods: It is completely data-driven and does not require formalized prior knowledge such as explicit rules or physics-based models of fault modes. More importantly, it can detect previously unobserved anomalies that may not be similar to any known faults. As recent reviews confirm, modern DL methods show good performance when applied to real CPS data and have now established themselves as the state-of-the-art for this task [86, 208].

However, purely data-driven and unsupervised anomaly detection methods for CPSs typically lack structural system information. The CPSs are treated either as a collection of independent signals or as a single large black box [62, 208]. Although some approaches have been proposed that incorporate system hierarchy information to improve anomaly detection [2, 48, 88, 184], there seems to be no systematic investigation on the exploitation of this information specifically for diagnostic purposes.

1.1.2 Diagnosis

The need for automated diagnostic solutions for CPSs is not only recognized in research [142], but also in industry, government agencies [21] and non-governmental

organizations [10]. As a result, a wide variety of diagnostic approaches have been developed in recent decades, particularly in the control theory and the AI community. Escobet et al. [50] categorizes these approaches into model-based methods, data-driven methods, and knowledge-based methods.

As mentioned earlier, all of these approaches require a high degree of formalized prior knowledge. Diedrich and Niggemann [41], for example, show that consistency-based diagnosis, which is a model-based method, requires descriptive models with logical expressions such as “If the pump and valve are both in a non-abnormal state, the observed flow readings should be classified as OK”. Similarly, structural analysis, which is also a model-based method, requires system models in the form of differential equations and explicit fault models [57].

Knowledge-based approaches, as their name suggests, also require extensive prior knowledge. These methods typically integrate expert knowledge in the form of ontologies, heuristics, or rules to detect and classify errors [17]. They often require fault-symptom relationships defined by domain experts as inputs [40].

Purely data-based diagnostic methods, on the other hand, appear to be more accessible, but require sufficient data representing both the nominal state and relevant fault cases. Typically, ML or DL models classify sensor data samples into specific fault classes [127, 166, 207]. However, this means that only previously observed and labeled faults can be diagnosed. Such methods therefore cannot be used in systems where novel faults occur frequently or where the training data for faults are not extensive enough or not available at all [50].

This need for prior knowledge can be a significant practical obstacle to the development of diagnostic systems, especially for large, complex CPSs such as manufacturing plants or space stations. For this reason, researchers are investigating approaches that combine pure anomaly detection methods with traditional diagnostic techniques in order to reduce the level of prior knowledge required.

1.1.3 Combining Anomaly Detection and Diagnosis

The literature on CPS contains several methods that combine anomaly detection and diagnosis. Depending on how anomaly detection and diagnosis are combined, these approaches can be divided into three main groups.

The first category includes approaches that extend anomaly detection with basic diagnostic features. Some researchers even go so far as to refer to the identification of sensors or signals in the CPS datasets that exhibit anomalies as a diagnosis [61, 209]. Following the medical metaphor, however, the author of this thesis believes that identifying signals that behave abnormally is more comparable to recognizing symptoms than to diagnosing.

The second group contains methods that first apply data-driven anomaly detection techniques and then process their outputs in various diagnostic reasoning procedures in a second step. For example, Bunte et al. [22] and Diedrich and Niggemann [41] use residues that can be calculated by data-driven methods as inputs for Consistency-Based Diagnosis (CBD) methods. Mohammadi et al. [126], on the other

hand, integrate physical structural information into recurrent neural networks to generate residuals, resulting in so-called “grey-box recurrent neural networks” that combine data-driven learning with model-based constraints. In a similar way, Fullen et al. [58] combine case-based reasoning with data-driven anomaly detection to improve diagnosis performance and reduce the effort required for knowledge acquisition and formalization. Garcia-Alvarez et al. [60] integrate model-based diagnosis with a Principal Component Analysis (PCA) to improve fault detection in systems with multiple operating points. Finally, Rehak et al. [145] use a different approach that is very similar to the one presented in this work, using the outputs of a data-driven anomaly detection as inputs for a graph-based diagnosis reasoning algorithm.

The third category includes methods that reverse the approach of the methods in the second category. They first use system models to calculate residuals, which are then analyzed in a second step using anomaly detection methods. Pesola et al. [141], for example, discuss this idea for a use case on spacecraft data and identify anomalies in residual signals. Jung et al. [90] present a similar approach by combining model-based residuals with anomaly classifiers. This method enables the identification of unseen errors and the classification of different fault cases even though only one of these fault cases is contained in the training data.

Despite great progress in these approaches, which combine anomaly detection and diagnosis methods, limitations still exist. The methods described here continue to require a high degree of prior knowledge, whether in the form of system models, causal relationships, or explicit error patterns. Furthermore, these methods focus on individual components or sensors and neglect the potential advantages of aggregation to a more abstract level, such as that of subsystems. This granular focus can be a disadvantage, especially for large, complex systems, where diagnosis at the subsystem level could significantly reduce both the complexity of the diagnostic task and the required prior knowledge. Finally, abstractions at the subsystem level are easier to align with practical maintenance strategies, where entire functional units are often replaced instead of individual components [3, 35].

1.1.4 Machine Learning Operations

Developing algorithmic solutions for automated anomaly detection and diagnosis is anything but trivial, as described above. However, the deployment and maintenance of these software systems add another significant layer of complexity. A discipline dedicated to these challenges is called MLOps. The abbreviation is a fusion of “ML” and “operations” and was created in reference to Development and Operations (DevOps), a management concept for software projects that includes concepts such as continuous delivery and infrastructure as code for classic software projects. MLOps is an extension of DevOps principles to include ML-related aspects.

As Sculley et al. [157] noted in their seminal paper 10 years ago, ML systems contain complex interactions between code, data and models that are not found in classic software projects. These interactions can lead to technical failures and system

1 Introduction

instability. For ML software in complex systems such as the ECLSS, these challenges are particularly relevant for several reasons: First, ML models often suffer from concept drift because physical systems naturally wear out and because maintenance work or reconfigurations can cause subtle changes in system behavior. Cody et al. [33] describe how maintenance measures can fundamentally change system behavior and thus pose an “existential threat of single-use models” in production environments. Second, the volume of data often poses technical challenges. Although numerous open-source frameworks such as Dask [149] and Spark [206] have been developed to simplify the distributed (pre-)processing or training on such datasets, any parallelization brings additional technical complexity. Third, the models often need to be applied in real time to CPS data streams, which requires robust real-time features of the models deployments [178]. Finally, solutions often consist of modular systems deployed in the form of microservices, which is advantageous in many scenarios but also adds complexity [103].

MLOps attempts to address these challenges with a set of principles and practices. These include automated ML pipelines that document relationships between training datasets and the resulting models, as well as continuous performance monitoring for deployed models [170]. These tools and practices not only make the creation of ML models scalable and reproducible, but also enable the fully automated re-training and re-deployment of models on new data.

At the conceptual level, the MLOps approaches in the literature have now largely converged [97]. However, the tool landscape is still very volatile and fragmented [15]. Berberi et al. [15] have identified several challenges in their recent review on MLOps. In addition to the lack of consensus on the tooling, many of the existing open-source MLOps platforms are not yet feature complete in the sense that they do not cover all steps in the lifecycle of typical ML applications. In particular, the monitoring of model performance and thus the detection of concept or data drift is often insufficiently supported, or not supported at all. However, these challenges are particularly relevant for safety-critical systems such as the ECLSS, which have strong requirements in terms of robustness.

1.2 Research Questions

As motivated in the previous section, this thesis aims to develop a diagnostic method that minimizes the need for prior knowledge while remaining capable of generating useful diagnoses for faults in complex CPSs. Therefore, the remaining chapters address the following research questions:

RQ1: How can subsystem-level health states (symptoms) suitable for diagnostic algorithms be generated by enhancing data-driven anomaly detection methods with typically available prior knowledge?

As already indicated in Subsection 1.1.3, a promising way to reduce the required prior knowledge is to increase the abstraction level of the diagnosis. This would allow carrying out the diagnostic reasoning process using the health states of entire components or subsystems rather than individual sensor signals. However, most methods require discrete observations or symptoms as inputs, that are not readily available, especially not on this aggregated level. The symptoms used as inputs by most diagnosis methods are binary indicators of whether a component or subsystem is behaving normally or abnormally, rather than continuous anomaly scores.

In the ECLSS, for example, if an issue is detected in the Cabin Fan Assembly (CFA), maintenance procedures typically involve replacing the entire assembly rather than repairing or replacing individual parts within it. The subsystem-level abstraction, which is illustrated in Figure 1.2, represents an intermediate level between individual sensors and the entire system in the system hierarchy. Health states that reflect the state of these subsystems could enable an analysis that balances the high level of detail provided by univariate sensor-level diagnosis approaches with the low requirements for system-wide anomaly detection methods in terms of necessary prior knowledge. The mapping between sensors and their corresponding subsystems can often be derived from the documentation or even from naming conventions in the data itself. The names of the sensor readings of the ECLSS, for example, follow a consistent naming pattern that includes the subsystem identifiers.

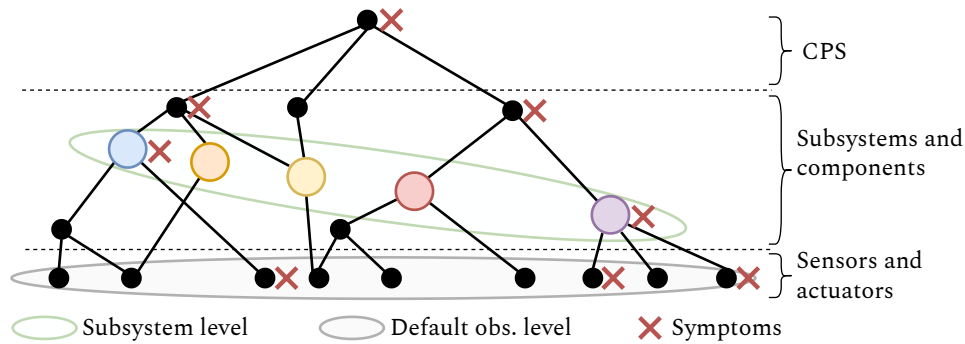


Figure 1.2: CPS hierarchy illustrating subsystem and observation levels. The colored nodes correspond to subsystems visualized in Figure 1.3. The proposed approach models health states at the subsystem level. This establishes a level between whole-system (multivariate) and individual sensor (univariate) in the system hierarchy.

In combination with symbolic AI algorithms, these binary health states are particularly promising because they would not only solve the problem of transferring

continuous values into logical expressions [124, 125], but also because they could be generated at precisely the level of detail that is ideal for diagnostic AI algorithm. Assuming that suitable Neural Network (NN) architectures can indeed be developed to generate these subsystem-level health states, the second research question addresses the subsequent diagnostic challenge:

RQ2: How can subsystem-level symptoms and structural information about fault propagation paths be combined for diagnostic reasoning without requiring detailed behavioral models?

The health states discussed above identify the parts of the overall system whose behavior deviates from normal behavior. However, in complex systems containing many interdependent subsystems, errors can propagate [145]. The search for the subsystem that triggered the fault is referred to as the diagnostic reasoning problem in the context of this work.

A second type of prior knowledge about the structure of the CPS that could be helpful is the causal fault propagation relationships between the subsystems [145, 188]. Such relationships, which are either already available or can easily be formalized in most cases, can be represented in the form of a directed graph in which the nodes represent the subsystems and the edges represent the paths along which symptoms can potentially propagate (see Figure 1.3).

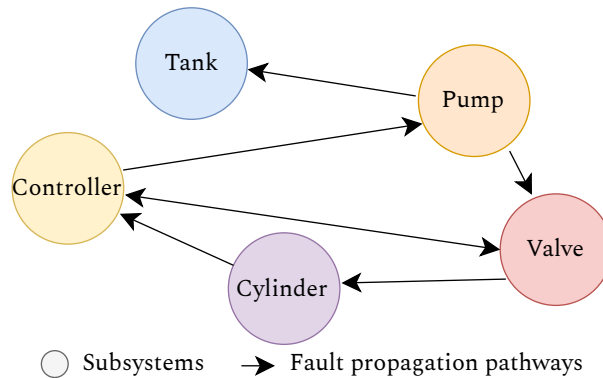


Figure 1.3: A fault propagation graph for an illustrative hydraulic system. The nodes represent the subsystems and the edges represent potential fault propagation paths between them.

In the illustrative example graph of a simple hydraulic system, as shown in Figure 1.3, the nodes are subsystems such as water tanks, pumps, valves, and hydraulic cylinders. The edges represent how, for example, a defective pump can lead to abnormal pressure readings in the connected valves, or how a leaky tank can cause irregular flow patterns in downstream components. If several subsystems show symptoms, it is not trivial to determine the subsystem that caused the symptoms. This research question therefore investigates whether the health states developed

in RQ1, together with the prior knowledge on the structure and fault propagation of the CPS, enable a diagnosis that does not require extensive modeling work.

RQ3: How can MLOps tools and practices support the implementation and operation of anomaly detection and diagnostic methods in production environments?

As already mentioned in Section 1.1, a significant part of the complexity of developing and using modern ML-based diagnostic methods lies in the deployment and maintenance of such systems. For this reason, this research question focuses on the applicability of the MLOps tools and principles to CPS-related use cases such as the diagnostic procedure discussed in this paper.

Many open source MLOps platforms run on top of Kubernetes, which is a container orchestration platform for resource allocation and workload isolation [25]. Since its release in 2015, Kubernetes has become the de facto standard for cloud-native infrastructure. Kubeflow [8] is a well-established MLOps platform designed explicitly for use with Kubernetes. It is open-source and bundles various tools that, in combination, are supposed to support the entire ML lifecycle. However, the existing documentation and case studies generally focus on the individual tools rather than describing and discussing their integration or the entire workflow, ranging from data exploration to the (re-)deployment of the ML-models [7, 8, 62].

Therefore, this research question investigates the extent to which the steps necessary for the development and deployment of the solution proposed in this thesis can be orchestrated in an automated workflow with the help of tools from the Kubeflow ecosystem. In this context, practical challenges and corresponding solution approaches are also investigated.

1.3 Contributions

In order to answer the research questions discussed above, this thesis makes four contributions to the fields of anomaly detection, diagnosis, and MLOps for CPSs.

Subsystem-Level Anomaly Detection Architecture A novel NN architecture is presented that is explicitly designed to localize anomalies within a CPS at the subsystem level. This architecture includes a latent space structure that incorporates prior knowledge about the CPS. The model uses this architecture to generate binary health states at the subsystem level based on the system's multivariate time series. These health states can be further processed downstream in symbolic diagnostic reasoning algorithms, thereby aiming to answer RQ1.

Graph-Based Diagnostic Algorithm The second contribution is a new diagnostic algorithm that identifies those subsystems whose faults are most likely to be the cause of the observed symptoms. The algorithm is based on graph theory and requires the fault propagation graph discussed in RQ2 for initialization. It evaluates potential

1 Introduction

root causes based on a set of criteria that are drawn from basic diagnostic principles. It has polynomial time complexity and avoids the explosion of computational complexity that is characteristic of many diagnostic approaches (answer to RQ2).

Integrated Diagnostic Framework The contributions mentioned so far can be used independently of each other and combined with other methods. However, they were developed with the aim of providing a comprehensive solution for the diagnosis of large and complex CPS when combined. The third contribution of this thesis is this combined framework. In line with the motivation to minimize the necessary prior knowledge, only three inputs are required for its initial implementation: (i) historical observations of the nominal system behavior, (ii) a fault propagation graph that represents basic causal relationships between subsystems, and (iii) a mapping from subsystems to the signals that correspond to them. These three inputs are usually available in practice or can be created with little effort. This approach enables both the detection and localization of symptoms in the CPS and the identification of causes, resulting in a sorted list of subsystems responsible for the observed anomalies (thus answering both RQ1 and RQ2).

MLOps Implementation The final contribution is an automated workflow using tools from the Kubeflow ecosystem that deploys the algorithmic solutions described in the contributions above in a reproducible and scalable way. The workflow orchestrates the entire process from preprocessing with distributed computing tools and hyperparameter search to model training, deployment, and automated retraining. The methods used are described in such a way that they can be transferred to other ML-based applications independently of the specific use case. Based on the exemplary implementation, both the strengths of the current features of the Kubeflow ecosystem and those areas where further development would be useful are identified (answering RQ3). Within the scope of this research, contributions to the open-source tools used were also made.

All contributions described in this section are evaluated within a series of experiments that examine both the components in isolation and the overall framework. The experiments use both simulated datasets and publicly available datasets from different real-world systems. The source code for the proposed solutions and all experiments discussed in this thesis is publicly available in the GitHub repositories of the corresponding publications (see next section) under the MIT license, along with detailed documentation.

1.4 Outline of this Thesis

The thesis is organized in four parts: (I) Background, (II) Formal Definition and Solution, (III) Evaluation, and (IV) Conclusion and Outlook.

Part I Background The first part of this thesis describes the theoretical background upon which the contributions of this thesis are built. Chapter 2 introduces the specific ML methods used in the anomaly detection models, the graph theory concepts relevant to the diagnosis algorithm, as well as the core concepts of MLOps. The first part also contextualizes the contributions within the current state of the art and distinguishes them from existing work in this field (Chapter 3).

Part II Formal Definition and Solution The second part describes the contributions of this thesis. To this end, Chapter 4 first formalizes the problem statement as well as the inputs and outputs of the proposed solution. Based on this formalization, Chapter 5 presents the individual components of the framework and their interaction along with the MLOps reference implementation.

Part III Evaluation Part III describes and discusses the experiments carried out to evaluate the contributions, as well as the theoretical results. The experimental results are covered in Chapter 6, and the theoretical results in Chapter 7.

Part IV Conclusion and Outlook Finally, the last part summarizes this thesis and provides an outlook on future research directions. Chapter 9 reviews the contributions and their impact on the CPS community. Chapter 10 describes research directions that, in the author’s opinion, could be promising for further developing the topics discussed in this thesis.

1.5 List of Publications

The contributions presented in this thesis have been published and discussed in several scientific publications. This section puts these publications in the context of this thesis’ research questions, and splits them into two lists. Publications that have made a significant contribution to this work are listed under “Main Publications” (prefix M), while those whose content has been taken into account to a lesser extent are listed under “Further Publications” (prefix F).

Publications Related to RQ 1: The NN architecture for subsystem-level anomaly detection has been partially published in the Proceedings of the 12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS 2024) [M1], where it was honored with the IFAC Young Author Award. An early version of this work was presented and discussed at the 34th International Workshop on Principles of Diagnosis [F3]. A related approach that also leverages the hierarchical structure of CPSs for anomaly detection was presented at IEEE ETFA 2024 [M2].

1 Introduction

Publications Related to RQ 2: The contributions to Research Question 2, particularly the graph-based diagnostic algorithm and its integration into the comprehensive framework, are published as a preprint [M3], which is currently under preparation for journal submission.

Publications Related to RQ 3: The results addressing Research Question 3 and the MLOps contributions were published in the Proceedings of ML4CPS 2024 [M4]. Content relevant to the MLOps and open-source communities was additionally presented and discussed at the Kubeflow Summit (KubeCon + CloudNativeCon Europe Co-Located Events) [M5] and PyCon DE & PyData 2025 [F14], both peer-reviewed presentations.

Further Publications: A purely data-driven diagnostic method utilizing fault labels was described in an international patent [F1]. The use case that motivated this research, a diagnosis and reconfiguration software for the Columbus module’s ECLSS, is described in a manuscript currently under review in the CEAS Aeronautical Journal [F2]. The complexity of modern CPSs as a fundamental challenge for diagnostic tasks is discussed in a publication at ML4CPS 2025 [F4]. Several additional publications address the robustness of NNs deployed in CPS [F5, F6], alternative approaches to combining symbolic AI for CPS diagnosis [F7, F8], the application of representation learning methods to CPS data [F9] and materials science [F10, F11], as well as best practices for ML research in general [F12]. The fundamental concepts of anomaly detection in CPS were also published as a book chapter in “Digital Transformation” [F13].

Main Publications

- [M1] H. S. Steude, L. Moddemann, A. Diedrich, J. Ehrhardt, and O. Niggemann. “Diagnosis driven Anomaly Detection for Cyber-Physical Systems”. *IFAC-PapersOnLine* 58, 4 2024. 12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2024, pp. 13–18. ISSN: 2405-8971,2405-8963
- [M2] J. Ehrhardt, P. Overlöper, D. Vranjes, H. S. Steude, A. Diedrich, and O. Niggemann. “Using Modular Neural Networks for Anomaly Detection in Cyber-Physical Systems”. In: *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2024, pp. 01–07
- [M3] H. S. Steude, A. Diedrich, I. Pill, L. Moddemann, D. Vranješ, and O. Niggemann. *Data Driven Diagnosis for Large Cyber-Physical-Systems with Minimal Prior Information*. 2025. arXiv: 2506.10613 [cs.AI]
- [M4] H. S. Steude, C. Geier, L. Moddemann, M. Creutzenberg, J. Pfeifer, S. Turk, and O. Niggemann. “End-to-end MLOps integration: a case study with ISS

telemetry data". In: *ML4CPS – Machine Learning for Cyber-Physical Systems* (Berlin, 2024). UB HSU, 2024

- [M5] H. S. Steude and C. Geier. "Efficient Integration of Kubeflow Tools: An ISS Data Case Study on Anomaly Detection". In: *KubeCon + CloudNativeCon Europe 2024 Co-located Events: Kubeflow Summit* (Paris Expo Porte De Versailles). Peer-reviewed presentation at CNCF-hosted co-located event. Paris, France, 19, 2024

Further Publications

- [F1] H. S. Steude and A. Demeschkin. "Method and computer program product for monitoring a bleed air supply system of an aircraft". Pat. 2022189235. National Phase Entries: US 18549599 (2024-03-21), EP 2022708959 (2024-01-17), KR 1020237034489 (2023-11-07), CN 202280034242.5 (2023-12-26). 15, 2022
- [F2] M. Tappe, L. Moddemann, H. S. Steude, N. Hranisavljevic, S. Myschik, C. Geier, M. Creutzenberg, P. Grashorn, H. Ernst, and O. Niggemann. "A Supervised AI-Based Toolchain for Anomaly Detection, Diagnosis, and Reconfiguration for the Life Support System of the Columbus Module of the ISS". *CEAS Aeronautical Journal*, 2025. Accepted, but final submission still due
- [F3] H. S. Steude, L. Moddemann, A. Diedrich, J. Ehrhardt, M. Kalech, and O. Niggemann. "Diagnosis Driven Anomaly Detection for CPS". in: *Proceedings of the 34th International Workshop on Principle of Diagnosis*. Loma Mar, USA, 2023
- [F4] N. Hranisavljevic, T. Westermann, S. Plambeck, H. S. Steude, G. Benndorf, and O. Niggemann. "A Model Learning Perspective on the Complexity of Cyber-Physical Systems". In: *ML4CPS – Machine Learning for Cyber-Physical Systems* (Berlin). UB HSU, 2025
- [F5] A. Windmann, H. S. Steude, and O. Niggemann. "Robustness and Generalization Performance of Deep Learning Models on Cyber-Physical Systems: A Comparative Study". In: *IJCAI 2023 Workshop of Artificial Intelligence for Time Series Analysis (AI4TS)*. 2023
- [F6] A. Windmann, H. S. Steude, D. Boschmann, and O. Niggemann. "Assessing Robustness for Prognostic Models on Cyber-Physical Systems". In: *Proceedings of the IEEE Emerging Technology and Factory Automation (ETFA)*. Not yet submitted. 2025
- [F7] L. Moddemann, H. S. Steude, A. Diedrich, and O. Niggemann. "Discret2di-deep learning based discretization for model-based diagnosis". *IFAC-PapersOnLine* 58, 4 2024, pp. 640–645

1 Introduction

- [F8] L. Moddemann, H. S. Steude, A. Diedrich, I. Pill, and O. Niggemann. “Extracting Knowledge using Machine Learning for Anomaly Detection and Root-Cause Diagnosis”. In: *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2024, pp. 1–8
- [F9] H. S. Steude, A. Windmann, and O. Niggemann. “Learning physical concepts in CPS: A case study with a three-tank system”. *IFAC-PapersOnLine* 55, 6 1, 2022. 11th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2022, pp. 15–22. ISSN: 2405-8963,2405-8971
- [F10] H. S. Steude, D. Vranješ, J. L. Augustin, O. Niggemann, M. Lange-Hegermann, and D. Höche. “Use of neural networks for automated analysis and modeling of elasto-plastic material properties of metallic tensile specimens”. *Automation 2022 (VDI Berichte)* 2399, 2022, pp. 505–520
- [F11] S. Grednev, H. S. Steude, S. Bronder, O. Niggemann, and A. Jung. “AI-assisted study of auxetic structures”. *Acta Polytechnica CTU Proceedings* 42, 2023, pp. 32–36
- [F12] O. Niggemann, B. Zimmering, H. S. Steude, J. L. Augustin, A. Windmann, and S. Multaheb. “Machine learning for cyber-physical systems”. In: *Digital Transformation*. Ed. by B. Vogel-Heuser and M. Wimmer. Springer Berlin Heidelberg, Berlin, Heidelberg, 2023, pp. 415–446. ISBN: 9783662650035,9783662650042
- [F13] D. Vranješ, J. Ehrhardt, R. Heesch, L. Moddemann, H. S. Steude, and O. Niggemann. “Design Principles for Falsifiable, Replicable and Reproducible Empirical Machine Learning Research”. In: *35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024)*. 2024, pp. 7–1
- [F14] H. S. Steude and C. Geier. *Scaling Python: An End-to-End ML Pipeline for ISS Anomaly Detection with Kubeflow*. Conference Talk. Darmstadt, Germany, 2025

Part I

Background

2 Foundations

Chapter Outline This chapter describes the theoretical foundations upon which the contributions of this thesis were developed. Section 2.1 briefly introduces popular NN architectures for time series analysis before focusing on Temporal Convolutional Networks (TCNs) and Variational Autoencoders (VAEs), which form the core components of the anomaly detection model proposed in this thesis. Section 2.2 describes the algorithms from graph theory that are used in the diagnostic reasoning algorithm. Finally, Section 2.3 provides insights into the MLOps principles and tools used in this work.

2.1 Time Series Analysis with Neural Networks

As already mentioned in Chapter 1, modern CPSs typically record multiple signals, in particular continuous-valued sensor readings. Therefore, the analysis of multivariate time series in CPS, for example for the detection of anomalies, is a much-discussed topic for which NNs have become the state-of-the-art methods in recent years [61, 86, 131, 192, 208]. The analytical challenge lies in modeling complex time-dependent interactions between multiple variables in typically large and messy CPS datasets. Many different neural architectures have been proposed, which are suitable to varying degrees for application to CPS data. These are briefly introduced in the next paragraph.

Classical Multilayer Perceptrons (MLPs) [151] are capable of approximating complex functions, but they lack a mechanism that facilitates modeling temporal dependencies. Recurrent architectures such as Long Short-Term Memory (LSTM) networks [81], Gated Recurrent Unit (GRU) networks [32], and Recurrent Independent Mechanisms (RIM) [68] were designed to address this limitation. However, their sequential nature of processing data also limits their ability to model long-range dependencies in time series, which are not uncommon in CPS. To address this limitation, Convolutional Neural Networks (CNNs), originally introduced for image processing, were adapted for time series by applying the convolution operation along the temporal dimension [187]. Building on this concept, TCNs were developed, which introduce so-called dilated convolutions (see Subsection 2.1.2) that improve the long-range modeling and computational efficiency of vanilla CNNs [12, 99] and are therefore well suited for application to CPS data. More recent architectures include Transformer-based models with attention mechanisms [182, 211] and the Mamba architecture [73] with state space models and selective scanning mechanisms.

The remainder of this section focuses on TCNs and VAEs, since these concepts are at the core of the neural architecture presented in Chapter 5. At the time of the development of this thesis’s contributions, TCNs represented a particularly suitable choice due to the advantages mentioned above. While researchers have proposed newer architectures in recent literature, such as Mamba or specialized Transformer variants that might further improve the performance, TCN-based approaches continue to show competitive results when benchmarked against CPSs-specific real-world datasets, as demonstrated in [192].

The following subsections cover the basic concepts of CNNs, TCNs, and VAEs, and provide the necessary background to understand the architectural innovations presented in later chapters.

2.1.1 Convolutional Neural Networks Overview

CNNs follow three principles: local receptive fields, weight sharing, and pooling operations. Local receptive fields allow individual neurons to process only a small region of the input, which makes it easier to learn local patterns and corresponding abstractions in the data. Weight sharing means that the same set of weights is applied to different parts of the input. This allows the patterns to be learned independently of their location in the time series. Pooling operations, that are often used in CNNs, are a type of downsampling that reduces the dimensions of the feature maps. When combined, these three principles allow CNNs to learn complex patterns with fewer parameters compared to MLPs.

The fundamental building block of CNNs is the convolutional layer. In this layer, a set of parameters that is optimized during training is applied to the input data in a convolution-like operation. Despite its name, in the deep learning frameworks used in practice, such as PyTorch [137], these convolutional layers technically don’t implement a convolution but a cross correlation. The difference from “true convolutions” is simply that the filters are not mirrored before the calculation.

In applications on CPS data, CNNs apply one-dimensional convolutions along the time dimension. These convolutions run in parallel in many so-called channels, where channels in the context of CPSs typically represent different sensor signals. Although a multivariate time series has two dimensions just like a grey-scale image, in these one-dimensional convolutions the convolution is only performed along the time dimension and not along both dimensions as in image recognition. That means that an input tensor $\mathbf{X} \in \mathbb{R}^{C_{\text{in}} \times L}$ with $C_{\text{in}} \in \mathbb{N}$ input channels and a sequence length $L \in \mathbb{N}$, computes an output $\mathbf{Y} \in \mathbb{R}^{C_{\text{out}} \times L_{\text{out}}}$, when processed by a convolutional layer with $C_{\text{out}} \in \mathbb{N}$ filters. The output length $L_{\text{out}} \in \mathbb{N}$ depends on the input length L and kernel size $K \in \mathbb{N}$.¹ Each element of the output can be computed as:

$$\mathbf{Y}_{j,i} = \mathbf{b}_j + \sum_{k=0}^{C_{\text{in}}-1} \sum_{m=0}^{K-1} \mathbf{W}_{j,k,m} \cdot \mathbf{X}_{k,i+m} \quad (2.1)$$

¹ L_{out} also depends on padding, stride, and dilation. For a detailed discussion of these topics, see Murphy [128] Chapter 14.

where $\mathbf{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times K}$ represents the filter weights with kernel size $K \in \mathbb{N}$, $\mathbf{b} \in \mathbb{R}^{C_{\text{out}}}$ is the bias term, $i \in \mathbb{N}$ is the time index, and $j \in \mathbb{N}$ is the output channel index. Note that each filter can be assigned to an output channel. The number of filters, each with their own sets of weights, therefore determines the dimensions of the output, as shown in Figure 2.1.

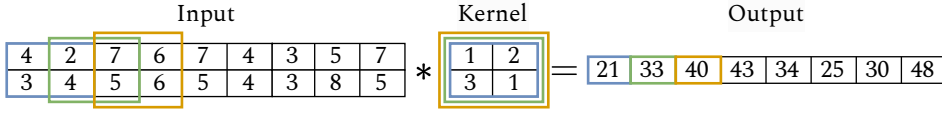


Figure 2.1: Illustration of a multi-channel one-dimensional convolution visualizing Equation 2.1. The two input channels (\mathbf{x}_1 and \mathbf{x}_2) each represent individual sensor readings at discrete time points. Both signals are processed by a single filter with kernel size 2. Note that the figure omits the bias term \mathbf{b} for simplicity.

As in other neural networks, nonlinear activation functions are also used in CNNs. In practice, the Rectified Linear Unit (ReLU) is often used:

$$\text{ReLU}(x) = \max(0, x). \quad (2.2)$$

Pooling layers reduce the spatial (in the case of CPS data, this refers to signals) or temporal dimension of feature maps. A very popular variant is max pooling [100], which outputs the maximum value within each window of $p \in \mathbb{N}$ consecutive values from an input sequence. This can be formalized as:

$$\text{MaxPool}_p(\mathbf{x})(i) = \max_{j=0, \dots, p-1} \mathbf{x}_{p \cdot i + j}. \quad (2.3)$$

Due to the features of one-dimensional CNNs described above, they are well suited for the analysis of multivariate time series [187]. However, one of their limitations is that the number of weights increases linearly with the size of the receptive field, which is the number of input elements processed within a convolution. This limitation is addressed by TCNs, introduced in the following subsection.

2.1.2 Temporal Convolutional Networks (TCNs)

TCNs extends CNNs with two new features, which are introduced in the following: dilated convolutions and residual connections [12]. Both of these features aim to increase the ability to learn long-term patterns.

Dilated Convolutions

Dilated convolutions were introduced by Yu et al. [204] in a paper on image processing. Using these dilated convolutions, the receptive field can grow exponentially with the depth of the network, measured in the number of NN layers, if the layers

2 Foundations

are designed accordingly. As described above, this growth is linear in non-dilated CNNs.

For a one-dimensional input sequence $\mathbf{x} \in \mathbb{R}^L$ of length L and a filter $\mathbf{w} \in \mathbb{R}^K$ with kernel size K , the dilated convolution for a single channel (for simplicity) can be written as:

$$\mathbf{y}_i = \sum_{j=0}^{K-1} \mathbf{x}_{i+d \cdot j} \cdot \mathbf{w}_j \quad (2.4)$$

where d is the dilation rate, i is the position in the output sequence, and j indexes the filter weights. Formula 2.4 shows how the dilation rate increases the size of the receptive field. This effect becomes even clearer in Figure 2.2, which shows the exponential growth of the receptive field with the number of stacked layers, when used in TCNs.

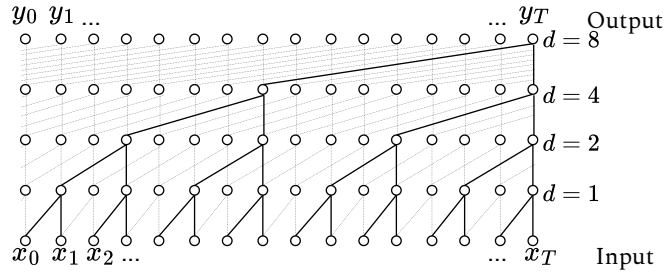


Figure 2.2: Dilated causal convolutions in a TCN with kernel size 2. Each row represents a layer within the TCN. The dilation rate increases with the layers ($d = 1, 2, 4, 8$). The connections illustrate how TCNs ensure what Bai et al. [12] call causality. Each output depends only on current and previous inputs, never on future ones. The figure also visualizes the exponential growth of the receptive field with the number of layers.

In addition to dilated convolutions, Bai et al. [12], who proposed TCNs, use so-called causal convolutions. In these convolutions, padding and masking are applied in a way such that the output at timestamp i only depends on inputs from timestamp $t \leq i$.²

The exponential growth of the receptive field mentioned above is achieved by the fact that the dilation rate grows with the pattern $d = 2^{l_{\text{NN}}}$ for layer $l_{\text{NN}} \in \mathbb{N}$, which is typical for TCNs. This means that even relatively shallow models can model long time spans. For a TCN with L_{NN} layers and a kernel size of K , the receptive field size $\rho \in \mathbb{N}$ grows according to:

²Padding refers to adding extra values (often zeros) to the input before applying the convolution. It can be used to modify the dimensions of the output. Masking describes the process of multiplying parts of the convolution outputs with zero. For details, see Murphy [128], Chapter 14 and 15.

$$\rho = 1 + (K - 1) \sum_{l=0}^{L_{\text{NN}}-1} 2^l. \quad (2.5)$$

Residual Connections

The second way TCNs extend standard CNNs is through residual connections, or shortcuts. These shortcuts are added to mitigate the gradient vanishing problem in backpropagation of deep neural networks [76] and are typically implemented within residual blocks. As shown in Figure 2.3, one of these blocks consists of two dilated convolutional layers, each followed by a normalization layer (batch or layer normalization, see [128], Chapter 14 for details) and the activation function [12]. The shortcut is that the input of these blocks is added to the output:

$$\mathbf{y} = \sigma(\mathbf{x} + f(\mathbf{x}, \boldsymbol{\theta}_{\text{block}})) \quad (2.6)$$

where $\boldsymbol{\theta}_{\text{block}}$ represents the trainable parameters within the residual block. Note that in case of different dimensions of the input and output, a 1×1 convolution is applied to the input \mathbf{x} before adding it to the output of the residual block $f(\mathbf{x}, \boldsymbol{\theta}_{\text{block}})$.

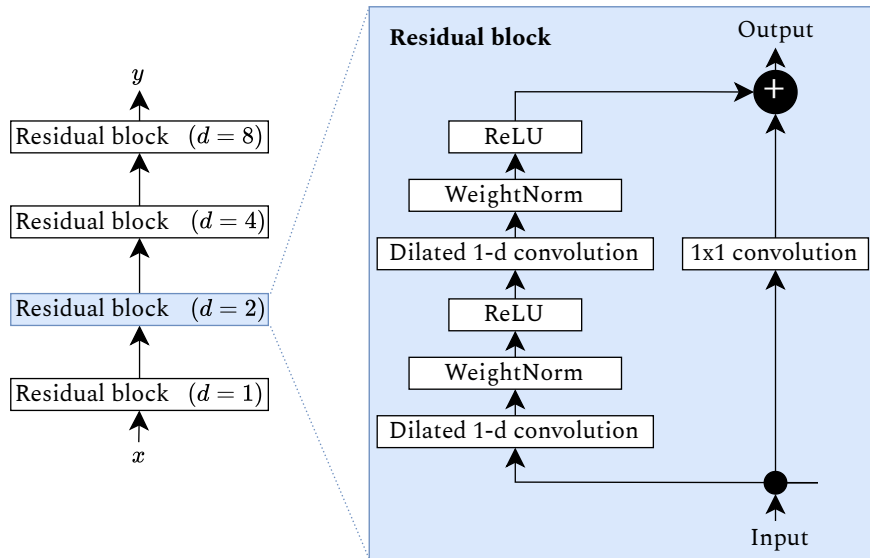


Figure 2.3: Neural architecture overview of a TCN. The left side shows an overview of the entire TCN, including the increasing dilation rate per layer. The right side is a detailed view of the components within the residual blocks, including the shortcut or residual connection.

Due to the combination of residual connections and dilated convolutions, TCNs are particularly well suited for time series analysis in the CPS context because they

can model not only long-range but also high-frequency patterns in multivariate time series. Anomalies, for example, can occur in the form of short outliers or slow regime changes [192]. Furthermore, TCNs are computationally efficient, at least relative to recurrent or fully connected NNs, which is useful for real-time analysis on streaming data or applications on edge devices.

2.1.3 Autoencoders for Dimensionality Reduction

Autoencoders (AEs) are NN architectures from the group of representation learning models [80]. In most cases, they are used to perform a dimensionality reduction very similar to that of PCAs [139], but in contrast to the PCA, AEs are highly non-linear, at least as long as they contain non-linear activation functions. Sakurada and Yairi [154] show how this non-linearity can, for example, increase the anomaly detection performance on multivariate time series.

Figure 2.4 illustrates this dimensionality reduction using a sample from the MNIST dataset [100], which is often used for didactic purposes in ML literature, especially since images can be easily visualized in a natural way.

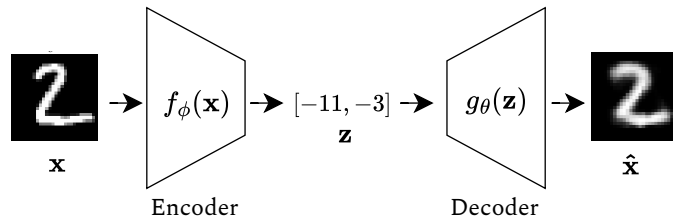


Figure 2.4: Basic architecture of an AE illustrated with an MNIST digit sample. The encoder, f_ϕ , compresses the input image $\mathbf{x} \in \mathbb{R}^n$ (where $n = 784$ for a flattened 28×28 pixel image) into a lower-dimensional latent representation $\mathbf{z} \in \mathbb{R}^m$ (where typically $m \ll n$, for example, $m = 2$). The decoder, g_θ , then attempts to reconstruct the original input from this compressed representation and computes $\hat{\mathbf{x}} \in \mathbb{R}^n$.

The encoder, which is typically implemented as a NN f_ϕ with parameters ϕ , maps the input data $\mathbf{x} \in \mathbb{R}^n$ to a lower-dimensional latent representation $\mathbf{z} \in \mathbb{R}^m$ where $m \ll n$:

$$\mathbf{z} = f_\phi(\mathbf{x}). \tag{2.7}$$

The decoder, a second NN g_θ with parameters θ , then computes the reconstruction of the original input from this latent representation:

$$\hat{\mathbf{x}} = g_\theta(\mathbf{z}) = g_\theta(f_\phi(\mathbf{x})). \tag{2.8}$$

The weights of the AE are optimized to minimize a reconstruction loss function $\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}})$ that measures the difference between the original input and its reconstruction, the output of the AE. For a dataset with N samples, the total loss is:

$$\mathcal{L}(\phi, \theta) = \sum_{i=1}^N \mathcal{L}(\mathbf{x}_i, g_{\theta}(f_{\phi}(\mathbf{x}_i))). \quad (2.9)$$

where \mathbf{x}_i represents the i -th training example. A typical choice for continuous data is the Mean Squared Error (MSE):

$$\mathcal{L}_{MSE}(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2. \quad (2.10)$$

2.1.4 Autoencoders for Anomaly Detection

The ability of AEs to compress data samples into a latent space via a non-linear transformation and to reconstruct them from this compressed representation makes them well suited to detect anomalies in data. For this purpose, an AE is trained exclusively on data that does not contain any anomalies. The weights are optimized to reconstruct samples from the distribution of normal samples. The reconstruction of a sample that does not originate from the distribution of normal data, which is the case, for example, with anomalies in CPS time series, will therefore exhibit a larger reconstruction error, which makes this error a natural anomaly score [154]. Formally, the anomaly score for a new data point \mathbf{x} can be defined as:

$$\alpha(\mathbf{x}) = \|\mathbf{x} - g_{\theta}(f_{\phi}(\mathbf{x}))\|^2. \quad (2.11)$$

A threshold τ can then be applied to this score to classify the input as either normal or anomalous:

$$\delta(\mathbf{x}) = \begin{cases} 1, & \text{if } \alpha(\mathbf{x}) > \tau \\ 0, & \text{otherwise} \end{cases} \quad (2.12)$$

where $\delta(\mathbf{x}) = 1$ indicates an anomaly and $\delta(\mathbf{x}) = 0$ indicates normal behavior.

The AE framework can be easily adapted to different data types because any NN layer architecture can be used in the encoder and decoder networks. For CPS time series, for example, Recurrent Neural Networks (RNNs) or TCNs are suitable (see Section 2.1), so that the latent representation also contains information about temporal dependencies [143].

2.1.5 Variational Autoencoders for Anomaly Detection

VAEs presented by Kingma and Welling [93] extend the AE framework to a probabilistic model. In VAEs, data samples are not mapped directly into a latent space, but rather onto the parameters of the latent space distribution (see Figure 2.5). This probabilistic setup leads to more regularized and disentangled latent space repre-

sentations, quantifiable uncertainty, and the ability to use the decoder as a generative model. The following discussion of VAEs is based on the comprehensive treatment in Murphy [128] (Chapter 20) and the tutorial overview by the authors of the original VAE formulation Kingma and Welling [92].

From Deterministic to Probabilistic Encoding

As already indicated, the encoder in the VAE produces the parameters of a probability distribution, typically mean $\mu_\phi(\mathbf{x})$ and standard deviation $\sigma_\phi(\mathbf{x})$. The latent space representation \mathbf{z} that is passed to the decoder is then sampled from the distribution specified by these parameters. Figure 2.5 visualizes this architecture, using a sample from the MNIST dataset like in the AE example in Figure 2.4.

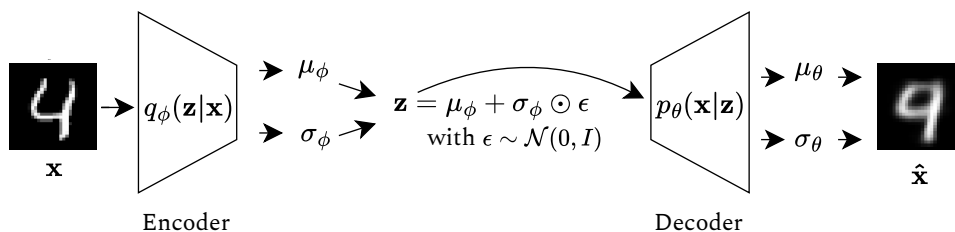


Figure 2.5: Architecture of an VAE illustrated with an MNIST digit sample. Unlike the deterministic AE in Figure 2.4, the VAE encoder $q_\phi(\mathbf{z}|\mathbf{x})$ maps the input to distribution parameters $\mu_\phi(\mathbf{x})$ and $\sigma_\phi(\mathbf{x})$. The latent representation \mathbf{z} is then sampled from this distribution. The sampling is done using the reparameterization trick to allow gradient computation for the backpropagation algorithm $\mathbf{z} = \mu_\phi(\mathbf{x}) + \sigma_\phi(\mathbf{x}) \odot \epsilon$, where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$. The decoder $p_\theta(\mathbf{x}|\mathbf{z})$ then reconstructs the input from this sampled latent variable.

In the case of a Gaussian prior distribution, which is used in most cases³ the probabilistic data generation process can be described as follows: The latent variable \mathbf{z} is sampled from a prior distribution $p(\mathbf{z})$. The sample \mathbf{x} in the data space is then generated from this latent variable by a conditional distribution:

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mu_\theta(\mathbf{z}), \sigma^2\mathbf{I}). \quad (2.13)$$

Here, $\mu_\theta(\mathbf{z})$ represents the output of the decoder network and σ^2 the variance parameter that can either be fixed or optimized during training. The marginal likelihood of a point \mathbf{x} can be written as:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} \quad (2.14)$$

³Gaussian distributions are the most common choice for the prior distribution in VAEs. However, there are alternatives, for example, to model categorical or discrete latent variables [85, 113], which are also used for CPS datasets [124, 125].

which is a generally intractable integral. Therefore variational inference techniques are needed as described in the following paragraphs.

Variational Inference for Training

The encoder network defines an approximate posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$ that approximates the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$ by computing its parameters μ_ϕ and σ_ϕ^2 :

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mu_\phi(\mathbf{x}), \sigma_\phi^2(\mathbf{x})\mathbf{I}). \quad (2.15)$$

The parameters ϕ and θ of the encoder and decoder networks, respectively, are optimized simultaneously in the same backward pass of the backpropagation. The loss function used for training is the Evidence Lower Bound (ELBO) $\mathcal{L}(\theta, \phi; \mathbf{x})$. As derived in Kingma and Welling [93] on page 18, the ELBO provides a lower bound for the marginal log likelihood and consists of two terms. (i) The reconstruction term, highlighted in equation 2.16, represents the expected value of the log likelihood of a data sample \mathbf{x} conditional on the latent space variable \mathbf{z} . Note that this expected value not only depends on the decoder's parameters θ , but also on the encoder's parameters ϕ , as indicated by the index of the expectation symbol $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}$. (ii) The second term, also highlighted in equation 2.16, is the regularization term, which encourages posterior distributions $q_\phi(\mathbf{z}|\mathbf{x})$ to be close to the prior $p(\mathbf{z})$, measured by the Kullback-Leibler divergence:

$$\log p_\theta(\mathbf{x}) \geq \mathcal{L}(\theta, \phi; \mathbf{x}) = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction term}} - \underbrace{D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))}_{\text{Regularization term}}. \quad (2.16)$$

Because the forward pass of the VAE involves a sampling process, direct gradient calculation is not possible. For this reason, the so-called reparameterization trick is used, which computes the latent variable \mathbf{z} instead of directly sampling from $q_\phi(\mathbf{z}|\mathbf{x})$ as follows:

$$\mathbf{z} = \mu_\phi(\mathbf{x}) + \sigma_\phi(\mathbf{x}) \odot \epsilon \quad (2.17)$$

where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ is an auxiliary noise variable and \odot represents element-wise multiplication.

The β -VAE Extension

The β -VAE, introduced by Higgins et al. [78], is an extension of the standard VAE that introduces a hyperparameter β which controls the weight of the KL divergence term in the ELBO loss:

$$\mathcal{L}_\beta(\theta, \phi; \mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \underbrace{\beta}_{\text{New}} \cdot D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})). \quad (2.18)$$

2 Foundations

By adjusting β , it is possible to control the trade-off between reconstruction accuracy and regularization (the constraint on the latent space structure). With values that satisfy $\beta > 1$, the model learns a more disentangled representation at the cost of reconstruction quality. When $\beta < 1$, the loss puts more emphasis on accurate reconstruction and moves the latent posterior further away from its prior. Burgess et al. [23] provide a detailed analysis of how β affects the disentanglement of latent variables and describe strategies for modifying this parameter during training. For complex multivariate time series data, the β parameter is particularly important, as standard VAEs ($\beta = 1$) often struggle with the “posterior collapse” problem, where the model ignores the latent space in favor of minimizing the KL divergence [109]. The β -VAE framework’s flexibility and its probabilistic foundation, makes it a well suited basis for the subsystem-level anomaly detection approach presented in Chapter 5.

The effects of the VAE framework on the latent space are illustrated in Figure 2.6. The visualized latent space representations of MNIST images are predictions from two models, an AE (left) and a VAE (right). Both models were trained for 20 epochs on the training data, while the figure shows the latent space representations of the test samples. The dimension of the latent space is 2 in both models, $\mathbf{z} \in \mathbb{R}^2$, so that, in contrast to the visualization by Murphy [128], no additional t-SNE [112] transformation was used here. The visualization shows that the latent space of the VAE has fewer low-density regions between the digit clusters and that the entire distribution is much closer to a 2-D Gaussian (the prior) than the one produced by the AE.

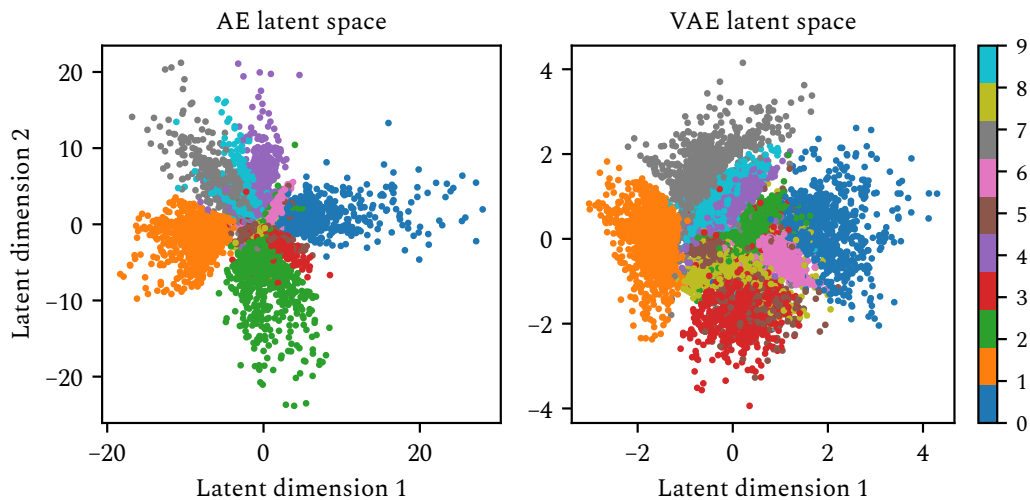


Figure 2.6: Comparison of latent spaces in AE (left) and VAE (right) for MNIST digits. The colors represent the digit classes (0-9). For the VAE, the scatter plot shows the mean parameters μ_ϕ of the latent distribution.

VAE-based Anomaly Detection

Similar to AEs, the reconstruction quality can also be used as an anomaly score for VAE. However, because the framework is probabilistic, VAEs use the likelihood of the observed data points \mathbf{x} under the learned model instead of comparing them to their reconstruction:

$$\alpha(\mathbf{x}) = -\log p_{\theta}(\mathbf{x}|\mu_{\phi}(\mathbf{x})). \quad (2.19)$$

As with AEs, a threshold τ can be applied to these scores to make binary anomaly decisions:

$$\delta(\mathbf{x}) = \begin{cases} 1, & \text{if } \alpha(\mathbf{x}) > \tau \\ 0, & \text{otherwise} \end{cases} \quad (2.20)$$

where again $\delta(\mathbf{x}) = 1$ indicates an anomaly.

For a more comprehensive discussion on VAEs, readers are referred to Chapter 20 of Murphy's book [128] on probabilistic machine learning and the tutorial overview by Kingma and Welling [92].

2.2 Graph-Based Algorithms for Diagnostic Reasoning

Graph theory is a widely used framework for modeling entities and their relationships, which is used across domains in various applications [18, 42, 130]. Graphs are also used in the diagnosis literature [83, 95, 145, 155]. Since the diagnostic reasoning algorithm presented in this thesis is also based on the analysis of graphs, this section describes the necessary basic concepts from graph theory.

2.2.1 Fundamentals of Graph Theory

According to the introduction to graph theory by Diestel [42], a graph $G = (V, E)$ is defined by a set of vertices (or nodes) V and a set of edges E that represent relationships between them.

Graphs can be divided into undirected and directed graphs, among other types. In undirected graphs, edges are unordered pairs $\{u, v\}$ where $u, v \in V$ represent a bidirectional relationship between u and v . In directed graphs, on the other hand, these pairs are ordered (u, v) and represent a relationship from node u to v and not vice versa.

Another graph concept used in the description of the algorithm proposed in this thesis are paths. In a directed graph, a path describes a sequence of nodes, v_1, v_2, \dots, v_n such that $(v_i, v_{i+1}) \in E$ for all $i \in \{1, 2, \dots, n-1\}$, which means that node v is reachable from node u if there is a path from u to v . A cycle is therefore a path that starts at a node and ends at the same node, as illustrated in Figure 2.7 using nodes B, C, and D as examples. While Directed Acyclic Graphs (DAGs) don't contain such cycles, cyclic graphs do. However, DAGs offer computational advantages for algorithms such as topological sorting, Bayesian network inference, and causal discovery.

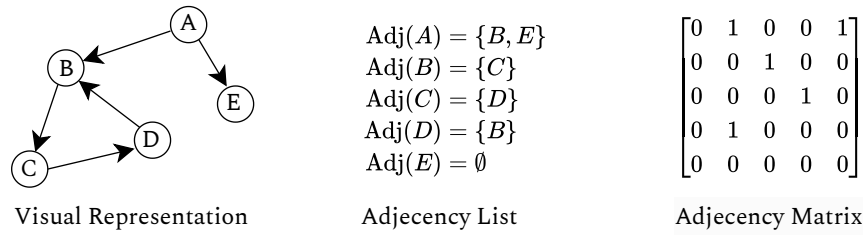


Figure 2.7: Different representations of a directed cyclic graph. The figure shows the visual representation of a graph (left), its adjacency list (center), and the adjacency matrix (right). The diagnostic algorithm presented in this thesis uses directed cyclic graphs for modeling fault propagation between subsystems. The cycle $B \rightarrow C \rightarrow D \rightarrow B$ in this example illustrates a feedback pattern that cannot be represented in acyclic graphs.

Adjacency lists and adjacency matrices are the most commonly used representations of graphs (see Figure 2.7). An adjacency list representation maintains, for each vertex, a list of its adjacent vertices, and the adjacency matrix is a $|V| \times |V|$ matrix where entry (i, j) indicates whether an edge exists from vertex i to vertex j . Both have their advantages and disadvantages in different applications. Adjacency lists are memory-efficient for sparse graphs (where $|E| \ll |V|^2$), since they require $O(|V| + |E|)$ space, and $O(1)$ time complexity for iteration over neighbors. Adjacency matrices, on the other hand, have a $O(1)$ time complexity for edge lookups but require $O(|V|^2)$ space regardless of the number of edges.

2.2.2 Graph Algorithms

This section introduces the graph algorithms that form the basis for the diagnostic reasoning algorithm presented in Chapter 5, including Breadth-First Search (BFS), Depth-First Search (DFS), and distance-based node grouping. A comprehensive description of these algorithms can be found in the standard textbooks by Cormen et al. [36] and Bondy and Murty [18].

Breadth-First Search

The BFS is a search algorithm for graphs which, as its name suggests, first searches broadly and then slowly works its way deeper. Within a loop, the algorithm visits all neighbors of a starting node, then the neighbors of all neighbors, and so on. The nodes that have already been visited and those that still need to be visited are stored in a queue. In each iteration, the algorithm takes a node that has not yet been visited from the queue and repeats the procedure until the queue is empty.

In this way, the shortest paths from the start node to all other nodes are found in a natural way, because the path that the algorithm has taken when it goes from the start node $u \in V$ to any other node $v \in V$ is also the shortest path from u to v . BFS has a time complexity of $O(|V| + |E|)$.

Depth-First Search

Another search algorithm for graphs is DFS, which goes deep into the graph before exploring breadth. The algorithm travels along the edges leading away from the current node as long as there is a path to a node that has not yet been visited. Only when this is no longer the case does it retrace its steps and search for alternative paths. This algorithm can be implemented either recursively or with a stack.

Like BFS, DFS has a time complexity of $O(|V| + |E|)$. However, the paths and memory usage of these approaches differ. While BFS requires storing all nodes at the current level (which can be large in wide graphs), DFS needs to store only the path from the root to the current $v \in V$, which reduces memory usage for deep graphs.

Distance-Based Node Grouping

One method for clustering nodes within a graph according to their distance from a start node is to use “shells”. A “shell” V_k contains all nodes whose paths from the start node have a certain length: $V_k = \{v \in V \mid \text{distance}(u, v) = k\}$. Because of how it explores nodes level by level, BFS is particularly well suited for identifying these “shells”.

The distribution of nodes across these shells can reveal structural properties of the graph, such as expansion rates and clustering patterns, which might provide useful information on how effects travel through the graph.

2.3 Machine Learning Operations (MLOps)

Research question RQ3 of this thesis is concerned with topics from the field of MLOps. For this reason, this chapter introduces some basic concepts from this discipline and places them in the context of the Cross-Industry Standard Process for Data Mining (CRISP-DM) cycle. The section also discusses the cloud-native infrastructure upon which many modern MLOps platforms are built.

2.3.1 The ML Development Lifecycle and CRISP-DM

As Martinez-Plumed et al. [115] recently noted, the CRISP-DM [161] is still one of the most widely used methods for structuring machine learning projects, even though it was introduced more than 20 years ago. The process model, which is also used

in the field of CPS [63], defines six phases: (i) business understanding, (ii) data understanding, (iii) data preparation, (iv) modeling, (v) evaluation, and (vi) deployment. This iterative process is illustrated in Figure 2.8.

MLOps is a discipline that also focuses on ML-projects, but puts more emphasis on the technical challenges and corresponding tools. Recent reviews of MLOps [15, 97] identify functional areas of MLOps that can be mapped to the six phases of the CRISP-DM cycle. While CRISP-DM deals with the conceptual framework of ML projects, MLOps is more concerned with the tools needed to get ML software into production and to maintain it. The following section maps these MLOps tools and practices to the six phases of the CRISP-DM.

- (a) **Data versioning and management:** [CRISP-DM phases (ii) and (iii)] Similar to how source code management tools such as Git track the status of software and its changes, these tools were designed to track how datasets used to train ML models were created. This also includes data transformations which might have a significant impact on the quality of the model.
- (b) **Model development:** [CRISP-DM phase (iv)] The development of the ML models themselves is supported by many MLOps tools, which support for example, model selection or hyperparameter tuning. In most cases, this is based on common open-source ML frameworks such as PyTorch, Tensorflow, or scikit-learn.
- (c) **Distributed training:** [CRISP-DM phase (iv)] Especially for large data sets, but also for large models, distributed training can reduce training time significantly. Distributed training refers to the parallel execution of calculations required to train individual models. Since this parallelization increases the technical complexity of the training, many modern MLOps tools attempt to simplify this process for developers.
- (d) **Model testing/validation:** [CRISP-DM phase (v)] ML models need to be evaluated and tested before they are used. Tools in this category support the creation of plots and metrics that describe the performance of the models.
- (e) **Model deployment:** [CRISP-DM Phase (vi)] Here, MLOps tools offer a wide variety of solutions designed to make it easier for data scientists to transition from experimental models to production systems. Topics such as model versioning, containerization, and the provision of Application Programming Interfaces (APIs) are traditionally handled by data engineers rather than data scientists.
- (f) **Model inference:** [CRISP-DM Phase (vi)] These tools are closely related to those used in model deployment and include, among other things, optimized serving infrastructure to efficiently deploy various types of ML models while taking into account latency, throughput, and resource utilization.

- (g) **Model performance monitoring** [CRISP-DM phases (vi) → (i)] This group includes tools and approaches that systematically test the deployed models. As motivated in Section 1.1.4, this step is particularly relevant for identifying model or concept drift in models for CPS that change over time.
- (h) **Experiment tracking and metadata storage:** [CRISP-DM phases (iv), (v)] The experimental nature of the development process of ML models means that in practice it is often unclear how exactly models were created or how they could be reproduced. Tools in this category therefore provide experiment tracking and model versioning support.

Figure 2.8 illustrates how these MLOps features map to the traditional CRISP-DM cycle.

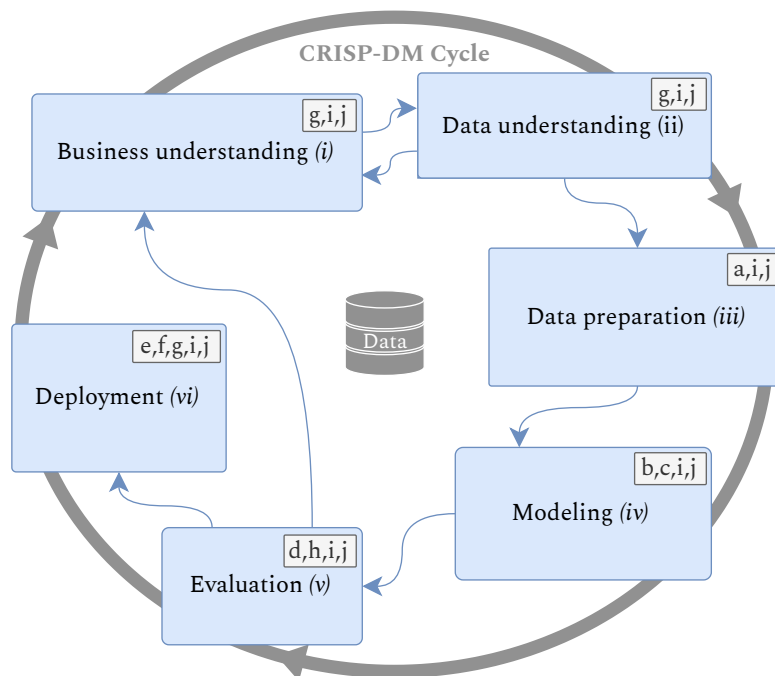


Figure 2.8: The CRISP-DM cycle with MLOps features mapped to each phase. The six-phase cycle shows the standard CRISP-DM process (i-vi). These MLOps features are mapped to each phase: (a) Data Versioning and Management, (b) Model Development, (c) Distributed Training, (d) Model Testing/Validation, (e) Model Deployment, (f) Model Inference, (g) Model Performance Monitoring, (h) Experiment Tracking and Metadata Store, (i) Orchestration, and (j) Code Management.

In addition to features that map well to individual phases of the CRISP-DM, MLOps platforms typically also offer features that span the entire process or are relevant to the entire process. These include:

2 Foundations

- (i) **Orchestration:** This refers in particular to pipelining tools that combine the features described above into a workflow and can execute the individual steps automatically.
- (j) **Code Management:** When it comes to code management, the development of ML software is no exception, and most MLOps tools support one or another code management software such as Git [180].

2.3.2 Containerization and Cloud-Native Infrastructure

Many production-grade ML deployments and even entire MLOps platforms rely on containerization to meet requirements for flexibility, hardware and infrastructure independence, and scalability. Kubernetes is a container orchestration platform that has established itself as the de-facto standard for managing container-based software since its launch 10 years ago [25]. The most important features of Kubernetes include: (i) efficient resource management for Central Processing Unit (CPU), memory, and accelerators such as Graphics Processing Units (GPUs), (ii) horizontal scaling to handle variable workloads, (iii) workload isolation to run multiple pipelines concurrently, and (iv) declarative configuration that enables infrastructure-as-code practices. MLOps platforms running on Kubernetes extend this feature set with the tools described in Section 2.3.1. A popular example of this is the KubeFlow ecosystem, which ranked first in a recent comparison by Berberiet al. [15] that evaluated both feature coverage and community adoption of various open source tools.

For GPU intensive tasks such as training deep NNs, KubeFlow integrates parallelization frameworks such as PyTorch Distributed Data Parallel (DDP) [104] or TensorFlow's distributed training strategies [1], for which the GPUs for the individual training processes in Kubernetes can be provided via the open source NVIDIA GPU Operator, for example. CPU-based workloads can also be distributed with Kubernetes in the backend using frameworks such as Dask [149] or Spark [206]. A major advantage of these Kubernetes-based solutions is their flexible resource allocation. Compute resources can be shared so that multiple users can access them, or combined so that individual jobs can use the power of many compute nodes simultaneously.

Despite, and in part because of, the rapid development of the MLOps tool landscape in recent years, the field also faces unresolved challenges. The wide range of different tools, some with different concepts and philosophies, makes it difficult for practitioners to choose the right tool for their use cases. No standards have yet been defined for the interfaces of the individual features [15].

3 State of the Art

Chapter Outline This chapter provides an overview of the state-of-the-art in the research areas relevant to the contributions of this thesis. Section 3.1 therefore covers DL approaches for anomaly detection and emphasizes solutions that make use of prior knowledge about the structure of CPSs, while Section 3.2 reviews data-driven and ML-based solutions applied to diagnosis problems. Furthermore, Section 3.3 discusses the current state of research in the field of MLOps. Finally, Section 3.4 outlines the research gaps addressed by this work.

3.1 Anomaly Detection in Cyber-Physical Systems

Despite some contributions that question the superiority of DL over traditional static methods for anomaly detection in multivariate time series [5, 147, 156], current research largely focuses on DL methods that have become the state-of-the-art for this task [61, 86, 134, 208]. The overview of current research in this section therefore mainly focuses on DL-based work.

3.1.1 Modern Anomaly Detection Methods

According to the taxonomy proposed by Zamanzadeh Darban et al. [208], DL approaches for time series anomaly detection in CPSs can be grouped into different categories, of which the main ones are reconstruction-based and prediction-based methods. The following paragraphs therefore focus on these two categories of models, which make up the majority of the current literature in this field.

Prediction-based Methods

Prediction-based methods apply models trained on data describing the normal operations of the CPS to make predictions for future values of these signals. The core idea is that these models fail to predict time series when anomalies occur since the anomalies cause previously unknown patterns in the time series. Anomalies can therefore be identified by increased prediction errors. These approaches have evolved from traditional time series forecasting techniques, including statistical and classical ML-based methods, to sophisticated DL architectures, as discussed by Audibert et al. [5]. This section provides an overview of the main groups of these DL architectures.

Recurrent neural networks (RNNs) were developed to model sequence data. Therefore, they are among the earliest DL models that were used for prediction-based anomaly detection in CPSs time series. More recent CPS literature also contains papers that propose newer RNN architectures such as GRUs [32] and RIMs [68] for time series forecasting, however, LSTMs are used in most studies [19, 49, 114, 190, 191]. Ezeme et al. [51] and Du et al. [46] show how RNNs can be used not only for anomaly detection in sensor signals, but also in text-based log files of technical systems. The authors also point out that RNN models can be updated with new data after initial training to counteract concept and data drift.

Temporal Convolutional Networks (TCNs), which are presented in detail in Section 2.1, are also widely used for prediction-based anomaly detection in the CPS domain. For example, Kravchik and Shabtai [96] demonstrate how 1-D CNNs identify cyber-attacks in the SWAT dataset. Similarly, Wang et al. [186] use TCNs for anomaly detection in satellite telemetry data, Neupane et al. [129] for anomalies in automotive vehicle sensor data, and Ma et al. [111] to identify false data injections in smart grid data. The categorization into recurrent or convolutional architectures is not mutually exclusive, there are several papers that propose NN architectures that use combinations of CNNs and RNNs [120, 199]. TCNs are also used in transformer-based architectures designed for anomaly detection in CPS [77], which will be presented in the next section.

Transformer-based models have also gained attention and popularity in the CPS literature in recent years because their attention mechanism makes them perform well at modeling long-range dependencies in sequential data. Public interest in transformer-based Large Language Models (LLMs) and the chatbots powered by them [39, 74, 182] has certainly supported this trend. There are several examples of transformer-based anomaly detection models for CPS, including the TimesNet model proposed by Wu et al. [195], which transforms 1D time series into 2D tensor representations to encode complex temporal patterns as well as the model proposed by Jeong et al. [87], which is inspired by the popular BERT model [39]. Both papers validate their approaches on CPS datasets. Transformer-based architectures are combined with Graph Neural Networks (GNNs) for prediction-based anomaly detection in CPS datasets [29].

Graph neural networks (GNNs) are used in the context of CPS predictions to model the relationships between different sensors or components as graphs (see Section 2.2.1). For example, Deng and Hooi [38] propose a model that first learns a graph from the time series data that models their characteristics in order to use this graph for predicting system behavior. Similarly, Dai and Chen [37] combine methods for learning graphs from data with normalizing flows [148] to detect anomalies. Wang et al. [185] present an interesting transformer-based GNN approach in which the interactions

between sensors are dynamically adjusted over time. All of the above studies on GNNs validate their methods against public CPS datasets.

Despite major advances in prediction-based anomaly detection, all of the approaches above share a disadvantage that lies in the nature of predictions itself: if the system is subject to changes that occur without prior indication through patterns within the time series, they cannot be accurately predicted [65]. If such cases occur when no anomalies are present, which can be the case, for example, with external influencing factors or changes in control variables, there is a high probability that the anomaly detection model will generate false positives.

Reconstruction-based Methods

Reconstruction-based methods for anomaly detection are not based on errors in the prediction quality, but on the principle of reconstruction errors described in Section 2.1.4. They are therefore particularly advantageous in for CPSs, where unpredictable state changes are common.

Autoencoders (AEs), which are introduced in detail in Section 2.1.3, are the most commonly used among reconstruction-based anomaly detection models. The AEs proposed in the CPS literature use all of the NN layer architectures discussed in the previous section in the encoder and decoder NNs. RNN-based AEs are applied, for example, in anomaly detection for industrial furnaces [143], electrical power grids [176], and IoT devices in general [132]. Similarly, convolutional AEs have been employed for detecting anomalies in photovoltaic systems [135] and machine sound signatures [34]. Campos et al. [24] proposed a convolutional autoencoder with attention mechanisms that was validated against multiple CPS datasets. More recently, even transformer-based AEs have been applied to CPS datasets, as demonstrated by Kang and Kang [91], Shang et al. [160], and Tuli et al. [181].

Variational Autoencoders (VAEs), which are explained in more detail in Section 2.1.5, are also implemented with various NN layer types. Li et al. [105] designed a hierarchical VAE architecture with two sets of latent variables, where one models the inter-metric dependencies (relationships between different signals) and another capturing temporal dependencies (patterns within each individual signal over time). Lin et al. [107] and Wu et al. [194] introduce the LSTM-VAE and the VAE-TCN respectively, which they demonstrate on multiple CPS sensor datasets. More recently, Min et al. [122] introduced a variational transformer that applies the VAE framework to anomaly detection in autonomous vehicle data.

Generative adversarial networks (GANs) consist of a generator model that creates fake data samples and a discriminator model which is tasked with distinguishing them from real samples [67]. In contrast to AEs, Generative Adversarial Networks (GANs),

just like VAEs, are generative models. Several GAN-based anomaly detection models are proposed in the CPS literature. Li et al. [102], for example, use LSTMs in both the generator and discriminator models and a combination of reconstruction error and discrimination in the loss. Chen et al. [28] present a framework that combines GANs with VAEs. They use two generators and two discriminators, one pair operating in the data space and the other in the latent space. In a more recent work, Du et al. [45] introduce FGANomaly, which attempts to solve the problem of unlabeled anomalies in the training dataset using a filter mechanism and a specially designed loss function. Although GANs show promising results in anomaly detection in CPS data, the models are sometimes difficult to train [118], which is why they are less commonly used in practice.

Reconstruction-based anomaly detection methods should generally deliver better results on complex CPS data than prediction-based methods because they do not have to make predictions and are therefore better able to cope with unpredictable but not anomalous cases. However, state transitions in CPS are not only partially unpredictable but also often rare, as seen in the ECLSS use case. Due to this imbalance, the models tend to ignore the rare events in favor of a better fit of the steady-state data, which can lead to false positives.

3.1.2 Subsystem-Level and Structurally-Informed Approaches

The methods presented in Section 3.1.1 operate largely as “black boxes” that view the CPS as a single entity or as a collection of signals. However, as described in the introduction, CPSs can be divided into subsystems, components, or processes that follow physical or causal rules. This type of prior knowledge can integrate valuable context into ML models that could improve both the performance and interpretability of the results. The current literature on CPS contains several contributions that investigate this integration, using different approaches.

Modular Neural Network Architectures

One approach to integrating structural knowledge into ML models is the use of modular NNs. The approaches model the behavior of individual components or subsystems of the CPS using individual NNs. For example, Ehrhardt et al. [48] propose modular NNs that approximate the functions represented by each subsystem. The entire system can then be modeled by connecting the inputs and outputs of the models according to the CPS structure, allowing both local and global anomalies to be detected. In a similar approach, Vranješ and Niggemann [184] demonstrate how modeling at the subsystem level, rather than modeling the entire system, can improve performance in anomaly detection. Finally, Wu et al. [197] propose an interesting approach in which GNNs are divided into predefined subgraphs based on prior knowledge, which are defined along signal types or components.

Physics-Informed Constraints

Probably the most well-known approach to integrating prior knowledge into NNs for time series analysis are Physics-Informed Neural Networks (PINNs). These models introduce physical constraints into the loss function or model architecture. For example, Wu et al. [196] use prior knowledge from piping and instrumentation diagrams to modify the loss function of a GNN so that the model favors graphs that are similar to the physical structure. In addition, both Misyris et al. [123] and Falas et al. [52] implement the standard PINN framework, in which the loss function is extended with physical constraints, for CPS use cases.

Expert Knowledge Integration

In addition to the traditional approach of PINNs, there are also several other examples of how expert knowledge can be integrated into NN architectures. Raman and Mathur [144] present an approach that is very similar to modular NNs. The authors use knowledge about process invariants from the system documentation to define the inputs and outputs of different NNs. Saez et al. [153] present another example for manufacturing systems, where physical models of the individual process phases are combined with data-driven techniques. Different anomaly detection models and threshold values are used for different process phases. However, the integration of expert knowledge and physical modeling goes so far that the use of ML is very limited. A more recent and promising approach was presented by Westermann et al. [189], who use formal knowledge graphs and ontologies in combination with data-driven methods. The goal of their approach is to increase the interpretability of anomaly detection models. Their method first learns a state automaton based on the sequences of actuator signals and then enriches them with information from the knowledge graphs. In this way, identified anomalies can be enriched with information that is understandable to engineers.

Despite the promising results achieved by the approaches integrating structural knowledge into ML models, a number of challenges remain. On the one hand, the formalization effort for most models is still relatively high and requires manual work by system experts for implementation. On the other hand, excessive restrictions on prior physical knowledge could limit the ability of the models to learn patterns from data. Thus, this trade-off needs to be balanced. Furthermore, most models work at the signal level, and there is still little focus on subsystem or component level approaches.

3.2 Machine Learning Approaches for Diagnosis in Cyber-Physical Systems

According to Escobet et al. [50], a diagnosis, which requires that faults have first been detected, is “the set of components that explain the fault”. As already men-

tioned in Section 1.1.2, traditional diagnostic methods for CPSs require extensive prior knowledge and high modeling efforts. This section therefore reviews approaches that address this problem by combining diagnostic methods with data-driven ML techniques.

3.2.1 Supervised Learning for Fault Classification

The most intuitive approach to creating data-driven diagnoses is to use supervised learning methods. An example of this is the patent by Steude and Demeschkin [166], in which the system status of an aircraft component is classified into different categories. These include a variety of known faults, but also normal operation and an abnormal but unknown status. In a similar approach, Zaman et al. [207] use DL models to classify faults in industrial processes. Xia et al. [198] propose a more sophisticated modeling approach, which they call residual-hypergraph CNN, to perform this classification. In their model, unsupervised relationships between variables are first modeled using prior knowledge as a graph to compute more informed classifications. The substantial number of publications in this area is summarized in two relatively recent reviews. Mian et al. [121] consider the state of the art of ensemble models for fault classification use cases, while Cen et al. [26] describe the use of ML and transfer learning approaches for this task.

As already mentioned in the introduction to this thesis, these methods are very good at recognizing errors based on their patterns in the data, but they require the availability of labels for these errors. This in turn means that previously unseen error cases cannot be identified [50]. For this reason, there is research on methods that use unsupervised learning.

3.2.2 Diagnostic Insights from Unsupervised Anomaly Detection Models

The CPS literature contains methods that modify purely data-driven DL approaches to provide information relevant for diagnosis. Although some of the authors of these papers refer to their methods as diagnoses, from the perspective of the author of this thesis, these approaches only localize symptoms in the systems rather than diagnose them. Two examples are the works of Garg et al. [61] and Zhang et al. [209], which present DL models that not only detect anomalies in the overall system, but also localize them by determining which signals deviate from the normal state. Dai and Chen [37] present a more sophisticated approach that first uses normalization flows to model the relationships between the signals. The method outputs anomaly values for individual time series. While these methods may represent first steps toward diagnosis, they do not determine the causes.

3.2.3 Integrating Data-driven Methods with Diagnostic Reasoning

This subsection discusses approaches combining data-driven methods with diagnostic reasoning methods, which have been developed to overcome the disadvantages the approaches have when used in isolation.

Hybrid Model-Based and Data-Driven Approaches

ML and model-based diagnostic reasoning have been combined in two ways. Either data-driven methods are used to generate inputs for model-based diagnostic reasoning models, or conversely, the outputs of model-based methods are further processed in ML models.

In the first category, there are some works in which residual values generated by ML procedures were first binarized and then used as observations of a CBD [22, 41, 124]. These approaches retain the capability of root causes analysis, but reduce the effort required to create the models. Mohammadi et al. [126] use a more complex modeling approach, which they call “grey box recurrent NNs”, that enriches the ML model for residue extraction with prior knowledge about the physical structure of the system. Similarly, Jung et al. [89] design a set of RNN-based residual generators using a structural model of a heavy-duty truck’s urea injection system.

Conversely, some papers present methods that analyze the residuals from model-based methods in downstream ML models. For example, Pesola et al. [141] use a relatively simple model-based method to generate the residual values and handle the uncertainties and complexities that the model does not cover using downstream ML models. Jung et al. [90] present a similar approach that supports the classification of multiple faults even if only single-fault training data is available. Lundgren and Jung [110] also present a ML model that analyzes residual values for diagnosis, but considers them to be given.

These hybrid approaches demonstrate the potential in the combination of symbolic and sub-symbolic AI approaches for the diagnosis of technical systems, but challenges remain. On the one hand, expert knowledge and modeling effort are still required for implementation. On the other hand, the models discussed only consider diagnosis at the signal level, in some cases also at the component level, but do not investigate the aggregation of the diagnosis problem at the subsystem level.

Graph-based Diagnostic Reasoning

Graph-based diagnostic methods are well suited to represent complex fault propagation paths in CPSs due to the intuitive representation of cause-effect relationships. Since one of the contributions of this thesis is a graph-based diagnostic reasoning algorithm, this subsection provides a dedicated overview over related approaches.

Graph-based diagnostic reasoning fundamentally relies on the concept of causality. Causal graphs usually follow the idea of causality defined by Forbus [54], which

is formally defined as value propagation between entities. An early example of these graphs used for diagnosis is proposed by Stumptner and Wotawa [174]. However, their approach suffers from the limitation that only tree-structured graphs can be used to model the causality which don't allow cycles. Some more recent papers focus on extracting causal graphs from time series data. For example, Grünbaum et al. [72] and Runge et al. [152] propose methods that extract causal graphs from time series of CPS and natural science, respectively. In addition, Silva et al. [163] has recently demonstrated the feasibility of extracting capability ontologies from text-based system descriptions using LLMs, suggesting that fault propagation graphs can be created in a similar way.

The availability of the methods described in this section supports the assumption that causal graphs such as those used as input in the method proposed in Section 5.1 can be created from data or system descriptions if they are not already available. Similar causal graphs are also described in Weber and Wotawa [188] or Bozzano et al. [20].

Several approaches have been proposed that use graphs learned directly from data for anomaly detection. The model proposed by Yang et al. [201], for example, learns a causal graph from time series data in order to subsequently identify system modules to detect and localize anomalies in them. In a similar solution for detecting cyber attacks in control systems, Koutroulis et al. [95] train causal models on normal operation data to subsequently identify the affected variables based on prediction errors. Like the models presented in Section 3.2.2, however, these methods do not provide a diagnosis in the sense of root cause analysis, rather, they localize the symptoms. Furthermore, they assume that the causality to be modeled can be derived entirely from the observations in the time series.

However, there are also approaches that use prior knowledge formalized as graphs explicitly for diagnosis. One example is the contribution by Zhang et al. [210], in which graph-based diagnosis is applied to a power system. The approach uses abductive reasoning to identify candidates for faulty components by tracing symptoms backwards through the graph and then performing an evidence theory assessment. However, the approach assumes that symptoms on components are already present and do not have to be derived from observations.

With EasyRCA, Assaad et al. [4] presented a method that identifies root causes of anomalies in multivariate time series. The method uses the temporal order of the identified anomalies and identifies those anomalies whose occurrence cannot be explained by the occurrence of another earlier anomaly. However, unlike the method presented in this thesis, this method only works on acyclic causal graphs.

Another approach using fault propagation graphs defined by experts as input is presented by Rehak et al. [145]. Similar to the approach presented in this thesis, the authors use graphs that model the fault propagation paths between components. Identified anomalies are then analyzed for diagnostic purposes using this graph. However, in the graphs used in their paper, root causes are also explicitly represented as nodes, and the method can therefore only be applied to predefined faults.

3.2.4 The Role of Abstraction in Diagnostic Methods

One of the key questions in the development of diagnostic systems for complex CPSs is at which aggregation level the system should be diagnosed [94]. Most of the approaches mentioned above aim to identify the signals or components responsible for the faults. The abstraction of the problem to the subsystem level, that is the search for the subsystem that causes the fault and not the components contained in it, could reduce the modeling effort, which of course limits the granularity of the diagnosis.

This concept of abstraction has been studied in the model-based diagnosis community. Chittaro and Ranon [31], for example, have formalized the concept structural abstractions for diagnosis and demonstrated that the aggregation of components into subsystems reduces the complexity of multi-fault diagnosis. Similarly, Perrot and Travé-Massuyes [140] have formulated the diagnosis as an optimal constraint satisfaction problem and show how well-chosen abstractions can reduce the computational complexity. In addition, Lamperti and Zhao [98] present a higher-order discrete-event system and a corresponding diagnosis method that makes use of different levels in the system hierarchy so that not every component needs to be modeled in detail.

These methods have been developed before the recent advances in data-driven and hybrid diagnostic methods. As a result, they focus primarily on traditional model-based reasoning and do not make use of ML approaches, which might further reduce the required knowledge engineering effort.

3.3 State of the Art in MLOps

The deployment and operation of ML-based software solutions such as the anomaly detection and diagnosis methods mentioned in this chapter involve additional complexity, as motivated in Section 1.1.4. To contextualize the MLOps-related contribution of this thesis, the remainder of this chapter provides an overview of the current state of research in MLOps and puts a special focus on cloud-native solutions and their application in the field of CPSs.

3.3.1 Evolution of MLOps Frameworks and Platforms

The MLOps landscape has evolved rapidly in recent years, with a trend from individual tools to entire MLOps platforms. Recent literature reviews on MLOps [15, 97, 150, 177] indicate that the community is largely converging on the basic principles and feature requirements (see Section 2.3). However, the practical implementations remain fragmented and for many of the functional requirements of MLOps there are many partially competing tools. As highlighted by Berberi et al. [15], the MLOps market is still growing, both in terms of open-source and commercial solutions.

3.3.2 Cloud-Native Infrastructure and MLOps Platforms

The development of MLOps is closely linked to the progress made in cloud-native technologies. Kubernetes, as the core technology behind the cloud-native concept, is the underlying infrastructure for many production grade ML systems, reflecting the general trend towards more cloud computing [25, 158]. This also applies to modern LLMs applications. For example, OpenAI—the company behind ChatGPT—has deployed a Kubernetes cluster with thousands of nodes to train LLMs in a massively parallelized and GPU accelerated way [133]. Tools such as Slurm [203] are still widely used, especially in research, but Kubernetes has become the standard in production environments, particularly due to its container orchestration capabilities, declarative configuration approach and extensive ecosystem of extensions. Kubernetes is also increasingly being used in research, as implementations such as its use at the CERN [66] data center indicate.

On this basis, several MLOps platforms have been developed that extend Kubernetes' feature set by the functionalities introduced in Section 2.3, in order to support the whole ML lifecycle. Among the cloud-native MLOps platforms, Kubeflow stands out. It ranks first in a recently published comparison of open-source MLOps platforms by Berberi et al. [15], where the authors compile a weighted score that considers both popularity and feature completeness. Another indication of Kubeflow's popularity and maturity is that the platform has been promoted as an incubation project by the Cloud Native Computing Foundation (CNCF), the global organization for standardizing cloud-native technologies [56].

The individual components within the Kubeflow ecosystem have been used and demonstrated in various scientific studies, but in most cases in isolation and not in combination with other tools from the ecosystem. For example, George et al. [62] introduce Katib, Kubeflow's hyperparameter optimization framework, and demonstrate the hyperparameter tuning features with multiple model architectures and datasets. Xu et al. [200] present an automation of a data preprocessing service for power grid applications implemented on Kubeflow Pipelines (KFP), Kubeflow's workflow engine [9]. The implementation of a ML algorithm for a radio access network discussed by Subramaniam and Subramaniam [175] uses KFP as well as Kserve, Kubeflow's model serving component [7]. Beyond industry applications, Kubeflow has found adoption in research domains as well. Golubovic and Rocha [66] describe a Kubeflow deployment at CERN used for high-energy physics applications and Granlund et al. [69] for healthcare studies.

Like the contributions mentioned above, the official Kubeflow documentation [8] primarily describes individual components rather than their integration in the entire workflows. However, the experience of implementing the algorithms presented in this thesis in Kubeflow has shown that it is precisely this integration that poses technical challenges.

3.3.3 Challenges for MLOps in Diagnostic Applications

For ML use cases for CPS, two MLOps topics are particularly relevant. The first one is concept and data drift which, according to Cody et al. [33] and Spotorno Bieger et al. [164], play a major role for CPSs. Physical systems naturally change through wear and maintenance measures, which may change the distributions of their sensor readings. Although the need for features that solve these problems has been recognized, no standard tools have yet been established in this area, and many MLOps platforms lack their support completely.

Data processing presents another significant challenge in the CPS domain. Tappe et al. [178] describe the requirements for real-time inference systems for streaming data. The size of the data often requires the application of out-of-core frameworks such as Spark [206] and Dask [149], which add another layer of complexity. Furthermore, the integration of these technologies into a coherent MLOps workflow has not been explored sufficiently, particularly in the context of CPS diagnostics.

3.4 Research Gaps and Opportunities

The literature review in this chapter reveals several gaps that motivated the research questions described in Section 1.2. Despite the developments in the areas of anomaly detection and diagnosis discussed in the sections above, it is still challenging to develop automated diagnostic software for a complex system such as the ECLSS. Most current approaches face limitations in one of three areas: (i) they require substantial expertise and modeling efforts to build and maintain, (ii) they depend on labeled fault data, or (iii) they identify symptoms rather than root causes. The integration and deployment of these approaches into production environments introduces additional challenges.

In the area of anomaly detection, most methods operate either at the signal level or at the system level. Only a few papers study anomaly detection on the intermediate subsystem level in the system hierarchy. Reconstruction-based approaches such as TCN-AE [116] and TCN-VAE [194], which are most similar to the model proposed in this work, do not integrate any prior knowledge about the structure of the CPSs. Methods that do integrate this type of information, such as the modular NN presented by Vranješ and Niggemann [184], only mention diagnosis as a potential downstream task without further investigating or evaluating it. Thus, the challenge of designing NN architectures that recognize symptoms at the subsystem level in CPSs has not been sufficiently addressed, which motivates RQ1, asking how subsystem-level health states suitable for diagnostic algorithms can be generated by enhancing data-driven anomaly detection methods with prior knowledge.

Regarding diagnostic methods, supervised approaches require labeled fault datasets that are (fortunately, from a safety perspective) often unavailable for safety-critical systems. Therefore, many purely data-driven unsupervised methods have been proposed [61], which, however, only identify and localize symptoms, but do not perform the actual diagnosis. Model-based methods, as for example the method

proposed by Diedrich and Niggemann [41], require detailed behavioral models that are hard to develop for large-scale systems such as the ECLSS. Existing graph-based diagnostic approaches, such as the one proposed by Assaad et al. [4], often impose restrictions on the graph structure (e.g., acyclicity). These limitations motivated RQ2, which explores how subsystem-level symptoms and structural information about fault propagation paths can be combined for diagnostic reasoning without requiring detailed behavioral models.

In the field of MLOps, the fragmented tool landscape poses a challenge for practitioners and researchers because there are no uniform standards. Furthermore, reviews such as the one by Berberi et al. [15] and other contributions in this field focus primarily on individual tools and their features rather than their integration into end-to-end workflows. This motivated RQ3, which investigates how MLOps tools and practices can support the implementation and maintenance of anomaly detection and diagnostic software in production environments.

The contributions of this thesis, described in Section 1.3, aim at addressing these research gaps. The following chapters present a framework that aims to overcome these limitations while maintaining practical applicability.

Part II

Formal Definition and Solution

4 Formal Definition

Chapter Outline This chapter formalizes the diagnostic problem addressed in this thesis as well as the required inputs and expected outputs. In addition, the research questions described in Section 1.2 are revisited and restated using this formalization. The notation established in this chapter will then be used in the subsequent chapter to describe the solution approach.

4.1 Basic Notation and Definitions

As motivated in Section 1.2, a CPS can be modeled as a non-empty finite set of subsystems $S = \{s_1, \dots, s_{|S|}\}$. Furthermore, there is a non-empty set of signals $P = \{p_1, \dots, p_{|P|}\}$ where each $p \in P$ corresponds to measurements from a sensor, actuator states, or any other signal that describes the state of the CPS. $T = \{t_1, t_2, \dots, t_{|T|}\} \subset \mathbb{N}$ describes the sorted set of discrete time points at which the signals were sampled, such that $t_i < t_{i+1}$ for all $i \in \{1, 2, \dots, |T| - 1\}$. For each signal $p \in P$, the corresponding time series is written as \mathbf{x}_p , where each entry $x_p(t) \in \mathbb{R}$ holds the value at timestamp $t \in T$. Stacked together, these time series form the matrix $\mathbf{X} \in \mathbb{R}^{|T| \times |P|}$, where each row corresponds to a time point $t \in T$, and each column corresponds to signal $p \in P$. For a simple example system, these quantities are visualized in Figure 4.1.

Sequence models do not process individual points in time, but rather time windows. Thus, let $\Delta t_w \in \mathbb{N}$ denote the corresponding window size parameter, where the subscript w indicates “window”. For any time index i with $i \geq \Delta t_w$, the matrix $\mathbf{X}_{i_i} \in \mathbb{R}^{\Delta t_w \times |P|}$ represents the collection of signal readings for the Δt_w most recent time steps up to and including time t_i .

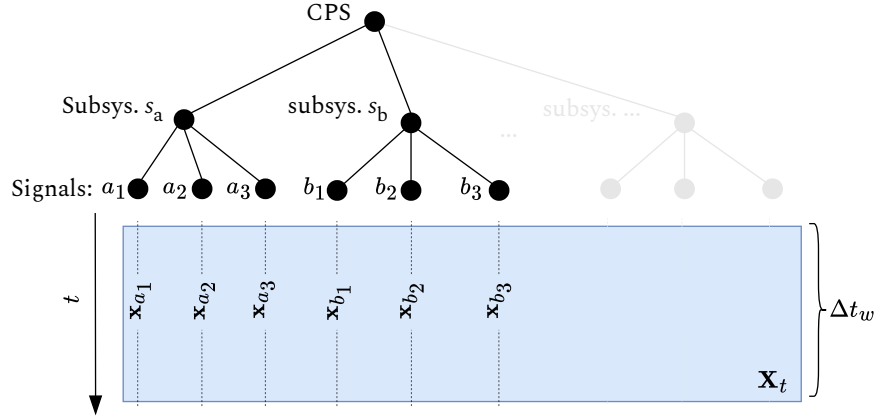


Figure 4.1: Visualization of a simple CPS example with hierarchical organization. The example shows two subsystems $S = \{s_a, s_b\}$, their signals $\{a_1, a_2, a_3, b_1, b_2, b_3\}$, and the time series data matrix \mathbf{X}_t , which contains the most recent sensor readings. The figure also visualizes the time window Δt_w used for sequence-to-sequence modeling.

4.2 Problem Statement

Using the notation described above, the diagnostic problem that this thesis aims to solve can be defined as follows: For a given point in time t , both the set of symptomatic subsystems $S_{\text{sym}}(t) \subseteq S$ and the set of subsystems $S_{\text{causal}}(t) \subseteq S$ must be identified, whose abnormal behavior is the most likely cause of all symptoms observed in $S_{\text{sym}}(t)$. Note that this formulation differs from that of classic diagnosis problems in that it does not search for the complete set of all possible or minimal diagnoses, but rather the single most probable one. The description of the solution in Chapter 5 and the discussion in Chapter 8 address this point in detail.

The solution uses prior information on fault propagation paths between subsystems and a mapping between subsystems and their associated signals, both formalized in the following subsection. Additionally, the approach requires only training data from nominal system operation.

4.2.1 Required Inputs

Although this thesis aims to develop a diagnostic approach that minimizes the necessary prior knowledge, the proposed method does not work entirely without input. The following paragraphs describe the three inputs that are necessary to implement the proposed method.

Causal Subsystem Graph As already indicated in the introduction, the proposed diagnosis framework requires knowledge of the structure of the CPS and basic information on the causal relationships between the subsystems as input. This prior

knowledge is formalized in the form of a directed graph $G = (S, E)$, whose vertices represent the subsystems and whose edges $(s_i, s_j) \in E \subseteq S \times S$ represent fault propagation paths, such that a fault in s_i can propagate to and affect subsystem s_j . Since directed graphs are an intuitive way of representing fault propagation paths in CPS, they have already been used in related work [4, 145, 174]. Because diagnostic reasoning is performed using this graph G , the granularity of the vertices $s \in S$ in the system hierarchy also corresponds to the granularity of the diagnosis. Each subsystem $s \in S$ represented by the nodes in G is typically monitored using multiple signals from the set P . The assignment of these signals to the subsystems is another input that will be described in the next paragraph.

Subsystem-Signals Map The subsystem-signals map associates each subsystem with its corresponding set of signals. More formally, the map can be written as $\mathcal{M} : S \rightarrow 2^P$, so that each subsystem $s \in S$ is mapped to a set of signals $P_s \subseteq P$ from the powerset of the set of all signals $\mathcal{P}(P) = 2^P$. In the example shown in Figure 4.1, the mapping would be $\mathcal{M}(a) = \{a_1, a_2, a_3\}$ and $\mathcal{M}(b) = \{b_1, b_2, b_3\}$. Note that \mathcal{M} explicitly allows signals to overlap across subsystems, as a single sensor might provide information relevant to multiple functional units. This happens for example in cases where a signal is the input of one subsystem and the output of another.

Training Data from Nominal System Operation The third and final input is the training data $\mathbf{X} \in \mathbb{R}^{|T| \times |P|}$ as defined in Section 4.1. Ideally, these should contain all nominal operating states of the CPS and be fault-free. In contrast to the supervised diagnostic approaches discussed in Section 3.2.1, the approach presented in this thesis does not require labeled fault data.

4.2.2 Expected Outputs

According to the problem description at the beginning of this section, the diagnostic method must generate two outputs:

Health State Vector and Symptomatic Subsystems The first output of the diagnostic framework is a health state vector $\mathbf{h}(t) = (h_s(t))_{s \in S}$ of the CPS at time t , where $h_s(t) \in \{0, 1\}$ for each subsystem $s \in S$. The binary value $h_s(t)$ indicates whether subsystem s behaves abnormally, with $h_s(t) = 0$ signifying “OK” (normal operation) and $h_s(t) = 1$ signifying “not OK” (anomalous behavior). The set of symptomatic subsystems $S_{\text{sym}}(t) \subseteq S$ contains those subsystems s for which $h_s(t) = 1$:

$$S_{\text{sym}}(t) = \{s \in S \mid h_s(t) = 1\}. \quad (4.1)$$

This set represents all subsystems exhibiting anomalous behavior at time t . Note that these symptomatic subsystems are not necessarily part of the diagnosis.

4 Formal Definition

Causal Subsystems The second and more important output is the set of causal subsystems $S_{\text{causal}}(t) \subseteq S$, which contains those subsystems whose faults are the most probable explanation for the observed symptoms $S_{\text{sym}}(t)$. A subsystem $s \in S$ is considered a potential element of the set $S_{\text{causal}}(t) \subseteq S$ if there is a path from s to some symptomatic subsystem $s' \in S_{\text{sym}}(t)$ in the causal graph G . Formally, for the set of candidate nodes C this means:

$$C = \{s \in S \mid \exists s' \in S_{\text{sym}}(t) : \text{path}(s \rightarrow s') \text{ in } G\}. \quad (4.2)$$

$S_{\text{causal}}(t)$ is a subset of this set of candidates that together represent the “best explanation” for the observed symptoms $S_{\text{sym}}(t)$. What exactly constitutes a “good explanation” in terms of the diagnosis is described in detail in Chapter 5. At a high level, the selection process considers factors such as the number of symptoms each candidate can explain, the proximity of candidates to symptomatic subsystems in the causal graph, and whether candidates themselves exhibit symptoms. Note that $S_{\text{causal}}(t)$ may include subsystems that are not symptomatic themselves, if they represent good candidates for explaining the observed symptoms according to these criteria. Not all faults manifest themselves as a detectable anomaly in the time series of the subsystem.

4.3 Relation to Research Questions

The formal descriptions of the inputs and outputs in the preceding paragraphs enable a more formal description of the research questions introduced in Section 1.2.

RQ1 (Formal): Given the training data \mathbf{X} from nominal system operation and subsystem-signals map \mathcal{M} , can a function $\mathcal{G} : \mathbb{R}^{\Delta t_w \times |P|} \rightarrow \{0, 1\}^{|S|}$ be developed that maps time series windows to binary health state vectors $\mathbf{h}(t)$ such that the subsystems whose corresponding values in \mathbf{h} equal 1 are those exhibiting symptoms?

RQ2 (Formal): Given health state vector $\mathbf{h}(t)$ and causal subsystem graph G , is there an algorithm $\mathcal{A} : \{0, 1\}^{|S|} \times (S, E) \rightarrow 2^S$ that computes the most probable causal subsystem set $S_{\text{causal}}(t)$ without requiring detailed behavioral models of individual subsystems?

RQ3 (Semi-Formal): Can the deployment, monitoring, and maintenance of functions such as \mathcal{G} and algorithm \mathcal{A} be automated using modern MLOps platforms within a unified workflow?

A solution on the basis of which these questions can be answered with “yes” should ideally fulfill a series of requirements that can be used as a guideline for the experiments described in Chapter 6. With regard to RQ1, the function \mathcal{G} should be

able to accurately localize anomalies at the subsystem level and outperform conventional univariate approaches and those that work on the entire system level. As for RQ2, algorithm \mathcal{A} should be able to compute reasonable diagnoses for symptoms in cyclic and non-cyclic graphs in polynomial time. The MLOps implementation (RQ3) should follow typical best practices from the discipline, be fully automated, and address the CPS-specific challenges described in Section 5.4.3.

5 Solution

Chapter Outline This chapter describes the core contributions of this thesis: a diagnostic method for CPS that addresses the three research questions formalized in the previous chapter. The chapter is structured as follows: Section 5.1 provides a general overview of the approach and its phases. Section 5.2 presents the components of the proposed implementation of function \mathcal{G} , namely a novel neural architecture used for subsystem-level anomaly detection and techniques to binarize its outputs. Section 5.3 describes the graph-based diagnosis algorithm \mathcal{A} . Finally, Section 5.4 covers the MLOps implementation of the entire framework.

5.1 Overview

The overall framework proposed in this thesis is visualized in Figure 5.1 and can be divided into three phases, which are described at a high level in the following paragraphs. A more detailed discussion of the individual components will be presented in the following sections of this chapter.

Phase a) Knowledge formalization In the first process phase, the inputs required to implement the method, namely the causal subsystem graph G and the subsystem-signals map \mathcal{M} , need to be formalized. As argued in the introduction, both inputs are often already available in practice or can be extracted from the system documentation with relatively little effort. For this reason, the following technical descriptions emphasize phases b and c, which are also described in Algorithm 1, and take G and \mathcal{M} as given.

Phase b) Model training The first step in phase b is the design of the NN architecture based on the subsystem signals map \mathcal{M} . The design, which is unique to every CPS, aims to localize anomalies at the subsystem level, which is a level that lies between that of individual sensors and the entire system and aligns with the graph G and the mapping \mathcal{M} (see Figure 1.2 and Figure 1.3). These architectural details are described in the following section. The model is trained with data collected during the nominal operations of the system (step b.1 in Figure 5.1). Because the outputs of the NN are continuous valued residuals, a second model is applied that transforms them into the binary health states \mathbf{h} necessary for diagnosis, which classify the status of each subsystem as either “OK” or “not OK” (Step b.2 in Figure 5.1). Together, these two components form the symptoms generator, which implements the function \mathcal{G} formalized in Chapter 4 and described in detail in the following section.

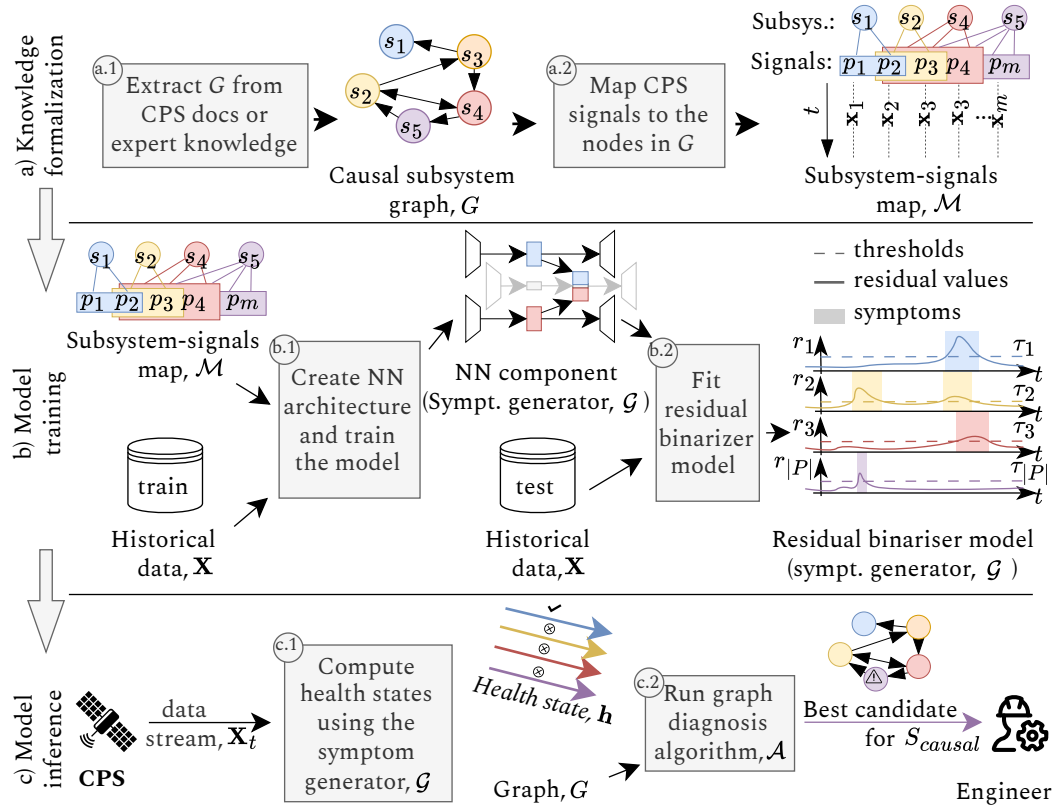


Figure 5.1: Overview of the three phases of the proposed diagnosis framework. The diagnostic framework contains three phases: a) knowledge formalization, where the required prior knowledge is extracted from system documentation, b) model training, where the symptoms generator \mathcal{G} is trained on nominal data, and c) model inference, where both the symptoms generator \mathcal{G} and the diagnosis algorithm \mathcal{A} are applied to the streamed CPS data.

Phase c) Model Inference Finally, phase c describes the application of the overall framework. The symptoms generator is continuously applied to the stream of sensor data and computes a health state vector $\mathbf{h}(t)$ for each time point based on the signal values of the last Δt_w time points. As soon as a symptom is detected, which means that $\exists s \in S : h_s(t) = 1$, the graph diagnosis algorithm \mathcal{A} is executed (Step c.2 in Figure 5.1), which calculates the most probable diagnosis based on the graph G .

Having introduced the overall concept, the following sections provide details on the two key contributions developed in this thesis: the symptoms generator \mathcal{G} and the graph diagnosis algorithm \mathcal{A} .

Algorithm 1 Overall Diagnostic Process (phases b and c only)

Require: Training data \mathbf{X} , subsystem-signals map \mathcal{M} , causal subsystem graph G , window size Δt_w

Ensure: Diagnostic results over time

- 1: // Phase b) Model Training
- 2: $\mathcal{G} \leftarrow \text{TRAINSYPMTGEN}(\mathbf{X}, \mathcal{M})$ ▷ Section 5.2
- 3: // Phase c) Inference
- 4: **while** system is running **do**
- 5: // Step c.1: Apply the symptoms generator
- 6: $\mathbf{X}_t \leftarrow \text{GETDATAWINDOW}(\Delta t_w)$ ▷ Latest sensor data
- 7: $\mathbf{h}(t) \leftarrow \mathcal{G}(\mathbf{X}_t)$ ▷ Symptomatic subsystems
- 8: // Step c.2: Apply the graph diagnosis algorithm
- 9: **if** symptoms detected in $\mathbf{h}(t)$ **then**
- 10: $S_{\text{causal}}(t) \leftarrow \mathcal{A}(G, \mathbf{h}(t))$ ▷ Section 5.3
- 11: **return** $S_{\text{causal}}(t)$ ▷ Root cause subsystems
- 12: **end if**
- 13: **end while**

5.2 Symptoms Generator

As indicated in the overview above, the symptoms generator implements the function \mathcal{G} defined in Section 4.3, which maps a time window of the multivariate time series data to binary health states for each subsystem. The technical implementation of this symptoms generator is split into two steps: first, a NN computes continuous residual values for each subsystem, which are transformed into binary health states by the residual binarizer in a second step. The following subsections describe both of these components in detail.

5.2.1 Neural Network Architecture

The neural architecture of the NN component within the symptoms generator uses the β -VAE [78] framework and a modified version of TCNs [12]. This makes it similar to the architecture proposed by Meng et al. [116] in its basic features, but there are two novelties of the architecture proposed in this work.

The first of which is the integration of structural knowledge formalized in the subsystem-signals map \mathcal{M} . Through this integration, for each subsystem in S used in the graph G , the model implements a dedicated encoder and decoder pair. Each of these pairs processes only those signals that are assigned to the corresponding subsystem s by the subsystem-signals map $\mathcal{M}(s) \subseteq P$. This concept is visualized in Figure 5.2 using only two example subsystems ($S = \{a, b\}$). The grayed encoders indicate that this approach scales to all subsystems in S .

From the signals of each subsystem $i \in S$, the encoders compute a dedicated latent space representation $\mathbf{z}_i \in \mathbb{R}^{m_i}$, with $m_i \in \mathbb{N}$ representing the size of the latent

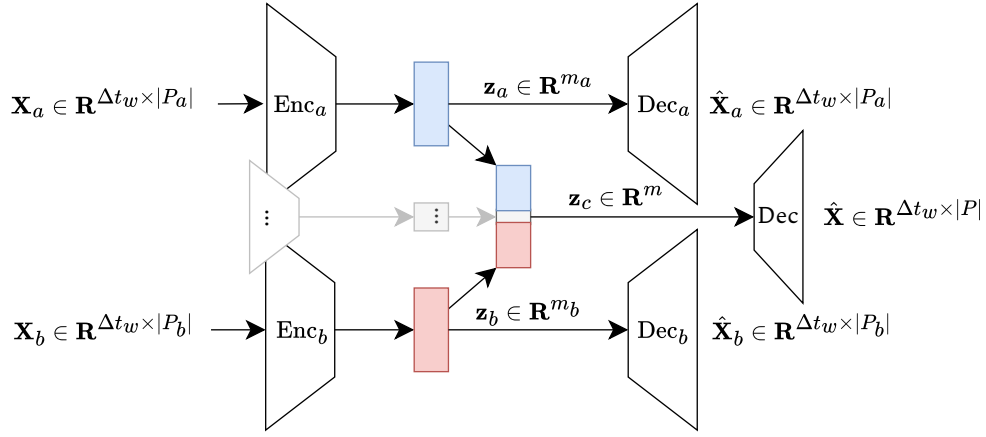


Figure 5.2: NN model architecture highlighting the concept of the composite latent space. In the visualization, only two subsystems ($S = \{a, b\}$) are shown as examples while potentially many more are indicated in gray. The subsystem-signals mapping \mathcal{M} defines which signals are modeled by the encoder-decoder pair for subsystem s . While each subsystem receives its dedicated latent space representation (here \mathbf{z}_a and \mathbf{z}_b), these are also combined into the composite latent space \mathbf{z}_c . In addition to subsystem-specific reconstructions $\hat{\mathbf{X}}_a$ and $\hat{\mathbf{X}}_b$, this composite latent space is used by a comprehensive decoder to reconstruct all signals in P in the matrix $\hat{\mathbf{X}}$.

space vector of subsystem i . These subsystem-specific latent vectors are then concatenated into the composite latent space $\mathbf{z}_c \in \mathbb{R}^m$, with $\mathbf{z}_c = [\mathbf{z}_1; \mathbf{z}_2; \dots; \mathbf{z}_{|S|}]$ and $m = \sum_{i=1}^{|S|} m_i$. The size of the composite latent space m is designed to be significantly smaller than the product of the time window length Δt_w and the number of signals $|P|$, such that the latent space still represents a bottleneck. The reconstruction of the full signal set P is performed by the decoder network that takes the composite latent space \mathbf{z}_c as input and produces reconstructed signals $\hat{\mathbf{X}}$.

The composite latent space provides two advantages over conventional approaches: First, the fact that the signals of the individual subsystems are processed in dedicated encoders and decoders ensures that symptoms within these subsystems are isolated. If an anomaly occurs in subsystem a , it cannot affect the reconstruction of the signals corresponding to subsystem b , which differentiates the proposed architecture from those modeling all signals in a single large NN. Second, the approach has the ability to identify cross-subsystem symptoms, namely those symptoms that can only be identified when multiple subsystems are observed simultaneously, which cannot be achieved with individual models for each subsystem.

The second novelty of the proposed NN is the TCN-VAE architecture of the individual encoder and decoder pairs. Figure 5.3 visualizes this architecture. The NNs shown are detailed views of the individual encoders and decoders (for example Enc_a and Enc_b) shown in Figure 5.2.

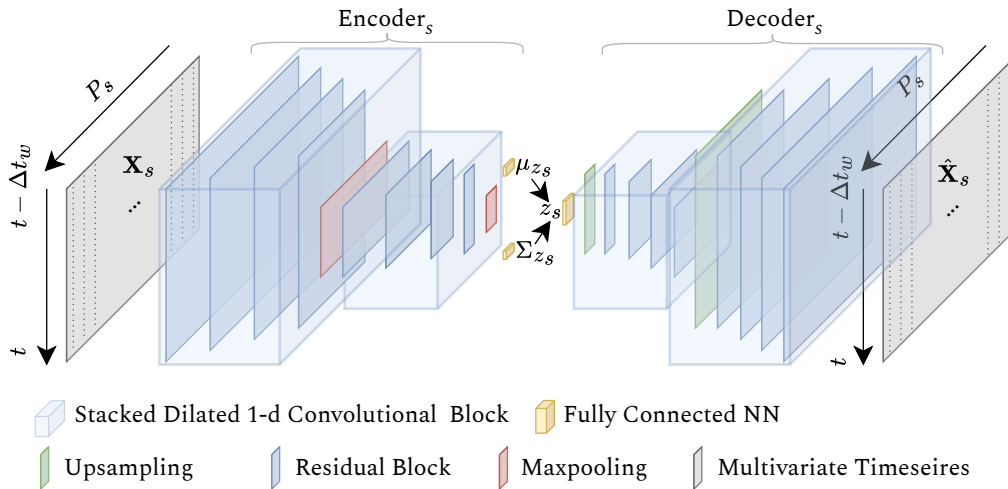


Figure 5.3: Detailed architecture of encoder-decoder pair for one subsystem s within the overall architecture shown in Figure 5.2. The diagram illustrates the internal structure of Encoder_s and Decoder_s for a specific subsystem s . Here the focus lies on the implementation of the NN layers within the individual encoders and decoders, including the stacked dilated convolutions, residual connections, max pooling operations, and dimensionality reduction. The encoder processes a multivariate time series \mathbf{X}_s from subsystem s and computes the parameters for the latent space distribution. The subsystem-specific latent space representation \mathbf{z}_s is sampled from that distribution and passed to the corresponding decoder, which computes the reconstructions $\hat{\mathbf{X}}_s$.

The network architecture is built upon residual blocks (see Section 2.1.2), which consist of two convolutional layers with “same” padding and equal dilation rates to maintain a constant sequence length. Each block incorporates a skip connection to enhance the gradient flow during training. These blocks are organized into a stacked dilated 1-D convolutional structure, similar to TCNs. The difference is that non-causal convolutions are used instead of the “causal” ones proposed by Bai et al. [12]. This approach is appropriate in a reconstruction setup, where the causality constraints described in the original paper do not apply.

Within these stacked blocks, the architecture progressively reduces the channel dimension, which compresses the representation along the signal dimension. The max pooling operations act orthogonally to this, reducing the temporal dimension by halving the time series length at each pooling step. Through this dual-dimensional compression, the input data is condensed into a compact latent representation. Finally, a fully connected layer is used to generate the parameters for the posterior distribution $q_\phi(\mathbf{z}|\mathbf{X})$ from the output of the second max pooling layer. The latent variable \mathbf{z} sampled from this distribution is fed into the decoder $p_\theta(\mathbf{X}|\mathbf{z})$. The decoders mirror the encoder’s architecture, but instead of max pooling they use upsampling layers to rebuild the original time series dimension.

The loss function used for training consists of three components: (i) the MSE for each individual decoder which is averaged over all decoders, (ii) the individual Kullback-Leibler divergences also averaged over the individual posteriors and priors of the subsystems, and (iii) the global MSE for the entire set of signals. The proper probabilistic VAE approach would use the likelihood as described in Section 2.1.5. In this implementation the MSE is used as an approximation that is computationally more efficient and often used in practice. This approximation is justified because, when assuming a Gaussian likelihood with fixed variance $p_{\theta}(\mathbf{X}|\mathbf{z}) = \mathcal{N}(\mathbf{X}|\hat{\mathbf{X}}, \sigma^2\mathbf{I})$, the negative log-likelihood term is proportional to $\|\mathbf{X} - \hat{\mathbf{X}}\|^2$, which is precisely the MSE.¹ Formally, the loss function is defined as follows:

$$\mathcal{L} = \frac{1}{|S|} \sum_{i=1}^{|S|} \left[\text{MSE}(\mathbf{X}_i, \hat{\mathbf{X}}_i) + \beta \text{KL}(q_{\phi_i}(\mathbf{z}_i|\mathbf{X}_i) \| p(\mathbf{z}_i)) \right] + \text{MSE}(\mathbf{X}, \hat{\mathbf{X}}) \quad (5.1)$$

where each \mathbf{X}_i holds the signals associated with subsystem i in the subsystem-signals map $\mathcal{M}(i)$ and $\hat{\mathbf{X}}_i$ is the reconstruction, so that the mean parameter of the output distribution $p_{\theta}(\mathbf{X}_i|\mathbf{z}_i) = \mathcal{N}(\mathbf{X}_i|\hat{\mathbf{X}}_i, \sigma^2\mathbf{I})$, with the variance term σ^2 fixed and not optimized during training. The parameter $\beta \in \mathbb{R}$ is the regularization weight introduced in [78].

5.2.2 Residual Binarizer

The second component of the symptoms generator, which implements the function \mathcal{G} , is the residual binarizer (not to be confused with the residual blocks above), which transforms the continuous-valued outputs, or residuals, of the NN components into binary health states. This subsection formally describes this binarizer.

Subsystem-Level Error Computation

The NN model described above computes a reconstruction of the signals of each subsystem $s \in S$. Based on the observed signals \mathbf{X}_s and the corresponding reconstructions $\hat{\mathbf{X}}_s$, the reconstruction error per subsystem at time t is defined as:

$$r_s(t) = \frac{1}{|\mathcal{M}(s)|} \sum_{p \in \mathcal{M}(s)} (x_p(t) - \hat{x}_p(t))^2 \quad (5.2)$$

where $\mathcal{M}(s)$ is the set of signals mapped to subsystem s . These reconstruction errors $r_s(t) \in \mathbb{R}$ can be interpreted as residual values because they represent the deviation of the observed data from the system behavior learned in the model.

¹The relationship between negative log-likelihood and MSE can be derived as follows: For $p_{\theta}(\mathbf{X}|\mathbf{z}) \sim \mathcal{N}(\hat{\mathbf{X}}, \sigma^2\mathbf{I})$, we have $\log p_{\theta}(\mathbf{X}|\mathbf{z}) \sim \log \exp(-\frac{1}{2\sigma^2}\|\mathbf{X} - \hat{\mathbf{X}}\|^2) \sim -\frac{1}{2\sigma^2}\|\mathbf{X} - \hat{\mathbf{X}}\|^2$. Since σ^2 is treated as a fixed constant during training, minimizing the negative log-likelihood is equivalent to minimizing $\|\mathbf{X} - \hat{\mathbf{X}}\|^2$. The decoder produces a distribution from which the mean $\hat{\mathbf{X}}$ is taken as reconstruction, since it is value that maximizes probability density.

In addition to the subsystem-specific residual values $r_s(t)$, the system-wide reconstruction error is also calculated in accordance with Figure 5.2:

$$r(t) = \frac{1}{|P|} \sum_{p \in P} (x_p(t) - \hat{x}_p(t))^2 \quad (5.3)$$

where $\hat{\mathbf{X}}$ represents the reconstruction of all signals computed by the decoder which gets the composite latent space \mathbf{z}_c as input. This reconstruction error might reveal anomalies that cannot be observed in the subsystem-specific residual values.

These reconstruction errors are used as anomaly scores. High values indicate a strong deviation from the distribution learned by the model. This simple approach was chosen because of its computational efficiency and interpretability. Garg et al. [61] propose some more sophisticated approaches, such as Gaussian-based approaches with static or dynamic parameters and kernel smoothing techniques.

Threshold Selection and Health State Assignment

To transform the continuous residual values $r_s(t)$ into binary health states $h_s(t)$, they are compared with a threshold value $\tau_s \in \mathbb{R}$:

$$h_s(t) = \begin{cases} 1 & \text{if } r_s(t) > \tau_s \\ 0 & \text{otherwise} \end{cases} \quad (5.4)$$

where $h_s(t) = 1$ stands for “not OK” and $h_s(t) = 0$ for “OK”.

If a few anomalies are present in the test set, these thresholds τ_s can be optimized with respect to the metric of choice. From the author’s point of view, the composite F1 score (F_1^c) introduced by Garg et al. [61] is a good choice for anomaly detection models in CPS data. The composite F1 score balances timestamp-wise precision with event-wise recall, where an event is an anomaly. It is defined as:

$$F_1^c = \frac{2 \cdot P^t \cdot R^e}{P^t + R^e} \quad (5.5)$$

where $P^t \in [0, 1]$ is the time-wise precision and $R^e \in [0, 1]$ is the event-wise recall. τ_s can be optimized using this metric:

$$\tau_s = \arg \max_{\tau} F_1^c(\tau; r_s, \mathbf{y}) \quad (5.6)$$

where $F_1^c(\tau; r_s, \mathbf{y})$ computes the composite F1 score achieved when using threshold τ to transform the residuals r_s into binary predictions, which are then compared against the labels \mathbf{y} .

In scenarios where no labels are available, heuristics can be used. A common approach is to define τ based on the distribution of residuals in the test set, which also contains no faults.

$$\tau_s = \mu_{r_s} + \alpha_{\tau} \cdot \sigma_{r_s} \quad (5.7)$$

where μ_{r_s} and σ_{r_s} are the mean and standard deviation of the reconstruction errors for subsystem s , and $\alpha_r \in \mathbb{R}$ is a sensitivity parameter, which is typically set to values between 2 and 3.

5.3 Graph Diagnosis Algorithm

Once the subsystem level health states h_s described above are available the actual diagnosis task begins, which is the identification of the subsystems whose faults are responsible for the observed symptoms. This section describes the graph-based algorithm \mathcal{A} , which was designed to solve this task and is one of the key contributions of this thesis.

5.3.1 Motivation and Background

Several diagnosis algorithms have been proposed that are similar to the one in this thesis [4, 145, 174], but none of them can solve the problem defined in RQ2. For example, the methods proposed by Assaad et al. [4], Diedrich and Niggemann [41], and Stumptner and Wotawa [174] require the causal graphs to be acyclic. This requirement is hard to meet for systems such as the ECLSS due to their highly interconnected subsystems and the complex interactions between them. The solution of Rehak et al. [145] allows these cycles, but relies on predefined root causes as nodes in the causal graph and does not support unseen faults.

The approach presented in the following paragraphs addresses two key challenges: First, diagnostic reasoning is performed within causal graphs that are cyclic, and second, it takes the possibility of several simultaneously occurring faults into account.

5.3.2 Algorithm Design Principles and Evaluation Criteria

The goal of the graph-based diagnosis algorithm \mathcal{A} is formally defined in RQ2 (see Section 4.3). The task is to find those subsystems $S_{\text{causal}}(t)$ whose faults are the most likely cause of the observed symptoms, based on the health state vector $\mathbf{h}(t)$ and the causal graph G . Figure 5.4 visualizes an exemplary graph G , as well as the health state vector $\mathbf{h}(t)$ represented in the color coding. The figure also shows that, based on this limited information, there is no intuitively correct choice for “the causal” subsystems.

To address this challenge, the algorithm uses five basic engineering principles for fault diagnosis, which are used as guidelines for its decision. The causal reachability principle (i) states that a subsystem can only be responsible for the symptoms in another subsystem if it can causally influence it [54]. In the event that there are several explanations for the observed symptoms, the parsimony principle (ii) ensures that those explanations are preferred that contain the smallest number of causal subsystems, which corresponds to the minimal diagnosis concept proposed by Reiter [146]. The proximity principle (iii) describes that symptoms tend to be observable in

the “causal proximity” of the root causes and that the probability of symptoms decreases with distance from the root cause [108]. In addition, the fault manifestation principle (iv) states that faults can in principle also be observed as symptoms in the causal subsystem. However, a lack of observability of the subsystem or false negatives in the symptoms generator might mask this effect. Finally, in accordance with the propagation coherence principle (v), faults typically propagate along connected paths in which the subsystems involved are symptomatic.

For the implementation of these principles in algorithm \mathcal{A} , quantitative criteria are used that describe how well a diagnosis candidate $c \in C \subseteq S$ can explain the symptoms based on the principles. These criteria are defined as follows:

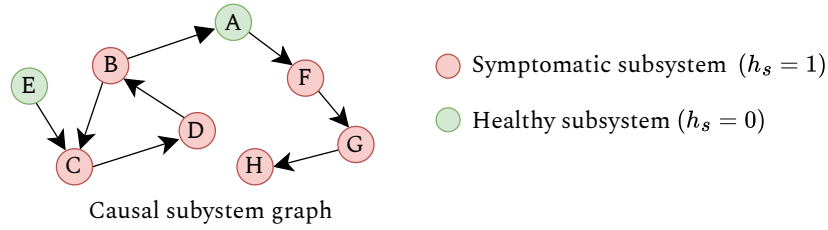


Figure 5.4: Example fault propagation graph G with symptomatic subsystems S_{sym} . This example illustrates the trade-offs between different diagnostic criteria: node E maximizes reachability and parsimony by potentially explaining all symptoms with a single cause. However, node E performs poorly on the anomaly status and chain criteria.

Reachability Criterion This criterion implements the causal reachability principle by calculating the proportion of symptomatic subsystems that can be reached from the candidate in the graph. More formally, for a candidate node c and the set of unexplained symptoms $U \subseteq S_{\text{sym}} \subseteq S$, the reachability value is defined as follows:

$$\sigma_r(c) = \frac{|S_{\text{reach}}(c, U)|}{|U|} \quad (5.8)$$

where $S_{\text{reach}}(c, U) = \{u \in U \mid \exists \text{ path } c \rightarrow u\}$ represents the subset of unexplained symptomatic nodes that can be reached from c . A candidate subsystem that can explain many symptoms therefore receives a higher value $\sigma_r(c)$ than one that can only explain a few.

Distance Criterion This criterion implements the fault proximity principle by examining how densely symptoms are clustered around the candidates. For this purpose, the graph concept of “shells” is used, which describe which nodes are located at a certain distance from the reference node. In the example graph in Figure 5.4, nodes B and C are in the first shell of node E (distance 1), while nodes D and F are in the

5 Solution

second (distance 2) and so on. The density of symptoms in these shells around a candidate can therefore contain information about how well the candidate is positioned to explain these symptoms. Formally, shells are defined as follows:

$$\text{shells}[d] = \{u \in U \mid \text{shortest_path}(c, u) = d\}. \quad (5.9)$$

With this definition, the distance criterion can be formally defined as:

$$\sigma_d(c) = \max_d \frac{|\text{shells}[d]|}{d^2} \quad (5.10)$$

where the normalization with the squared distance d^2 approximates the growth of a typical graph with distance d . A high score $\sigma_d(c)$ therefore indicates that symptoms are densely clustered around the candidate.

Anomaly Status Criterion This criterion applies the fault manifestation principle by checking whether the candidate node itself is symptomatic:

$$\sigma_a(c) = \mathbb{1}_{S_{\text{sym}}}(c) \quad (5.11)$$

where $\mathbb{1}_{S_{\text{sym}}}(c)$ is the indicator function that equals 1 if $c \in S_{\text{sym}}$ and 0 otherwise.

Anomaly Chain Criterion To implement the propagation coherence principle, the length of the paths that start at a candidate and contain only symptomatic nodes is measured:

$$\sigma_c(c) = \frac{\text{max_length}}{|S_{\text{sym}}|} \quad (5.12)$$

where $\text{max_length} = \max\{|\gamma| : \gamma \in \text{paths}(c) \wedge \gamma \subseteq S_{\text{sym}}\}$ is the length of the longest path starting from c that consists entirely of symptomatic nodes. $\text{paths}(c)$ represents the set of all directed paths that start from node c in the graph G .

To obtain a single metric for each candidate subsystem, the scores of the individual criteria are combined into a weighted score:

$$\sigma(c) = w_1\sigma_r(c) + w_2\sigma_d(c) + w_3\sigma_a(c) + w_4\sigma_c(c) \quad (5.13)$$

where $\mathbf{w} = (w_1, w_2, w_3, w_4)$ such that $w_i \in [0, 1]$ and $\sum_i w_i = 1$. These weights represent an input parameter of the algorithm \mathcal{A} proposed in this thesis. The selection of values for the individual weights allows the algorithm to be adapted to different scenarios and CPSs. For example, the reachability criterion could be weighted more heavily via w_1 in linear production systems, or the distance criterion via w_2 in highly connected power distribution networks. In the example visualized in Figure 5.4, different values in the weights vector \mathbf{w} could lead to the inclusion of either node E (high reachability) or one of the nodes in the B-C-D cycle (higher anomaly status and chain metrics) in the set of causal subsystems S_{causal} .

Note that there is no dedicated criterion for the parsimony principle since it must be evaluated not on the basis of individual candidates but on the set of candidates in the diagnosis. This criterion is implicitly implemented in lines 14 to 18 of Algorithm 2 described in the following section.

5.3.3 Algorithm Description

The implementation of \mathcal{A} is shown as pseudocode in Algorithm 2 and can be summarized in four steps:

1. **Initialization** (lines 1-4 in Algorithm 2): In this step, based on the graph $G = (S, E)$ and the health state vector \mathbf{h} (see Section 5.2), the set of symptomatic subsystems S_{sym} is first defined as targets for the root cause analysis. In addition, the set of diagnosis candidates C is formed, which contains all subsystems from which paths start to at least one of the targets. This can be written more formally as $C = \{v \in S \mid \exists s \in S_{\text{sym}} : \text{path}(v \rightarrow s)\}$. The set of unexplained symptoms $U \subseteq S_{\text{sym}}$ is initialized with all symptoms.
2. **Candidate Evaluation** (lines 6-12 in Algorithm 2): For each of the candidate nodes $c \in C$ identified in the initialization, the four criteria defined above are evaluated. Once computed, the individual scores are summarized into the total score of the candidate $\sigma(c)$ using the weights \mathbf{w} passed to the algorithm as parameters.
3. **Root Cause Selection** (lines 13-14 in Algorithm 2): Based on the scored candidates, an initial suggestion for a diagnosis is made, which contains all candidates that have a score that exceeds the product $\theta\sigma_{\text{max}}$, where σ_{max} is the largest total score among all candidates. $\theta \in [0, 1]$ is a sensitivity parameter that will be discussed further in Section 5.3.4.
4. **Iterative Refinement** (lines 15-18 in Algorithm 2): In this step, the algorithm checks whether the initial suggestion for the diagnosis can explain all observed symptoms by removing all symptoms from U that can be reached from one of the candidates in the initial diagnosis. If unexplained symptoms remain, the algorithm starts the next iteration and re-evaluates the scores of the remaining candidates based solely on the unexplained symptoms.

The algorithm employs BFS for computing reachability and shortest paths. The algorithm terminates when either all symptoms are explained ($U = \emptyset$) or no more candidates are available ($C = \emptyset$).

5.3.4 Computational Complexity and Trade-offs

The algorithm \mathcal{A} described above balances completeness and computational efficiency. Once an initial set of root causes has been identified, it is extended by the

5 Solution

next best candidates if not all symptoms are explained already. Thus, not all potential combinations of root causal subsystems are considered. This design decision was made in order to avoid the combinatorial explosion problem often found in diagnosis methods. Nevertheless, the algorithm offers the option of adding nodes to the diagnosis via repeated runs with modified values of the parameter θ if the diagnosis calculated in the first run does not turn out to be the correct one after an analysis by engineers. θ determines the sensitivity of the algorithm. A lower value increases the number of subsystems obtained in the diagnosis. With a value of $\theta = 0$, all initial candidate elements would be in the diagnosis $S_{\text{causal}} = C$. Conversely, the largest value $\theta = 1$ minimizes the number of subsystems contained in the diagnosis, but still ensures that no symptom remains unexplained. If the powerset of all candidates were evaluated instead of the individual candidates c , the algorithm would have a worst-case time complexity of $O(2^{|S|}(|S| + |E|))$. In contrast, the proposed algorithm achieves $O(|S|^2(|S| + |E|))$, where the quadratic term is due to the fact that in the worst case each node must be evaluated as a candidate in each iteration. Due to this polynomial complexity, the algorithm remains applicable even for large systems. A more detailed evaluation of the runtime is described in Chapter 7.

Algorithm 2 Graph Diagnosis Algorithm**Require:** Graph $G = (S, E)$, health state vector \mathbf{h} , weights \mathbf{w} , threshold θ **Ensure:** Set of root causes S_{causal}

```

1:  $S_{\text{sym}} \leftarrow \{s \in S \mid h_s = 1\}$  ▷ Anomalous subsystems
2:  $C \leftarrow \{v \in S \mid \exists s \in S_{\text{sym}} : \text{path}(v \rightarrow s)\}$  ▷ Candidates
3:  $S_{\text{causal}} \leftarrow \emptyset$  ▷ Root causes
4:  $U \leftarrow S_{\text{sym}}$  ▷ Unexplained anomalies
5: while  $U \neq \emptyset \wedge C \neq \emptyset$  do
6:   for  $c \in C$  do
7:      $\sigma_r(c) \leftarrow \text{COMPUTEREACHABILITYSCORE}(c, U)$ 
8:      $\sigma_d(c) \leftarrow \text{COMPUTEDISTANCESCORE}(c, U)$ 
9:      $\sigma_a(c) \leftarrow \text{COMPUTEANOMALYSCORE}(c, S_{\text{sym}})$ 
10:     $\sigma_c(c) \leftarrow \text{COMPUTECHAINSCORE}(c, S_{\text{sym}})$ 
11:     $\sigma(c) \leftarrow w_1\sigma_r(c) + w_2\sigma_d(c) + w_3\sigma_a(c) + w_4\sigma_c(c)$ 
12:  end for
13:   $\sigma_{\max} \leftarrow \max_{c \in C} \sigma(c)$ 
14:   $B \leftarrow \{c \in C \mid \sigma(c) \geq \theta \cdot \sigma_{\max}\}$ 
15:  for  $b \in B$  do
16:     $S_{\text{causal}} \leftarrow S_{\text{causal}} \cup \{b\}$ 
17:     $U \leftarrow U \setminus S_{\text{reach}}(b, U)$ 
18:     $C \leftarrow C \setminus \{b\}$ 
19:  end for
20: end while
21: return  $S_{\text{causal}}$ 
22: // Helper functions for score computation
23: function  $\text{COMPUTEREACHABILITYSCORE}(c, U)$ 
24:    $S_{\text{reach}}(c, U) \leftarrow \{u \in U \mid \exists \text{path } c \rightarrow u\}$ 
25:   return  $|S_{\text{reach}}(c, U)|/|U|$ 
26: end function
27: function  $\text{COMPUTEDISTANCESCORE}(c, U)$ 
28:   // Group anomalies by their distance from  $c$ 
29:    $\text{shells}[d] \leftarrow \{u \in U \mid \text{shortest\_path}(c, u) = d\}$ 
30:   // Max. density, normalizing by approx. surface  $d^2$ 
31:   return  $\max_d |\text{shells}[d]|/d^2$ 
32: end function
33: function  $\text{COMPUTEANOMALYSCORE}(c, S_{\text{sym}})$ 
34:   return  $\mathbb{1}_{S_{\text{sym}}}(c)$  // Indicator function
35: end function
36: function  $\text{COMPUTECHAINSCORE}(c, S_{\text{sym}})$ 
37:   // Let  $\text{paths}(c)$  be all directed paths starting at  $c$ 
38:   // Find longest path through symptomatic nodes only
39:    $\text{max\_length} \leftarrow \max\{|p| : p \in \text{paths}(c) \wedge p \subseteq S_{\text{sym}}\}$ 
40:   return  $\text{max\_length}/|S_{\text{sym}}|$ 
41: end function

```

5.4 Machine Learning Operations Implementation

As motivated in the introduction, the implementation, deployment and maintenance of ML-based methods such as the symptoms generator (see Section 5.2) in production environments represent a significant challenge (RQ3). In contrast to traditional software, data-driven approaches add problems such as degrading model performance or non-reproducible model training. Furthermore, there are large differences in compute resource requirements between training and inference time. Finally, despite the rapid development of MLOps tools, it is still not trivial to combine the tools in a uniform workflow, since custom workarounds and adapters have to be developed. For CPS applications such as the diagnosis method of this thesis, there are typically large data volumes and requirements for security and streamed data. This section describes an MLOps reference implementation that addresses these challenges.

5.4.1 Implementation Strategy

The MLOps implementation follows a series of requirements that were developed based on the challenges described above:

1. **Scalable Data Processing:** Because the datasets of large and complex CPS also tend to be large and complex, the implementation must support efficient preprocessing via distributed computing so that out-of-memory datasets can be transformed in reasonable time for use in ML model training.
2. **Automated Model Optimization:** Just like preprocessing, training and hyperparameter tuning must be supported by parallelization frameworks due to the data and model size. This requirement adds additional complexity as the use of multiple GPUs distributed across different computers is a more recent and not as well-explored problem as distributed computing on CPUs.
3. **Continuous Model Maintenance:** Due to the likely occurrence of model and concept drift [33, 164], the implementation should automate the entire training pipeline, including preprocessing, hyperparameter tuning, training and deployment.
4. **Production-grade Inference:** For use on continuously streamed signals, the implementation should support low-latency and high throughput and also automatically scale compute resources up and down as required.

To implement a solution that meets these requirements, the pipeline was developed on the basis of Kubernetes and the Kubeflow ecosystem. This choice is based on recent reviews and comparisons of MLOps tools and platforms [15, 97]. Because neither a distributed computing framework nor a model registry were included in the feature set of the Kubeflow ecosystem at the time of development, Dask [149] and MLflow were also added to the infrastructure used [205].

5.4.2 MLOps Components and Workflow Integration

The reference implementation is shown at a high level in Figure 5.5, in which the individual steps are aligned with the phases of the CRISP-DM [115, 161] cycle (see Section 2.3 for more details on the connection of MLOps and CRISP-DM). The following paragraphs describe the most important steps of this pipeline with a focus on the technical integration of the different MLOps tools.

Data Processing and Preparation To integrate support for distributed computations, Dask [149], which is not part of the Kubeflow ecosystem, was integrated into the Kubernetes-based infrastructure used. Dask is also referred to as “distributed pandas” because it focuses on scaling data transformation workloads and provides an interface similar to pandas [179]. Because it supports Kubernetes as a backend, the integration efforts were limited to the installation of the Dask operator ².

Hyperparameter Optimization Most NN models are configured via hyperparameters, which are optimized in dedicated tuning procedures. In the case of the symptoms generator presented in this thesis, these include the dimension of the latent space, the learning rate or the β parameter. In this step, the implemented pipeline uses a tool called Katib [62], which is part of the Kubeflow ecosystem and was developed precisely for this purpose.

Distributed Model Training Training deep NNs on large datasets requires considerable computational resources, ideally including accelerators such as GPUs. Because the capacity of individual computers is limited, the GPUs are installed on several machines and provided via cluster technologies, such as Kubernetes. The challenge lies in the orchestration of the training process across these machines in the cluster. The implemented pipeline uses the built-in Kubeflow training operator [8], which provides native support for distributed training using PyTorch’s DDP algorithm [104]. Within the training procedure, the ring-based AllReduce algorithm [138] is used to efficiently synchronize the gradients calculated in different parallel processes based on different training data batches. The training process and the model versions are managed with MLflow [205] due to the lack of a native Kubeflow solution.

Model Serving and Inference For the integration of the model with the consuming microservices³, the model can be deployed as an API and has to be performant enough to keep up with the continuously streamed data. The deployment solution within the Kubeflow ecosystem that was used in the implemented pipeline is KServe [7].

²In Kubernetes, an “Operator” is a method of packaging, deploying, and managing a Kubernetes application [79]

³Microservices are units of a software application that consist of several loosely coupled services which can be developed and deployed independently of each other [103].

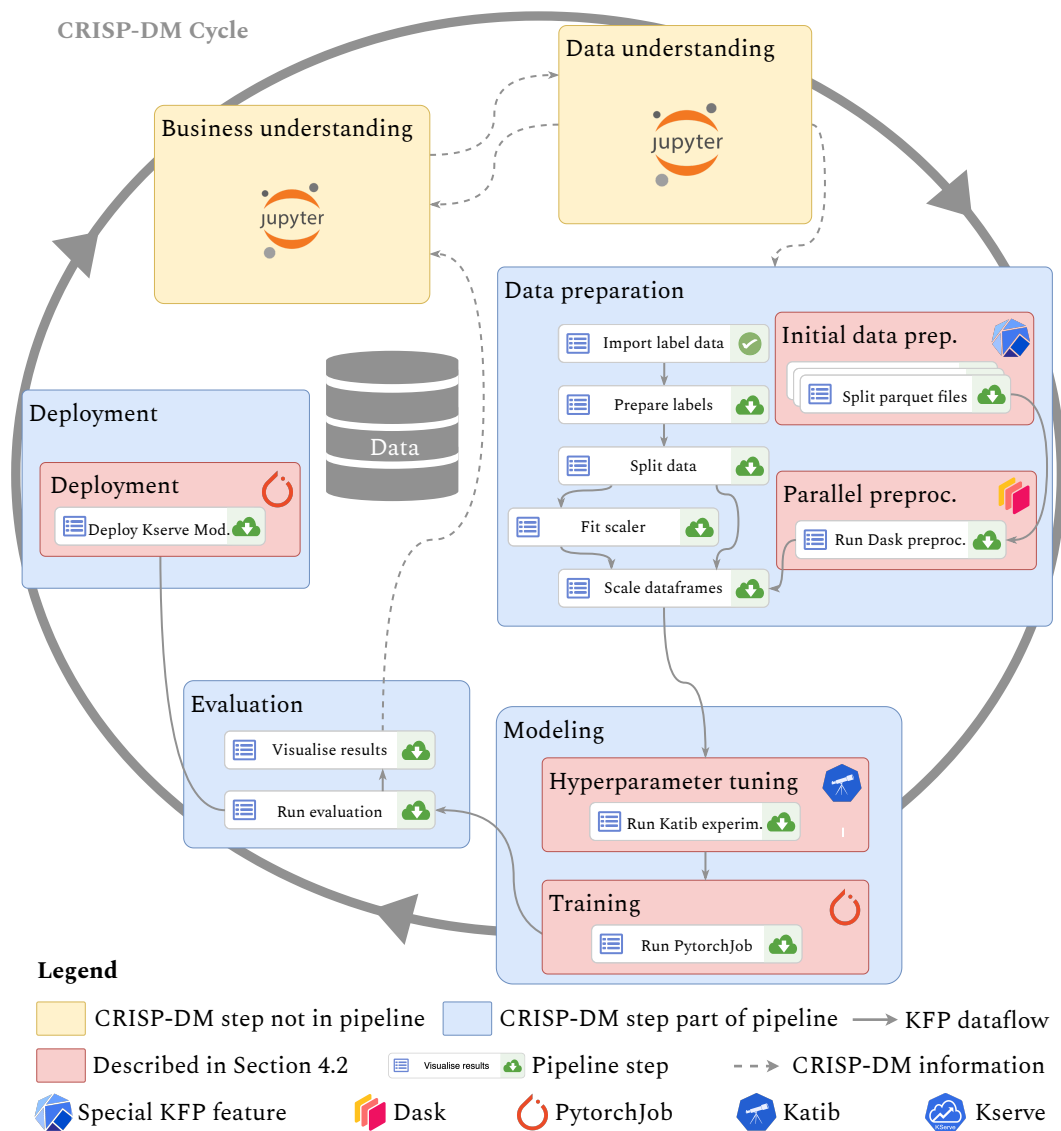


Figure 5.5: Overview of the MLOps pipeline implemented in order to answer research question RQ3. This figure shows the pipeline steps aligned with the CRISP-DM cycle phases. This reference implementation shows how cloud-native technologies such as Dask, Katib, PyTorch Training Operators and KServe can be combined in a single KFP pipeline.

APIs provided with KServe are serverless⁴, and are easy to set up with models developed in Kubeflow.

⁴Serverless computing is a software execution model where server management including scaling or monitoring is abstracted away from the developer [159]

Workflow Orchestration One of the core challenges is the integration of the above-mentioned tools into an automated workflow. The pipeline visualized in Figure 5.5 was implemented using KFP to accomplish this integration. The pipeline automates the integration of the tools and thus allows the reproducible execution of all contained steps, which is particularly relevant with regard to regular retraining due to changed system configurations or data drift for CPS such as the ECLSS.

5.4.3 Implementation Challenges and Solutions

The implementation of the ML pipeline described above posed several challenges. The following paragraphs describe the most important ones and their corresponding solutions.

Data Management As described above, Dask is used as a parallel computing framework, which distributes workloads to processes (or Kubernetes Pods) that run on different machines in the cluster. This also means that the data to be processed must be available on all nodes in the cluster. In addition, the data must be partitioned in such a way that it can be read in parallel by different processes according to Dask's computational graph. To fulfill these requirements, an initial pre-processing step is typically implemented, which splits large datasets into smaller chunks and stores them in a distributed storage system accessible by all worker nodes. In this reference implementation, an open-source S3-compatible object store [119] was used. This approach enables efficient parallel processing of datasets that would be too large to handle on a single machine (at least on reasonably sized machines).

Integration Between Components The biggest challenge was the integration of the tools within the Kubeflow ecosystem into a single holistic KFP pipeline, since most of them were developed independently of each other. KFP makes it very easy to exchange data between individual process steps in an ML pipeline and abstracts the management of the corresponding artifacts away from the user. However, as soon as other tools such as Katib, PyTorchJobs or KServe need to access these artifacts, custom workarounds are required that have been implemented using Python in the reference pipeline.

Monitoring and Retraining As Berberi et al. [15] show in their analysis, there is a lack of tools for monitoring model quality in MLOps platforms, especially in the case of the Kubeflow ecosystem. Therefore, one part of the implementation is a step in the KFP pipeline which evaluates the model. The reference pipeline also includes an "if condition" implemented in KFP that decides whether the downstream pipeline step that deploys the model to KServe is executed or not. With this check, the whole workflow can be executed automatically on new data either based on a schedule or triggered by detected model or data drift. Because KFP logs and saves the runs, it can be traced at any time which model version was created in which run.

5 *Solution*

The application of this reference implementation to the ECLSS use case is demonstrated in Section 6.5.

Part III

Evaluation

6 Experimental Results

Chapter Outline This chapter describes the experiments evaluating the diagnostic approach presented in Chapter 5. Section 6.1 first formulates the set of hypotheses used to evaluate the individual components of the overall solution. According to these hypotheses, the remaining sections describe the experiments and the corresponding results for the evaluation of the symptoms generator, the graph algorithm, the combination of these two components, and the MLOps implementation.

6.1 Evaluation Criteria & Hypotheses

The experiments described in this chapter are organized according to the four core contributions of this thesis as presented in Section 1.3. For each contribution, this section formulates a dedicated hypothesis that was used to design the experiments.

Hypothesis 1 (H1): *The composite latent space architecture outperforms conventional architectures in localizing anomalies at the subsystem level of CPSs.* This hypothesis evaluates the first contribution, the subsystem-level anomaly detection architecture. It examines whether the NN architecture proposed as part of the symptoms generator delivers better results in the detection and localization of symptoms in subsystems than conventional approaches. Note that this hypothesis deals exclusively with the symptoms generator and therefore not yet with the identification of the root causes.

Hypothesis 2 (H2): *The graph diagnosis algorithm identifies causal subsystems S_{causal} in both acyclic and cyclic graphs, assuming health state vector \mathbf{h} and graph G are accurate.* This hypothesis addresses the second contribution, the graph-based diagnostic reasoning algorithm. The focus here is on evaluating the graph algorithm in isolation. Since \mathbf{h} and G are assumed to be known and accurate, both potential misclassifications of the symptoms generator and faulty graphs G are not considered.

Hypothesis 3 (H3): *When used in combination, the symptoms generator \mathcal{G} and the graph diagnosis algorithm \mathcal{A} identify root causes of faults in complex systems.* This hypothesis evaluates the third contribution, the integrated diagnostic framework visualized in Figure 5.1. It examines the extent to which the holistic approach can be used for fault diagnosis in CPS. It therefore also evaluates the practical feasibility of formalizing the necessary prior knowledge in realistic scenarios.

Hypothesis 4 (H4): *The Kubeflow-based MLOps implementation enables the automation of the entire ML pipeline for deploying and retraining the diagnostic models described above.* This hypothesis evaluates the fourth contribution, the MLOps implementation. It examines whether the MLOps tools presented in the solution chapter can be integrated in a holistic end-to-end ML pipeline covering all the steps described in the CRISP-DM cycle.

In the following sections, the experiments designed to test each of these hypotheses are presented. Section 6.2 evaluates the subsystem-level anomaly detection architecture (H1), while Section 6.3 assesses the graph diagnosis algorithm (H2). The integrated diagnostic framework that combines these two contributions is evaluated in Section 6.4 (H3). Finally, Section 6.5 demonstrates the MLOps implementation (H4). The code to reproduce the experiments or to study implementation details is publicly available on Github.¹²³

6.2 Evaluation of the Symptoms Generator

This section presents the experiments and results corresponding to the evaluation of the symptoms generator \mathcal{G} introduced in Section 5.2.

6.2.1 Experimental Setup

That TCNs, although proposed some time ago [12], still show performance comparable to modern architectures such as RIMs or Transformers when applied to CPS data has been demonstrated in our recent paper [192], where different NN architectures were benchmarked on various real-world CPS datasets. The anomaly detection capabilities of TCN-VAEs and TCN-AEs were also demonstrated using various real-world datasets [61, 194].

Therefore, the experiments in this section focus on the question if the proposed composite latent space architecture improves subsystem-level anomaly detection compared to other state-of-the-art approaches. However, despite a large number of datasets for evaluating anomaly detection methods on multivariate time series in general, there are no real-world CPS datasets that contain labels for anomalies at the subsystem level. For this reason, a dataset simulated specifically for this purpose was created where the precise location of the anomalies is known.

Dataset The simulation models a system that consists of two subsystems $S = \{s_a, s_b\}$ and records a total of 6 signals $P = \{a_1, a_2, a_3, b_1, b_2, b_3\}$, which are plotted

¹The content of Section 6.2 was published in [172] and the code can be found at <https://github.com/hsteude/diag-driven-ad-4-cps>.

²The content of Section 6.3 is published as a preprint [167] and the code is available at <https://github.com/hsteude/ad-diag-end2end-experiments>.

³The content of Section 6.5 was published in [168, 169, 170] and the code is available at <https://github.com/hsteude/code-ml4cps-paper>.

in Figure 6.1. The signals are assigned to the subsystems according to the following subsystem-signals map:

$$\mathcal{M} = \begin{cases} s_a \mapsto \{a_1, a_2, a_3\} \\ s_b \mapsto \{b_1, b_2, b_3\} \end{cases} \quad (6.1)$$

which aligns with those of the exemplary systems visualized in Figure 4.1 and 6.2. The signals a and b visualized at the top of Figure 6.1 represent the causal steering signals for their respective subsystems. Both of these steering signals are rectangular signals. The length of the high and low states of signal a is randomly sampled from a uniform distribution, ranging between 500 and 1000 timesteps. Signal b is a delayed version of signal a .

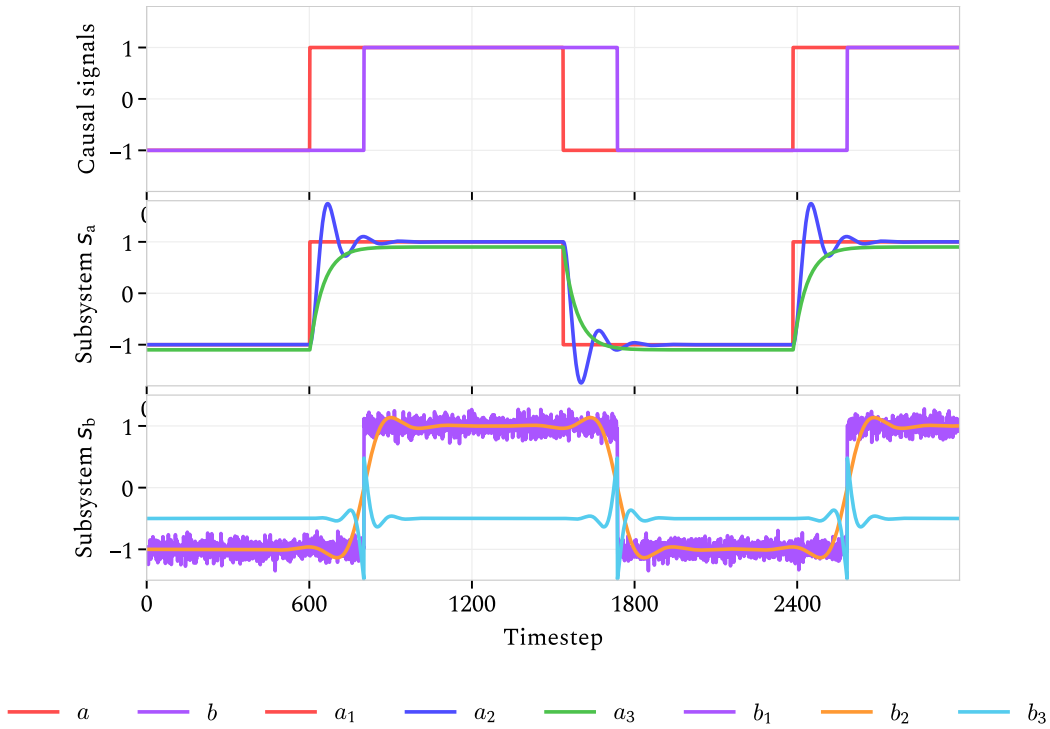


Figure 6.1: Healthy data sample (3000 timesteps). Causal signals a and b (top) with derived signals a_1 to a_3 and b_1 to b_3 (middle and bottom), used for model training and validation.

Benchmark Models Three modeling approaches that are typically used in practice serve as baseline models.

1. A *vanilla TCN-VAE*, consisting of an extensive encoder and decoder that processes all signals from the CPS, reconstructing the input data (illustrated in

6 Experimental Results

blue in Figure 6.2). For subsystem-specific anomaly detection, the average reconstruction error of the signals associated with each subsystem is calculated.

2. A *univariate TCN-VAE* whose encoder and decoder follow the architecture depicted in Figure 5.3, but models each signal $p \in P$ independently (shown in green in Figure 6.2). The subsystem-specific reconstruction error is similarly derived by aggregating the errors of the corresponding signals.
3. A *Gaussian Mixture Model (GMM)* trained on individual time points, capturing only the data distributions at each timestamp. For this model, a separate instance is trained for each subsystem to derive subsystem-specific anomaly scores (represented in yellow in Figure 6.2).

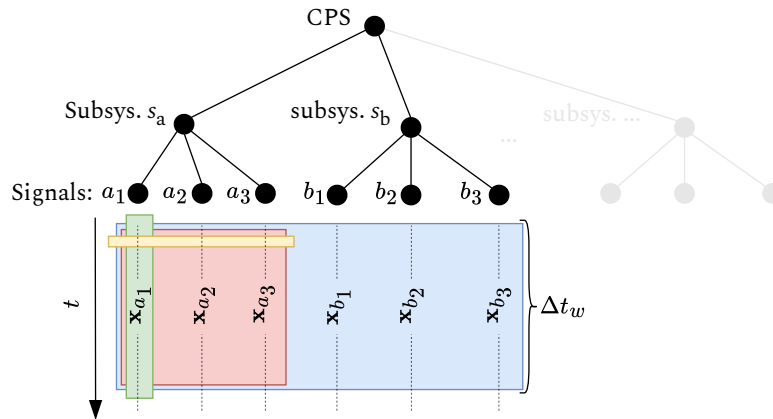


Figure 6.2: Different approaches to modeling subsystem-level anomalies in multi-variate time series data: blue (multivariate approach), green (univariate approach), yellow (point-wise approach), red (the proposed composite latent space approach).

The two TCN-VAE models were used as benchmarks for two reasons. On the one hand, they still belong to the state-of-the-art time series models, as discussed in Section 2.1.2 and Section 2.1.5. On the other hand, using them puts a stronger focus on the difference created by the composite latent space architecture, because the rest of the models are more similar than if other layer types such as LSTMs were used in the benchmarks. The Gaussian Mixture Model (GMM) is the only non-neural model, used as a benchmark. Since it only models the distributions of the time points in six dimensions, there is no need for a sophisticated model such as a deep NN.

Experimental Configuration To ensure fair model performance comparison, all NN-based models have approximately 500k parameters, where the total latent variables across models is kept uniform at 12, with the exception of the GMM which, being a shallow model, has a completely different parameter structure. Each model's

hyperparameters were tuned independently. The corresponding training was executed using the optimal hyperparameters and early stopping based on the validation loss.

Data from nominal operations was divided into a training and a validation split. To assess the anomaly detection capabilities, four distinct fault scenarios were introduced into the test dataset:

- Fault 1: The signal a_1 remains constant at a value of -1, simulating a “stuck-at” fault condition.
- Fault 2: An offset is introduced to signal b_3 , elevating its value by +1, simulating a calibration or drift fault.
- Fault 3: All signals within subsystem b are temporally shifted by the same amount, to mimic a delay fault.
- Fault 4: The signals for both subsystems are modulated to operate at twice their normal frequency, to represent a speed or performance anomaly.

The test set includes 100 samples from each of the four fault scenarios (1-4) and 400 healthy state samples such that the classes are balanced. Binary labels are used, where 0 indicates no fault and 1 indicates the presence of faults within subsystems or the entire signal set, as shown in Table 6.1. The thresholds for anomaly detection in each model and subsystem were computed to optimize their respective F1 scores as described in Section 5.2.2.

Table 6.1: Label allocation in the test set.

Fault type	Subsys. s_a	Subsys. s_b	All signals
Healthy	0	0	0
Fault 1	1	0	1
Fault 2	0	1	1
Fault 3	0	0	1
Fault 4	1	1	1

6.2.2 Results and Analysis

The results of the experiment are summarized in Table 6.2. Regarding subsystem-level anomaly detection, the proposed model consistently scores higher in the F1 scores than the benchmark models. For system-wide symptom identification, the proposed model’s performance is comparable to that of the vanilla TCN-VAE. This was expected, since the development focus was on improving symptom isolation at the subsystem level rather than enhancing detection capabilities across the entire system.

For a more detailed view, Figure 6.3 visualizes the distributions of the anomaly scores r_s in the different test cases (fault 1-4 and healthy) for each subsystem s_a, s_b

Table 6.2: Anomaly Detection Performance Across Subsystems.

Model	Subsys.	F1	Precision	Recall
GMM	s_a	0.436	0.982	0.28
Univar. TCN-VAE	s_a	0.662	1	0.495
Vanilla TCN-VAE	s_a	0.809	0.749	0.88
Proposed model	s_a	0.945	1	0.895
GMM	s_b	0.664	0.99	0.5
Univar. TCN-VAE	s_b	1	1	1
Vanilla TCN-VAE	s_b	0.786	0.663	0.965
Proposed model	s_b	1	1	1
GMM	all	0.811	1	0.682
Univar. TCN-VAE	all	0.677	0.88	0.55
Vanilla TCN-VAE	all	0.946	1	0.898
Proposed model	all	0.946	0.997	0.9

and for the whole system (all). The red colored box plots indicate that the subsystem is symptomatic, while the blue colored ones show no symptoms. Therefore, a good anomaly detection model would consistently produce distributions with higher values for the symptomatic (red) cases than for the healthy (blue) ones.

The proposed model (bottom row) demonstrates the most consistent separation between symptomatic and normal distributions across all scenarios. For example, in Fault 1 which affects only subsystem s_a , the proposed model correctly shows higher scores exclusively for subsystem s_a (red) while maintaining low scores for the unaffected subsystem s_b (blue). Similarly, for Fault 2 affecting only subsystem b , the model shows higher scores exclusively in that subsystem. However, the vanilla TCN-VAE model (third row) does not isolate the symptoms properly. When a fault occurs in subsystem s_a (Fault 1), it produces higher scores in subsystem s_b as well, which records signals equal to those from normal operations. The GMM fails to differentiate frequency-based anomalies (Fault 4) from normal operation, as its point-wise modeling approach cannot capture temporal patterns. The univariate TCN-VAE struggles particularly with those faults involving relationships between signals (Fault 3).

The experimental results provide empirical evidence supporting hypothesis H1. Furthermore, the proposed model achieves system-wide anomaly detection performance comparable to that of the vanilla TCN-VAE.

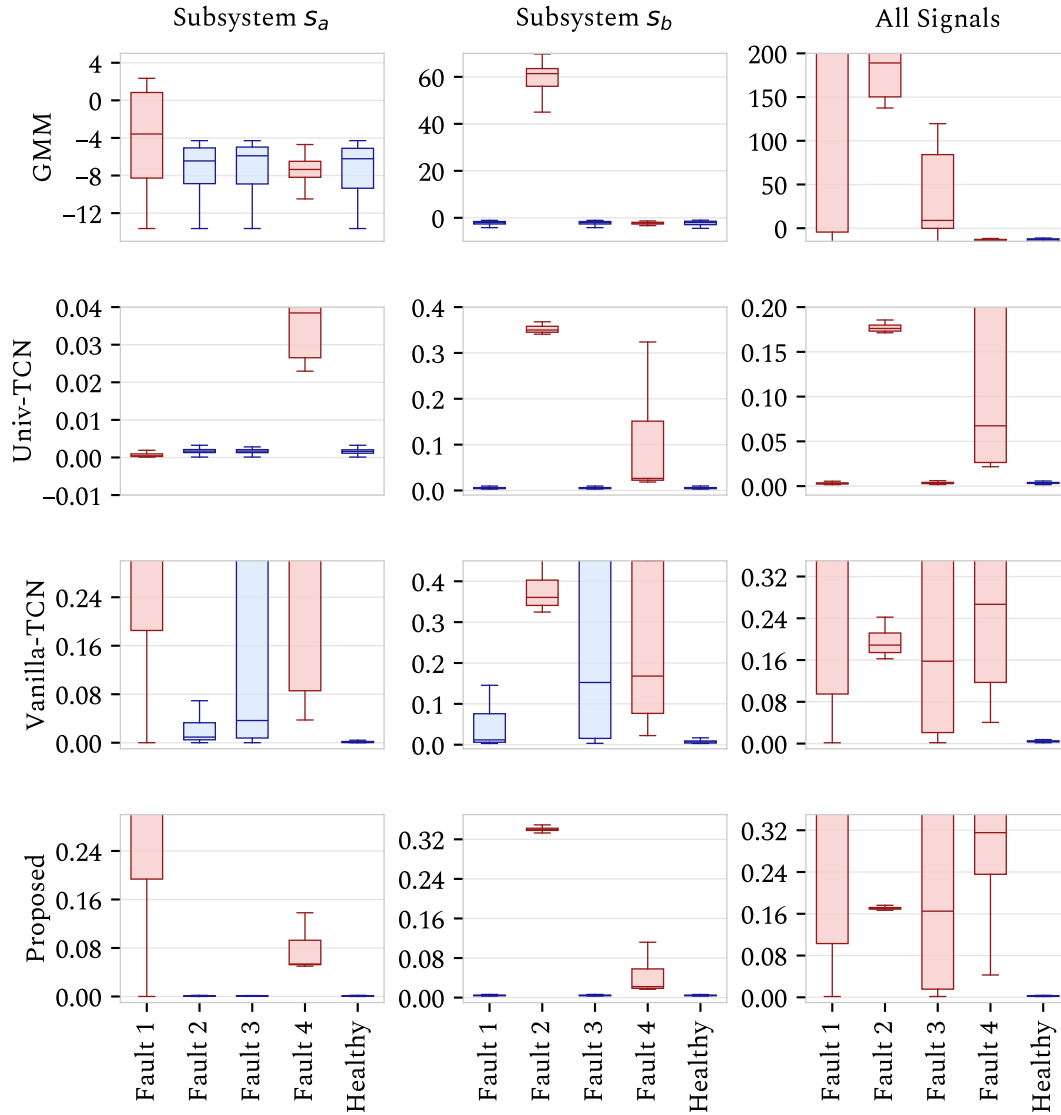


Figure 6.3: Distribution of anomaly scores across different fault scenarios for each model and subsystem. Red boxes indicate the presence of a symptom in the corresponding subsystem according to the ground truth labels, while blue boxes indicate normal operation. The different fault types and the healthy state are differentiated on the x-axis. The y-axis represents the anomaly score. The y-axis scales vary between subplots to focus on the relevant value ranges where discrimination occurs. A perfect model would clearly separate red (with high values) and blue (with small values) distributions.

6.3 Evaluation of the Graph Diagnosis Algorithm

This section describes the evaluation of the graph-based diagnostic reasoning algorithm presented in Section 5.3. It begins with a real-world example, the highly

cited Tennessee Eastman Process (TEP) [43], which is used in many studies on process monitoring [30]. In a second experiment, described in subsection 6.3.2, the capabilities and limitations of the algorithm are discussed using more illustrative artificial scenarios of varying complexity.

6.3.1 Tennessee Eastman Process

This experiment aims to evaluate hypothesis H2, which assumes that the diagnostic reasoning algorithm can identify the subsystem responsible for the observable symptoms if both the health state vector \mathbf{h} and the graph G are known and accurate. The TEP is particularly well suited here because it has been discussed and analyzed in great detail in the literature. Therefore, it is possible not only to derive G , but also \mathbf{h} for some of the faults.

Experimental setup The TEP dataset contains 53 signals. In the first step of this experiment, these signals were assigned to subsystems as formalized in the subsystem-signals map \mathcal{M} . For this assignment, the classification of signals created by Yin et al. [202] into what the authors refer to as blocks was used. Only their “Miscellaneous” category was further divided into the subsystems “Cooling_System”, “Compressor_System”, and “Purge_System”.

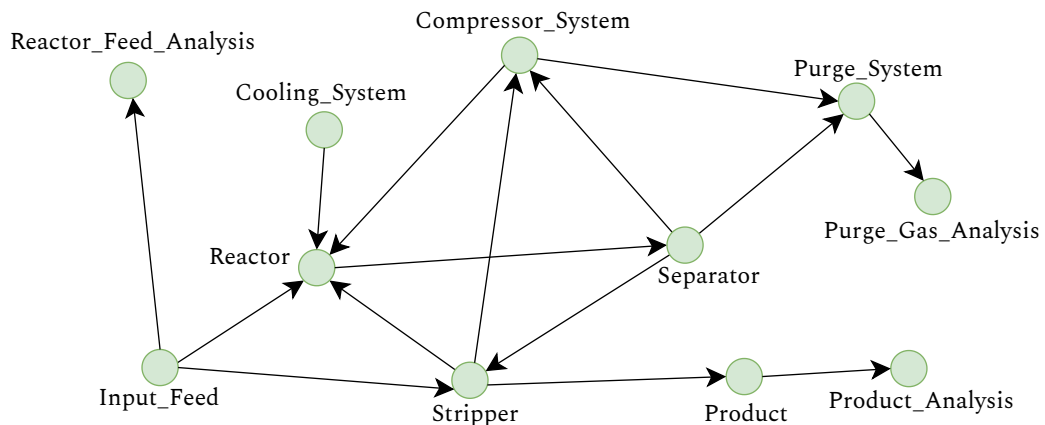


Figure 6.4: Causal graphical representation of the Tennessee Eastman process. The directed edges represent potential fault propagation paths between subsystems derived from the process flow diagram of the TEP.

In the next step, the graph G was created using the process diagram shown, for example, in the original paper by Downs and Vogel [43]. The result is illustrated in Figure 6.4. To derive the health state vector \mathbf{h} , the four faults discussed in detail by Chiang et al. [30] were used (Faults #1, #4, #5 and #11). Their signal-by-signal analysis allows the identification of the set of signals that show symptoms during these

faults. Together with the subsystem-signals map \mathcal{M} , this can be used to determine the health state vector \mathbf{h} for each fault.

In the absence of more detailed information, the graph algorithm for this experiment was initialized with evenly distributed weights ($w_r = w_d = w_a = w_c = 0.25$) for all four evaluation criteria and a moderately strict threshold of $\theta = 0.9$ (see Algorithm 2).

Table 6.3: Comparison of true causal subsystems, observed symptomatic subsystems, and predicted root causes for the four fault scenarios of the Tennessee Eastman Process that were described in detail by Chiang et al. [30].

Fault #	True S_{causal}	S_{sym}	Predicted S_{causal}
F1	Input_Feed	Reactor_Feed_Analysis, Input_Feed, Reactor, Compressor_System, Product_Analysis	Input_Feed, Compressor_System
F4	Cooling_System	Cooling_System, Reactor	Cooling_System
F5	Cooling_System	Cooling_System, Separator	Cooling_System
F11	Cooling_System	Cooling_System, Reactor	Cooling_System

Results and Analysis As shown in Table 6.3, the true causal subsystem is contained in the set of predicted root causes in all four cases. In fault #1, in addition to the true faulty subsystem, “Input_Feed,” the set of predicted root causes also contains the subsystem “Compressor_System”. However, the search space was significantly reduced from five symptomatic subsystems to only two in the set of predicted root causes. For all other faults that were analyzed by Chiang et al. [30] (#4, #5 and #11), the “Cooling_System” caused the fault and was predicted as the only root cause by the algorithm.

This experiment provides evidence that the algorithm can deliver added value in simple cases in real systems. However, in terms of the complexity of the diagnostic reasoning problem, these fault cases are rather trivial. Therefore, in the experiment discussed in the next section, dedicated scenarios are set up to better discuss the capabilities and limitations of the algorithm.

6.3.2 Controlled Graph Scenarios

To analyze the algorithm in greater depth, a series of scenarios were developed that vary in the complexity of the graph and the distribution of the symptoms. Since there is no prior knowledge on these scenarios with regard to fault propagation,

6 Experimental Results

equal weights of $w_i = 0.25$ for all $w_i \in \mathbf{w}$ were chosen (see Section 5.3.2 for details on the weights).

- **Scenario 1:** An acyclic graph with a single symptom cluster as a baseline case for causal analysis.
- **Scenario 2:** An acyclic graph containing two independent symptom clusters for evaluating multi-fault capabilities.
- **Scenario 3:** A cyclic graph with a single symptom cluster as a baseline for assessing the impact of feedback loops.
- **Scenario 4:** A cyclic graph with multiple symptom clusters for investigating the multi-fault performance in the context of cyclic dependencies.

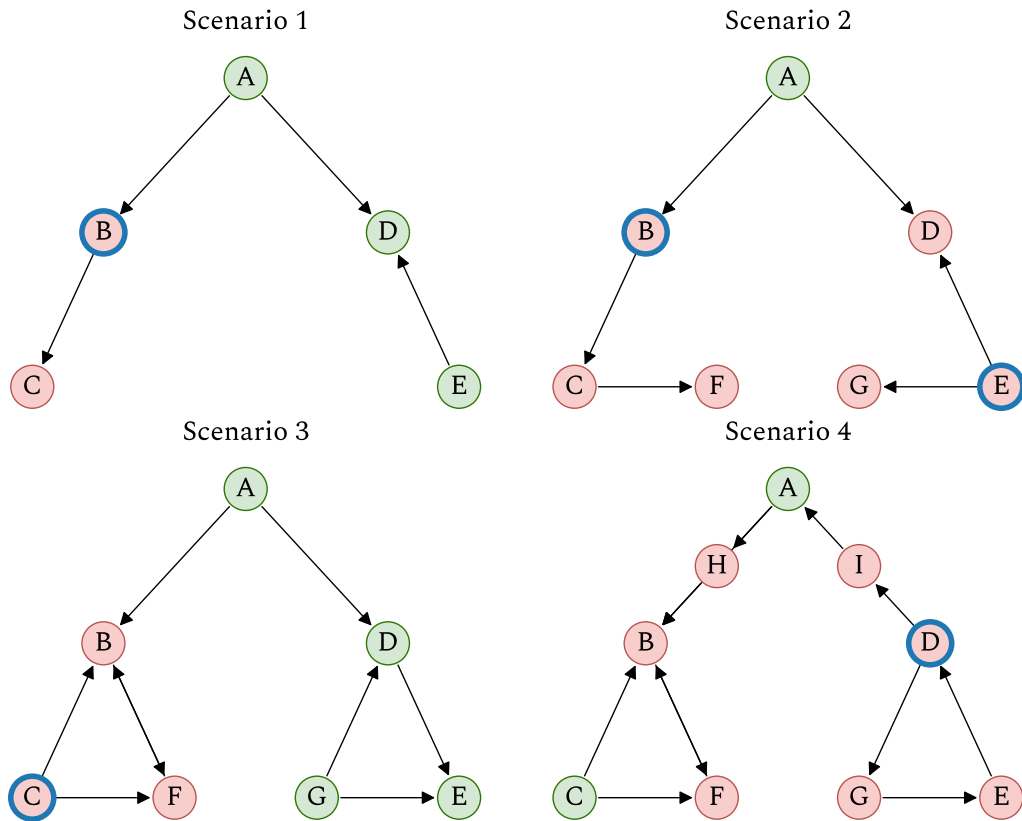


Figure 6.5: Evaluation of the graph diagnosis algorithm in four scenarios with $\theta = 1.0$. Green nodes represent subsystems in normal state ($h_s = 0$), red nodes represent symptomatic subsystems ($h_s = 1$ and elements of S_{sym}) that are given as input to the algorithm, and blue circles represent predicted root causes (elements of S_{causal}).

Figure 6.5 illustrates the four scenarios and the corresponding predictions of the graph diagnosis algorithm. The results show that the algorithm predicts good candidates across all the scenarios for G and h . In scenarios 1 and 2, the algorithm successfully identified both single and multiple causes in acyclic graphs, judged on the root causes that would have been chosen intuitively. The same holds true for scenario 3, which also includes cyclic dependencies.

Scenario 4, the most complex case, demonstrates two key features of the algorithm. First, the prediction of Node D as root cause for all symptoms means that the algorithm assumes that the fault propagated through the healthy Node A to the left half of the graph. Second, when setting $\theta = 1.0$, only node D is identified as potential root cause, due to its higher distance and chain metric scores compared to those of nodes E and G. Nodes E and G are included in the diagnosis when θ is reduced.

To analyze this behavior systematically, θ was varied from 0.0 to 1.0 in 0.1 increments. This sensitivity analysis has shown how, with decreasing values for θ , more nodes were added to the set of predicted root causes S_{causal} . For example, in Scenario 1, reducing θ from 1.0 to 0.8 adds node A to the diagnosis, and further reducing it to 0.6 includes node C, at which point all initial candidate nodes were also elements of the diagnosis $C = S_{\text{causal}}$. Similarly, Scenario 4 starts with only node D at $\theta = 1.0$. The algorithm progressively adds nodes G, E, and H as the threshold decreases, eventually including all nine nodes at $\theta = 0.5$. This behavior demonstrates that while the algorithm does not explore all possible combinations of candidate nodes, it nevertheless eventually suggests all possible subsystems as potential causes.

These results show that the algorithm generates diagnoses that align with intuitive reasoning in both acyclic and cyclic graphs and therefore provide evidence that supports hypothesis H2.

6.4 Evaluation of the Integrated Diagnosis Framework

This section describes the experiments evaluating the overall approach shown in Figure 5.1, which includes both the symptoms generator \mathcal{G} and the graph diagnosis algorithm \mathcal{A} , addressing hypothesis H3. The following subsection describes a demonstration of the holistic approach on the real-world SWaT dataset, while Subsection 6.4.2 performs a more thorough analysis using a hundred randomly generated dynamical systems.

6.4.1 Evaluation with the SWaT Dataset

In this experiment, the entire diagnostic approach is applied to the SWaT dataset [64], which is a benchmark dataset frequently used in the anomaly detection literature. This dataset is particularly suitable for evaluating hypothesis H3 because it fulfills two requirements: (i) it is well-documented, which allows the construction of

6 Experimental Results

both the causal subsystem graph G and the subsystem-signals map \mathcal{M} , and (ii) it includes detailed information on faults, which facilitates the validation of diagnostic results against ground truth labels.

Dataset and System Structure The SWaT dataset contains 51 signals that describe the physical behavior of a water treatment process both during normal operation and injected faults. The process runs in six phases, P1 to P6, which are shown in Figure 6.6.

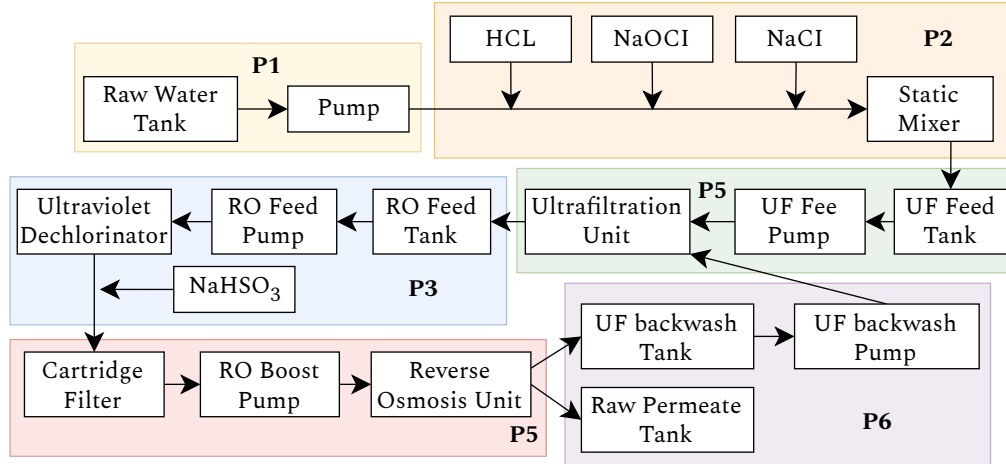


Figure 6.6: SWaT process diagram showing the six phases (P1-P6) that are used as subsystems in the diagnostic approach. Each subsystem represents a distinct process stage. The figure has been redrawn from the original presented in [64].

Experimental Procedure In this experiment, all steps of the overall approach visualized in Figure 5.1 are executed sequentially:

1. **Knowledge Formalization (Phase a):** The subsystem-signals map \mathcal{M} and causal subsystem graph G were constructed based on the dataset documentation provided by Goh et al. [64]. The boundaries of the subsystems were defined to align with the six process phases (P1 through P6) as illustrated in Figure 6.6. The names of the signals in the SWaT dataset contain three-digit codes that provide information about their assignment to the process phases. These three-digit codes were used to define the subsystem-signals map \mathcal{M} . The graph G was modeled according to the process flow shown in Figure 6.6.
2. **Model Training (Phase b):** The training procedure followed the methodology described in Section 5.2, but was supplemented by a few minor steps for data preprocessing and cleaning that were necessary for the specific dataset (which are also part of the public code repository). To address the volatile residuals $r_s(t)$ in normal operation, a conservative threshold (99th percentile) was

employed for residual binarization that was calculated based on the anomaly-free test data. Additionally, a moving median filter was applied to smooth the residuals before binarization.

3. **Model Inference (Phase c):** To evaluate the approach, the fault cases contained in the documentation were first reduced to those that, according to Goh et al. [64], caused an “Actual Change” and for which there was no “unexpected outcome”. This resulted in a dataset in which it can be assumed that the “attack point” is also the cause of the fault s_{true} .

Results and Analysis Table 6.4 summarizes the results of this experiment on the 11 attack scenarios. Each row shows the subsystems targeted in the attack (Ground truth), the subsystems identified as symptomatic by the symptoms generator S_{sym} , and the subsystems predicted as root causes S_{causal} (predicted).

Table 6.4: Attacked subsystems (ground truth), detected symptoms, and diagnosed root cause candidates for the SWaT experiment.

Attack	Ground truth	Symptoms (S_{sym})	Predicted (S_{causal})
#01	P1	P1, P5	P1
#02	P1	P1, P5, P6	P5, P1
#17	<u>P3</u>	<u>P1, P3</u>	<u>P1</u>
#21	P1	P1, P5	P1
#23	P6, P3	P3	P3
#25	<u>P4</u>	<u>-</u>	<u>-</u>
#26	P1, P3	P1, P6	P1
#27	<u>P3, P4</u>	<u>P2</u>	<u>P2</u>
#28	P3	P3, P4, P5	P3
#30	P1, P2	P1	P1
#35	P1	P1, P2, P6	P1

The results demonstrate that in eight of the eleven attack scenarios (73%), at least one of the attacked subsystems was correctly included in the set of predicted root causes S_{causal} . Notably, in three cases (highlighted in bold), the graph diagnosis algorithm successfully narrowed multiple symptomatic subsystems to a single root cause that matched the actual attack point.

However, the approach faced challenges in four cases (underlined). In attack #25, the symptoms generator failed to detect any anomalies. In attacks #17 and #27, the graph diagnosis algorithm identified different subsystems as root causes than those actually attacked. It is also worth noting that in attacks targeting multiple subsystems simultaneously (e.g., #23, #26, #30), the diagnostic approach typically identified only one of the affected subsystems.

These results provide evidence for hypothesis H3 by demonstrating that the diagnostic methodology can be applied to real-world CPS use cases. Although the approach did not provide the perfect diagnosis in all cases, the results were reasonably accurate given the limited prior knowledge and the minimal manual modeling effort required.

6.4.2 Randomly generated Dynamical Systems

Although the experiment on the SWaT dataset demonstrated that the proposed approach can provide helpful diagnoses, it is limited to only one system. Furthermore, neither the correctness of the labels for the root causes nor of the graph G or the subsystem-signals map \mathcal{M} can be guaranteed, as this information was derived from limited documentation. For this reason, controlled simulations are performed in this experiment for which both the graph G and the map \mathcal{M} are known. Furthermore, the anomalies are injected in such a way that the subsystem causing the symptoms is also known with certainty.

Experimental Setup This experiment is based on data simulated using randomly generated dynamic systems. Figure 6.7 visualizes the process steps involved in generating data for one of these systems. First, a randomly generated graph G is sampled, on the basis of which the state matrix $\mathbf{A} \in \mathbb{R}^{|P| \times |P|}$ of a linear dynamic system is created. The non-zero values within this matrix are also generated randomly. If this results in an unstable system, randomly sampled corrections are made until a stable system is achieved.

The state dynamics follow a linear system model $\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} + \mathbf{B}\mathbf{u}$, where $\mathbf{y} \in \mathbb{R}^{|P|}$ represents the state vector, $\mathbf{u} \in \mathbb{R}^m$ the control inputs, and $m \in \mathbb{N}$ the number of steering signals. The matrices \mathbf{A} and $\mathbf{B} \in \mathbb{R}^{|P| \times m}$ define the system's behavior. Normal operation data is generated by stimulating the system with rectangular pulse signals of randomly varying pulse durations. To create anomalous conditions, the parameters in matrix \mathbf{A} for a specific subsystem are slightly altered (also randomly). The subsystem for which the parameters are modified is also sampled randomly and provides the ground truth for the root cause of the fault.

The experimental procedure follows the complete diagnostic workflow as depicted in Figure 6.8. For each simulated system, the normal operation data is split into training and validation sets used to train and validate the symptoms generator respectively. In addition, another split with normal data is created that is not used for training or hyperparameter tuning, but is used to define the thresholds for the residual binarizer. Finally, the now fully initialized solution is applied to the dataset containing the anomalies. This means that first the symptoms generator \mathcal{G} calculates the health state vector \mathbf{h} , on the basis of which the graph diagnosis algorithm \mathcal{A} predicts the set of causal subsystems S_{causal} .

Experimental Trials The experiment comprised 100 trials with varying graph sizes (5-100 nodes), edge densities, noise levels, and anomaly intensities. Each node in

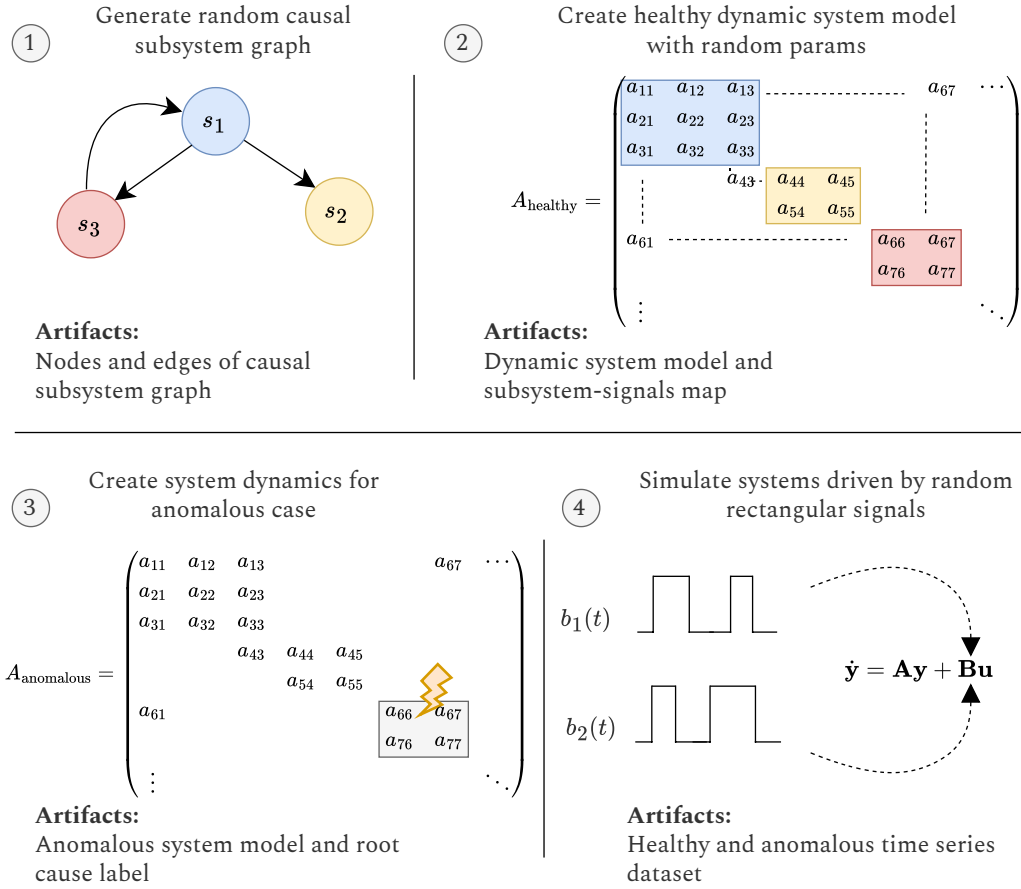


Figure 6.7: Time series data generation process for random dynamic system experiment. The process involves: (1) sampling a random directed graph G , (2) creating a corresponding state matrix with random entries reflecting the graph structure, (3) simulating system dynamics using linear differential equations, and (4) generating multivariate time series for both normal and anomalous conditions.

these simulated systems represented 2-5 signals, such that the resulting dataset sizes ranged from dozens to hundreds of signals. To systematically evaluate the performance of the integrated approach, the trials were categorized into five outcome groups:

- (i) Trials where the causal subsystem s_{true} is not classified as anomalous by the symptoms generator, which means $s_{\text{true}} \notin S_{\text{sym}}$
- (ii) Trials where the causal subsystem is detected as anomalous but not identified as a root cause in the diagnosis, which means $s_{\text{true}} \in S_{\text{sym}}$ but $s_{\text{true}} \notin S_{\text{causal}}$.

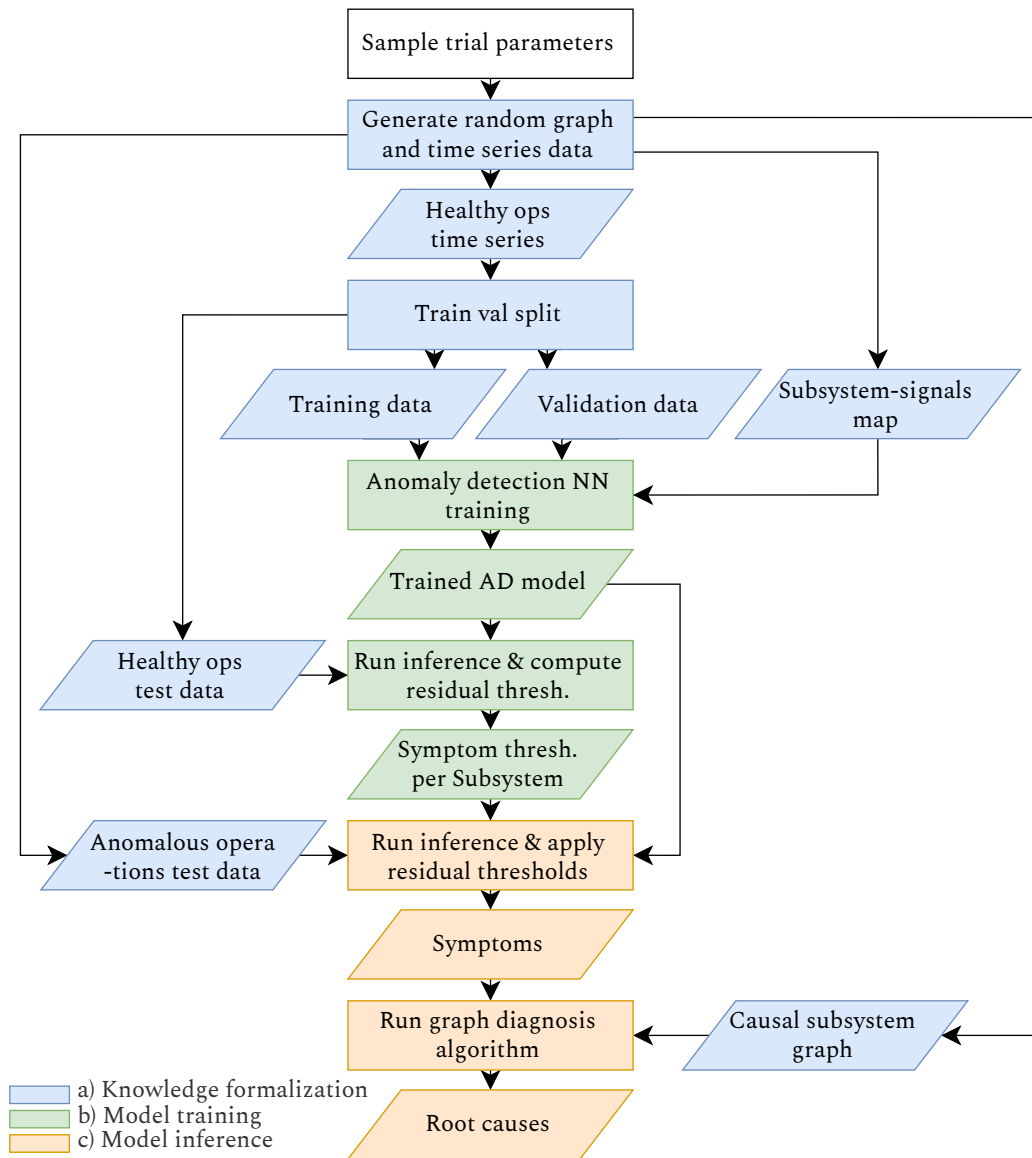


Figure 6.8: Experimental trial for random dynamical system experiment. All 100 trials of the experiment were executed according to the process shown. The experiments are implemented as a KFP pipeline that reflects this DAG. The color coding indicates which phase of the overall solution (see Figure 5.1) the individual steps of this process correspond to.

- (iii) Trials where the causal subsystem is correctly identified but the diagnosis fails to reduce the number of candidates compared to the symptoms, which means $s_{\text{true}} \in S_{\text{sym}}$ and $s_{\text{true}} \in S_{\text{causal}}$ but $|S_{\text{causal}}| \geq |S_{\text{sym}}|$.

- (iv) Trials where the diagnosis successfully reduces the candidate set but still includes multiple subsystems including the true cause, which means $s_{\text{true}} \in S_{\text{causal}}$ and $|S_{\text{causal}}| < |S_{\text{sym}}|$ but $|S_{\text{causal}}| > 1$.
- (v) Trials with perfect diagnosis where the true causal subsystem is identified as the sole root cause, which means $S_{\text{causal}} = \{s_{\text{true}}\}$.

Results and Analysis Figure 6.9 summarizes the outcomes across all 100 experimental trials. The results demonstrate that the integrated diagnostic approach successfully identifies the true causal subsystem as part of the root cause set in 82% of trials (categories (iii), (iv), and (v) combined).

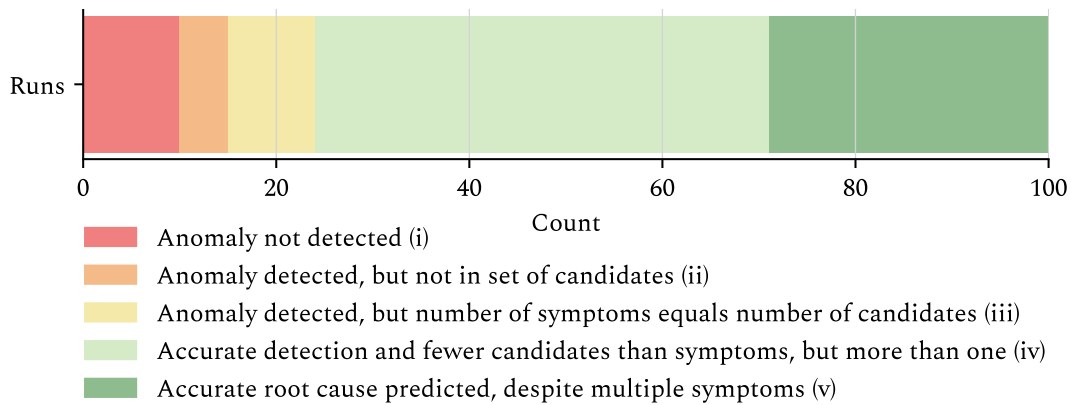


Figure 6.9: Result summary of the random dynamic system experiment across 100 trials with simulated systems. The bars represent the proportion of trials falling into each outcome category.

In 73% of cases (categories (iv) and (v) combined), the method successfully reduces the search space compared to the initial symptoms detection. Notably, 30% of trials achieve perfect diagnosis (category (v)), identifying the true causal subsystem as the sole root cause. The remaining cases include scenarios where the causal subsystem is missed in the final diagnosis (8%, category (ii)) or not detected as anomalous by the symptoms generator (10%, category (i)). These failures primarily occurred in trials with too subtle parameter alterations during anomaly generation. Such cases could potentially be addressed through refined threshold optimization strategies.

The results provide further evidence supporting hypothesis H3 by demonstrating that the integrated diagnostic approach effectively identifies root causes in complex systems with diverse fault propagation patterns. In contrast to the previous experiments, this approach also scales to larger graphs and datasets since some of the generated systems recorded hundreds of signals.

6.5 Evaluation of the MLOps Implementation

This chapter describes how the MLOps pipeline described in Section 5.4 was implemented in the specific case of the ECLSS [44]. Motivated by hypothesis H4, the section investigates the extent to which the tools available in the Kubeflow ecosystem can be combined in an end-to-end workflow to obtain a reproducible ML pipeline suitable for retraining. The focus is on the interfaces between the individual tools that are integrated into the workflow.

6.5.1 Use Case

The ECLSS system manages air temperature, humidity, and circulation within the Columbus module of the ISS [178]. The system consists of multiple interconnected components, including fan assemblies, heat exchangers, and temperature control valves, which are monitored by numerous signals. This particular system is an ideal real-world test case for the MLOps implementation for several reasons. First, it generates hundreds of distinct telemetry parameters collected at a minimum frequency of one Hertz, which creates a data volume and velocity challenge typical of large and complex CPSs. Second, due to component wear, maintenance interventions, special test procedures and setpoint adjustments, this system is prone to concept and data drift, which creates the need for regular retrainings of ML models.

6.5.2 Pipeline Implementation and Integration

Since the focus of this experiment is on MLOps and not on the specifics of the model itself, this section describes the implementation of a pipeline that uses a relatively simple model for anomaly detection in the overall ECLSS system. The pipeline discussed here represents the application of the concept described in Section 5.4. Like all other experiments described in this chapter, this pipeline was executed on a Kubernetes cluster with three nodes, each with 128 CPU cores, 500 GB memory, and a total of four NVIDIA A30 GPUs.

Data Processing and Preparation The data pre-processing challenge, which is particularly pronounced in the ECLSS use case, was addressed with Dask as described in Section 5.4. The ECLSS telemetry data was provided as yearly archives in parquet file format [55]. However, these files are so large that they cannot be easily read with standard tools such as pandas on reasonably sized machines. In order to use parallel computing frameworks such as Dask, these large files first had to be divided into smaller chunks and stored in MinIO [119], the object store used in this specific case. These files were then read and processed in parallel by Dask workers on different nodes of the cluster according to the computational graphs created by the Dask scheduler. The Dask cluster used for the computation was started with the Python Software Development Kit (SDK) from within a KFP component and also deleted again after the computations.

Hyperparameter Optimization The reference implementation suggests Katib for hyperparameter tuning. Since any containerizable code can be executed within KFP components, the Katib SDK or the Kubernetes libraries can also easily be used to start and manage Katib experiments from within a KFP component. However, because KFP manages the data exchange between individual steps in the pipeline, it is not trivial to make the data computed in previous steps of the KFP pipeline available to the Katib processes. To solve this issue, this implementation uses custom code to handle data retrieval from the object store where KFP manages the pipeline artifacts.

Distributed Model Training The implementation leveraged Kubeflow's Training Operators to enable distributed training across multiple GPUs and nodes, utilizing PyTorch's DDP paradigm. Conceptually, integrating distributed PyTorchJobs in the KFP pipeline involved the same challenge as for Katib. Custom solutions were needed to make the pipeline artifacts available to the training processes.

Model Deployment and Inference The final stage of the training pipeline utilized KServe to deploy trained models as serverless inference endpoints. During the training phase, the PyTorchJob logged the model artifacts to an MLflow server, which stored them in an S3-bucket. A conditional execution step was implemented inside the KFP pipeline that checks if the model meets predefined performance metrics. If it does, the next pipeline step will be executed, which deploys the model stored in the S3-bucket using KServe. The entire pipeline can be triggered on demand with updated data, which enables automated retraining cycles.

Integration into Microservice Architecture The ML pipeline described above, implemented in KFP, enables the automated (re)training of ML models. The following paragraphs describe how a model, which is trained and deployed using this pipeline, is integrated into the overall application motivating this work. The deployed software architecture consists of several microservices that can be categorized into two phases corresponding to phases b (model training) and c (model inference) from Chapter 5, as visualized in Figure 6.10.

The Columbus Module's telemetry data stream is connected to Kafka for the consuming streaming services such as the anomaly detection model. The data is also archived in the TimescaleDB database for future data analysis and model training purposes. The training process operates in batch mode on this historical data, utilizing the KFP pipeline described above. The inference components, on the other hand, work continuously on Kafka's data stream.

KServe offers the option of scaling the preprocessing and model inference workloads separately from each other. The CPU-based preprocessing is implemented in a so-called "transformer" (not to be confused with the NN architecture), which is directly connected to the Kafka data stream. The pure ML model, on the other hand, is implemented in a so-called "predictor" that is triggered by the transformer via

6 Experimental Results

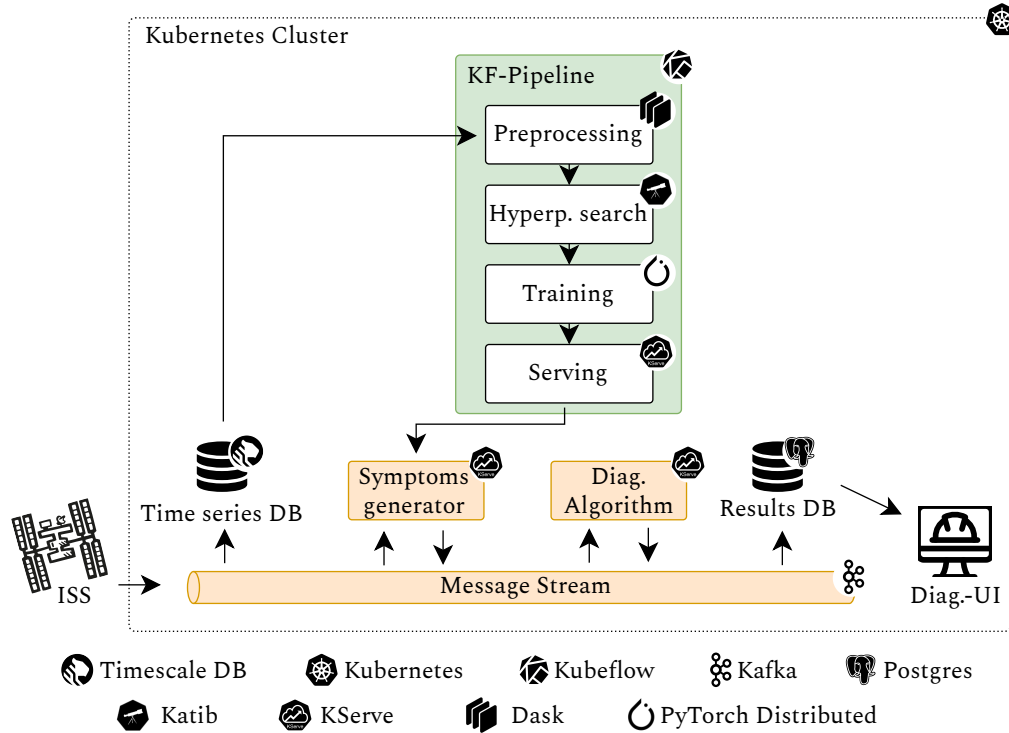


Figure 6.10: Microservice architecture for the ECLSS diagnostic system deployment. The diagram illustrates the integration of various technologies within a Kubernetes cluster including the KF-Pipeline (highlighted in green) and the inference services (highlighted in orange). Data exchange is handled through the message queuing system Kafka and historical time series data is stored in TimescaleDB. The color coding distinguishes between tools used during model training (phase b, green) and those used during continuous inference (phase c, orange) as described in Section 5.1.

API calls. If, for example, preprocessing is the bottleneck, additional replicas of the transformers can be used without requiring additional GPU-intensive predictors.

This implementation of the MLOps pipeline provides evidence supporting hypothesis H4. It demonstrates that it is indeed possible to combine various tools from the Kubeflow ecosystem in one holistic KFP pipeline. However, it still required several custom solutions and workarounds.

7 Theoretical Results

Chapter Outline This chapter presents the theoretical analysis of the graph-based diagnostic reasoning algorithm \mathcal{A} . Section 7.1 discusses the soundness and completeness properties of the algorithm, and Section 7.2 discusses its computational complexity.

7.1 Soundness and Completeness

Soundness and completeness are two fundamental characteristics of diagnostic algorithms. Feldman et al. [53] discuss these properties in detail in the context of model-based diagnosis, and Balzereit and Niggemann [13] in the context of reconfiguration. A diagnosis algorithm is sound if each of the diagnoses it outputs explains all observed symptoms, and it is complete if it is guaranteed that all possible diagnoses are found.

7.1.1 Soundness Analysis

As indicated above, a diagnostic algorithm is sound if every diagnosis explains all observed symptoms. In Algorithm 2, this property is guaranteed by the fact that the search for further diagnosis candidates is only terminated if there are no unexplained symptoms left in the graph.

The algorithm maintains a set of unexplained symptoms U , which is reduced by those symptomatic subsystems that can be explained by the initial diagnosis S_{causal} . If U does not correspond to the empty set $U = \emptyset$ after this step, the next iteration searches for the best candidates to cover the remaining unexplained symptoms (lines 15 through 18 in Algorithm 2):

$$U \leftarrow U \setminus S_{\text{reach}}(b, U) \quad (7.1)$$

where $S_{\text{reach}}(b, U) = \{u \in U \mid \exists \text{ path } b \rightarrow u\}$. The algorithm terminates if one of the following conditions is true:

1. $U = \emptyset$: All symptoms can be explained by candidate nodes in the diagnosis S_{causal} .
2. $C = \emptyset$: No more candidates are available.

For condition (1), soundness follows directly since every symptom in S_{sym} has been explained by at least one root cause in S_{causal} . As for condition (2), since every node has the trivial path to itself, every symptomatic node is also in the set of candidates:

$$C = \{v \in S \mid \exists s \in S_{\text{sym}} : \text{path}(v \rightarrow s)\} \quad (7.2)$$

which means that the case $U \neq \emptyset$ and $C = \emptyset$ theoretically cannot occur. The termination condition was added to make the implementation robust against edge cases.

Therefore, the algorithm is sound: every diagnosis S_{causal} can theoretically explain all symptoms in S_{sym} .

7.1.2 Completeness Considerations

This section describes the analysis regarding the completeness of the graph diagnosis algorithm \mathcal{A} . As already indicated in Section 5.3, the algorithm does not satisfy completeness in the diagnostic sense since it does not consider all potential combinations of subsystems that could explain the observed symptoms.

A conceivable implementation of an algorithm that satisfies completeness would evaluate all possible subsets of these candidates $2^{|C|}$ instead of the individual candidates $c \in C$. Among other things, it would need to check for each subset $S' \subseteq C$ whether all symptoms are in the union of the sets of nodes which can be reached by the individual candidates in S' . It is theoretically possible to implement such an algorithm, however the exponential number of subsets ($2^{|C|}$) makes this approach computationally prohibitive for large graphs, as those used in Section 6.4.2.

A formal proof of the incompleteness of Algorithm 2 can be constructed through a counterexample: Consider a graph with symptoms at nodes $s_x, s_y,$ and s_z . Suppose node s_a can explain s_x with high score, while nodes s_b and s_c together can explain all symptoms $s_x, s_y,$ and s_z , with lower individual scores than s_a . In the first iteration, s_a will be selected due to its high score (line 14 in Algorithm 2). The algorithm then updates the unexplained symptoms, removing s_x . In the next iteration, either s_b or s_c could be selected to explain the remaining symptoms s_y and s_z . Thus, the potentially better explanation set $\{s_b, s_c\}$ is never considered.

This disadvantage can be mitigated in practice by using the sensitivity parameter θ . If the diagnosis initially proposed by the algorithm does not prove to be correct, the parameter can be reduced to add further subsystems to the diagnosis. When θ reaches $\theta = 0$, the set of diagnoses equals that of the candidates $S_{\text{causal}} = C$. This means that although not all combinations of candidates are considered as root causes, all candidate subsystems are eventually considered individually.

7.2 Complexity Analysis

This section analyzes the time complexity of the graph diagnosis algorithm \mathcal{A} . According to the notation defined in Chapter 4, S describes the set of nodes (subsystems) and E describes the set of edges (fault propagation paths) in graph $G = (S, E)$.

Main Loop The main loop of the algorithm is executed when $U \neq \emptyset$ and $C \neq \emptyset$, where U describes the set of unexplained symptoms and C describes the set of diagnosis candidates. Since at least one candidate is removed from C per iteration, and $|C| \leq |S|$, there are at most $|S|$ iterations of the main loop in the worst case.

Per Iteration For each iteration, the algorithm performs the following computations:

1. Computation of scores for all candidates:
 - `compute_reachability_scores`: $\mathcal{O}(|S| \times (|S| + |E|))$ for all candidates. This complexity derives from running graph traversal for each candidate node $c \in C$, which has a complexity of $\mathcal{O}(|S| + |E|)$ per traversal [36].
 - `compute_distance_scores`: $\mathcal{O}(|S| \times (|S| + |E|))$, based on shortest path calculations through graph traversal for each candidate $c \in C$.
 - `compute_anomaly_scores`: $\mathcal{O}(|S|)$, since it only checks the membership of c in S_{sym} , which can be done in constant time.
 - `compute_anomaly_chain_scores`: $\mathcal{O}(|S| \times (|S| + |E|))$, which also requires graph traversal with the same complexity as the other scoring methods.
2. Selection of best candidates and update: $\mathcal{O}(|S|)$ to select the best candidates based on the computed scores and update the sets U and C .

Overall Complexity Combined, the steps per iteration result in a complexity of $\mathcal{O}(|S| \times (|S| + |E|))$. Since the main loop is executed at most $|S|$ times, the total time complexity is $\mathcal{O}(|S|^2 \times (|S| + |E|))$.

For dense graphs where $|E| \approx |S|^2$, this simplifies to $\mathcal{O}(|S|^4)$, while for sparse graphs where $|E| \approx |S|$, the complexity becomes $\mathcal{O}(|S|^3)$, which means the algorithm has polynomial time complexity with regard to the number of nodes in the graph.

8 Discussion

Chapter Outline This chapter discusses the experiments described in Chapters 6 and 7 as well as their results. The structure of this discussion is aligned with the key contributions of this thesis: the subsystem-level symptom generation (Section 8.1), the graph-based diagnostic reasoning algorithm (Section 8.2), the application of the combination of those two (Section 8.3), and the MLOps implementation (Section 8.4). These contributions and the experimental and theoretical findings are critically discussed based on the hypotheses set out in Section 6.1.

8.1 Subsystem-Level Anomaly Detection Architecture

Hypothesis H1 claims that the proposed composite latent space architecture in the symptoms generator improves anomaly detection at the subsystem level compared to state-of-the-art approaches. Parts of the proposed architecture, in particular TCNs and VAEs, have already been benchmarked on real CPS data and have achieved good results [61, 192, 194]. However, to the best of the author’s knowledge, there are no publicly available datasets for isolating symptoms at the subsystem level. Therefore, the experiment described in Section 6.2 compares the performance of the proposed model against benchmarks based on a dataset created specifically for this experiment.

Apart from this limitation, the results provide evidence that supports hypothesis H1. In the experiments, the proposed model outperforms all benchmark models in subsystem-level anomaly detection, measured by the F1 score. The composite latent space structure helps to isolate faults, which clearly distinguishes it from approaches that model all signals simultaneously. On the other hand, unlike univariate models, it is able to model patterns between variables and even detect anomalies that can only be identified when multiple subsystems are observed simultaneously.

In addition to the obvious limitation that no real-world datasets were used, the evaluation of the hypothesis could benefit from more complex example systems as well as from more complex fault scenarios. However, these points are implicitly mitigated in part by the use of the symptoms generator in the experiments on the integrated overall framework.

In summary, the findings show that the composite latent space approach successfully addresses the trade-off between system-wide anomaly detection capabilities and subsystem-level fault isolation, despite the limitations discussed above.

8.2 Graph-Based Diagnostic Algorithm

According to hypothesis H2, the graph diagnosis algorithm \mathcal{A} is capable of identifying causal subsystems in both cyclic and acyclic fault propagation graphs.

Both the experiment based on the TEP example and the four scenarios created for the purpose of this evaluation provide evidence supporting this hypothesis. The algorithm includes the true causal subsystem in its diagnosis in all of the four TEP faults analyzed. The scenarios created specifically for this experiment represent more complex diagnostic tasks including cyclic graphs, in which the algorithm also consistently computes diagnoses that correspond to the intuitively correct diagnosis. In addition, the sensitivity analysis shows how, as the sensitivity parameter θ decreases, more subsystems are included in the diagnosis.

However, the experiments and the results of the theoretical analysis in Chapter 7 also illustrate the limitations of the approach. Although the algorithm is sound in the diagnostic sense, it is not complete. There is therefore neither a guarantee that the best diagnosis will be found, nor that all possible diagnoses can be computed. Furthermore, the high level of aggregation brings limitations. For example, neither temporal aspects nor the causal connections between subsystems or the possibility of causal “and” links are supported by the graph or the algorithm. In addition, the binary health states represent a very high aggregation compared to the behavioral models often used in the diagnostic literature [53, 126].

These limitations were explicitly accepted during the design of the algorithm in favor of a solution that minimizes the necessary prior knowledge and modeling effort. The results show that the algorithm, when provided with accurate graph G and health state \mathbf{h} , can represent added value in the maintenance process.

8.3 Integrated Diagnostic Framework

The third hypothesis, H3, assumes that the combination of the symptoms generator \mathcal{G} and the graph diagnosis algorithm \mathcal{A} is suitable as an overall solution for identifying root causes of faults in CPS data. The corresponding results provide supporting evidence for this assumption.

As the results in Section 6.4.1 show, the overall framework identified the causal subsystems in most of the analyzed cases in the real SWaT system. In some cases, the set of identified symptoms was larger than that of the root causes, which particularly illustrates the effectiveness of the graph algorithm. The results of the random trials in Section 6.4.2 further illustrate the benefits of the overall framework. In more complex setups than in the SWaT example, with up to one hundred subsystems and datasets that contained several hundred signals, the overall framework identified the true causal subsystem as the cause in a substantial majority of cases.

The results show how the overall framework can provide helpful diagnoses despite very limited prior system knowledge. This is only possible through the interaction of the two components. The symptoms generator is designed so that the

subsystems modeled in the graph can be annotated with health states. The ability to isolate symptoms to subsystems in turn enables the localization of the most probable diagnosis in the graph algorithm.

However, some limitations need to be considered here too: First, the performance on the SWaT dataset showed more cases of incorrect or incomplete diagnoses compared to the simulated systems, which might be due to the complexity inherent in real data, or the fact that the constructed graph G might not reflect the true causality in the system. Second, the approach showed limitations in the identification of multiple concurrent faults. In these cases, often only one of several simultaneously affected subsystems was identified.

Despite these considerations, the integrated approach demonstrated good performance in both simulated and real-world data experiments.

8.4 MLOps Implementation

The final hypothesis, H4, claims that within a KFP pipeline, the entire retraining process of a ML model can be automated. Section 6.5 describes the implementation of such a pipeline for an anomaly detection model of the ECLSS, and thus confirms the hypothesis.

The implementation addresses the core requirements often posed on ML applications for complex CPSs: distributed data processing for extensive time series data, systematic hyperparameter optimization, scalable model training, and continuous monitoring with automated retraining. These functions are conceptually described by the reference architecture presented in the solution chapter and demonstrated in a concrete case for the ECLSS.

However, several limitations should be noted here: (i) The reference pipeline was only demonstrated in a single use case. The tools and the mechanisms for integrating them into the pipeline were also used for the other experiments and are use-case agnostic. Nevertheless, further use cases, especially those used in production, would strengthen the argument. (ii) There is no comparison to implementations in alternative MLOps platforms. These platforms might offer different trade-offs in functionality and ease of implementation. (iii) Finally, while the implementation demonstrates the retraining procedure, it lacks detailed information on how to trigger retrains. Further exploration of model performance monitoring would be beneficial.

The integration challenges encountered during implementation, particularly between components like Katib, PyTorchJobs and KFP, are areas where the KubeFlow ecosystem could benefit from more focus on interoperability. However, despite these challenges, the pipeline was implemented successfully, which provides evidence for hypothesis H4.

Part IV

Conclusion and Outlook

9 Conclusion

Chapter Outline This chapter wraps up this thesis and discusses the extent to which the research questions raised in Chapter 1 have been addressed. In addition, Section 9.2 and Section 9.3 describe the strengths and limitations of the contributions of this work respectively.

9.1 Addressing the Research Questions

This thesis presents a diagnostic method for complex CPSs that was developed under the premise of minimizing the necessary prior knowledge and modeling effort. The research was guided by three questions, which are discussed in the following paragraphs.

RQ1 asks how health states for CPS subsystems can be generated that are suitable for downstream diagnosis methods without requiring extensive modeling efforts. With regard to this question, this thesis presents a so-called symptoms generator. One part of this generator is a novel NN architecture that, in addition to using TCNs in the individual decoder and encoder networks, makes use of a newly developed composite latent space structure. Unlike other state-of-the-art approaches, this structure allows symptoms to be isolated to individual subsystems without losing the ability to detect anomalies that are only observable when monitoring the system as a whole. The corresponding experiments show how reconstruction-based models that model all signals simultaneously in large NNs are unable to guarantee anomaly isolation. The proposed approach can guarantee this without losing the advantages related to multivariate time series modeling and thereby answers RQ1.

RQ2 asks how symptoms at the subsystem level, like those discussed in RQ1, can be used together with basic information about fault propagation paths in the CPS for diagnostic reasoning. Or in other words, how can the subsystems that cause the observed symptoms be identified? This question is also based on the premise that as little prior knowledge and modeling effort as possible should be required. In response to these questions, this thesis presents a new graph-based diagnostic reasoning algorithm. In addition to the health states at the subsystem level, the only input required for this algorithm is a fault propagation graph that represents the subsystems as nodes and the possible fault propagation paths in the CPS as edges. The algorithm is based on graph theory and, unlike many other diagnostic methods, can also be applied to cyclic graphs. It computes the most likely diagnosis based on fundamental engineering principles, while maintaining polynomial time complexity, which means it can also be applied to large graphs. This concept builds on the

well-isolated health states from RQ1, which are specifically designed to match the granularity of the fault propagation graph. The experiments demonstrate both the functionality of this algorithm in isolation and the effectiveness of the combination of the symptoms generator and the graph diagnosis algorithm, thereby providing an answer to RQ2.

Finally, RQ3 addresses the more operational question of how the deployment and maintenance of methods such as those described above can be supported by modern MLOps tools. More specifically, it addresses the question of whether the entire training and retraining process of ML software for complex CPS can be orchestrated within a pipeline implemented in Kubeflow. This question was answered with a reference implementation of an end-to-end KFP pipeline designed for such cases and its demonstration on the specific use case of an anomaly detection model for the ECLSS. This implementation differs from the state of the art in that the tools used in this pipeline have previously only been considered in isolation. The fully automated pipeline presented in this thesis integrates tools for distributed preprocessing, hyperparameter tuning, distributed model training, experiment tracking, and deployment of the model as an API, thereby answering RQ3.

9.2 Key Strengths

The diagnostic approach presented in this thesis demonstrates several strengths compared to the state of the art.

One of these strengths lies in the abstraction of the diagnostic problem to a level that lies between the individual, mostly continuous sensor readings and the overall system. Through this aggregation, the actual diagnostic problem is transformed from the high-dimensional continuous space of time series monitoring into a more tractable diagnostic problem based on binary inputs.

Another strength of this approach is its low requirement for formalized prior system knowledge. Unlike model-based diagnostic methods that require detailed behavioral models, or supervised approaches that depend on comprehensive fault datasets, the proposed methodology requires only three inputs: (i) training data from nominal system operation, which is typically collected for monitoring purposes, (ii) a subsystem-signals map, which can often be derived automatically, and (iii) a graph representing basic fault propagation relationships, which requires far less detail than behavioral models. However, it is important to note that the framework must be tested on fault cases before it is put into production. If no historical faults are available, artificially injected data or simulation data can be used for this purpose.

With its composite latent space NN architecture, the proposed symptoms generator provides two key advantages: (i) The symptom isolation on subsystems is stronger than in conventional approaches without sacrificing the advantages of multivariate sequence models, and (ii) the health states generated are precisely

at the aggregation level used by the fault propagation graph, which enables the effective combination of anomaly detection and diagnosis.

A second key point is the proposed framework's modular design, which allows for adapting and extending it in multiple ways: (i) the encoder and decoder components within the composite latent space architecture can be replaced with alternative NN architectures, (ii) any classification method can be applied to the anomaly scores in order to transform them into binary health states, (iii) the entire symptoms generator could be replaced with any method capable of producing subsystem-level binary health states, whether based on NNs, statistical approaches, or domain-specific algorithms, as long as it fits the subsystems defined in the graph and (iv) the scoring functions in the diagnostic algorithm can be weighted differently, modified or replaced entirely to accommodate specific system characteristics and diagnostic priorities.

The same holds true for the MLOps implementation, which leverages a microservices architecture where individual components can be modified or replaced easily. The containerized approach ensures that specific tools within the pipeline can be exchanged with alternative implementations as long as they are run inside containers, which most popular ML frameworks do.

Lastly, the polynomial time complexity of the graph diagnosis algorithm represents an advantage. By considering the subsystem level rather than all components individually, and by employing heuristic scoring rather than exhaustive consistency checking, the approach avoids the combinatorial explosion. However, these design choices also lead to limitations, which are discussed in the next section.

9.3 Limitations

Despite the strengths of the proposed diagnostic approach, some limitations must be highlighted.

A major limitation is that the diagnostic reasoning algorithm does not take any temporal dynamics into account. While the health states that the algorithm receives as input have been generated by a DL model that is capable of modeling sequence patterns in time series, the graph algorithm only operates on snapshots in time. If symptoms propagate through the graph with time delays, the algorithm will likely compute poorly informed diagnoses.

Another limitation lies in the fact that the effectiveness of the diagnostic algorithm heavily depends on the accuracy of the causal subsystem graph. If causal relationships are missing or incorrectly represented in the graph, the algorithm will likely not correctly identify the true root causes of observed symptoms. As described in Chapter 3, there are methods to automatically extract similar graphs from time series data, but it remains to be explored whether they can be used in a setup like the one proposed in this thesis.

Regarding the MLOps implementation, while the symptoms generator can adapt to changing system behavior through the automated retraining pipeline, more se-

9 Conclusion

vere changes in the CPS still present challenges. The execution of the retraining pipeline would not be sufficient if, for example, the subsystem-signals map changed.

Additionally, the current implementation leaves open questions regarding concrete strategies for when to trigger retraining runs and how to validate model performance before deployment. The monitoring of unsupervised anomaly detection is particularly challenging in cases when anomalies themselves are scarce. In addition, the rapidly changing MLOps tool landscape poses a challenge because the implementation of entire MLOps platforms in corporations is not inexpensive and, once they are in production, may already be outdated.

10 Outlook

Chapter Outline This chapter discusses potential future research directions and a long-term vision based on the contributions presented in this thesis.

10.1 Advancing the Subsystem-Level Anomaly Detection Architecture

The symptoms generator might be enhanced by integrating several promising methods from different research areas.

First, the integration of physics-informed priors, as used for example by [196], into the NN architecture might lead to increased performance of the model. This approach could enable extrapolation to unseen but nominal regions beyond the training data distribution, thereby reducing false positives. While the architecture proposed in the symptoms generator does integrate prior knowledge on the system structure, it does not enforce or value the adherence to physical laws, as PINNs do.

Second, while the proposed model supports both subsystem-level and whole-system level anomaly detection, it could be extended to support any user-defined aggregation level. This might be useful for deeper diagnostic reasoning but would require more sophisticated logic in the diagnosis algorithm, which currently only supports one level of abstraction.

Finally, foundation models have gained significant recognition in text processing and computer vision in recent years, as pointed out in a recent review [11]. This trend has also encouraged researchers to investigate foundation models in the area of time series modeling [106]. The application of pre-trained and potentially fine-tuned foundation models in the symptoms generator might also improve its detection performance.

10.2 Enhancing the Graph-Based Diagnostic Algorithm

The graph-based diagnostic reasoning algorithm could be optimized using various approaches to address the limitations mentioned in Section 9.3.

A major improvement in diagnostic quality could be achieved by taking temporal effects in fault propagation into account, as described, for example, by Duan et al. [47]. The current algorithm completely neglects these effects.

In addition to temporal effects, other aspects could also be considered in more detail in the graph representation and the algorithm. For instance, extending the causal graph to include both disjunctive (OR) and conjunctive (AND) connections

would allow for modeling scenarios where multiple conditions must be satisfied simultaneously for a fault to propagate.

Another enhancement direction could involve formal knowledge representation approaches, such as ontologies [189]. This approach could enhance the causal graph with richer semantic descriptions and thus also increase the information gain for the maintenance engineers using the solution.

Finally, the solution could be enhanced by probabilistic reasoning to address uncertainty. By representing causal relationships with conditional probabilities, as used in [16], rather than the simple deterministic binary ones used in this thesis's approach, the algorithm could provide confidence intervals for diagnoses.

10.3 MLOps Advancements

The MLOps implementation could be further improved in several ways. First, while the open-source community has made significant progress with tools like Kubeflow that help data scientists transition from ML experiments to production software, deploying ML solutions like the one described in this thesis remains non-trivial. At the time of writing this thesis, implementing a complex pipeline requires substantial knowledge of containerization technologies and Kubernetes basics.

Second, the topic of model performance monitoring and concept drift detection is not yet supported out of the box by many MLOps tools, such as MLflow and Kubeflow Berberi et al. [15]. Here, custom solutions still need to be built.

Finally, given the highly fragmented and diverse landscape of MLOps platforms and tools, standard interfaces between the tools used during the ML-lifecycle could make it easier for practitioners and data scientists to use them and to migrate workloads from one platform to another.

10.4 Long-Term Vision

Given the motivation behind this thesis, there is an obvious long-term vision to develop a completely data-driven diagnostic system that does not require any prior knowledge and still provides accurate and informative diagnoses. Such a solution could involve extending the approaches proposed in this thesis or similar approaches to include the automated discovery of system structure information, causal relationships, or fault propagation paths from the system documentation or from time series data. Several promising approaches are already being investigated in this domain, including methods that learn similar graphs from sensor data [6], approaches that learn diagnostic rules based on learned discrete system states [125], and techniques that study the use of LLMs for generating formal system models [117]. However, these approaches have not yet been combined into a holistic diagnostic framework.

Even if such a framework existed, the challenge of concept and data drift still remains and is particularly important if structure-related information is learned from

data and later used in downstream tasks. While MLOps tools and workflows work well for standard ML tasks, it remains unclear how effectively they can be applied to such complex discovery and adaptation tasks.

Another promising avenue for future research has been opened up by recent breakthroughs in research on LLMs and, in particular, LLM-based agents. For example, the modular components of the diagnostic framework presented in this work could be exposed as tools to an LLM-agent through standardized protocols like the Model Context Protocol [82]. The agent could then leverage these tools, along with various other methods for time series analysis and diagnosis, to perform step-by-step reasoning that includes calls to these specialized tools when needed. This type of agent could be further enriched by retrieval-augmented generation [101], in order to provide it with access to system documentation, maintenance records, and historical fault data.

This idea could even be transferred to a multi-agent setup, where each of the agents would be focusing on different aspects like data preprocessing, anomaly detection, causal reasoning, or maintenance recommendation.

Bibliography

1. M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zhang. "TensorFlow: A system for large-scale machine learning". *USENIX Symposium on Operating Systems Design and Implementation*, 27, 2016, pp. 265–283.
2. A. Agarwala, A. Das, B. Juba, R. Panigrahy, V. Sharan, X. Wang, and Q. Zhang. "One network fits all? Modular versus monolithic task formulations in neural networks". *International Conference on Learning Representations (ICLR)* abs/2103.15261, 29, 2021.
3. T. S. Alemayehu, J.-H. Kim, and W.-D. Cho. "Optimal replacement model for the physical component of safety critical smart-world CPSs". *Journal of ambient intelligence and humanized computing* 13, 10 2022, pp. 4579–4590. ISSN: 1868-5137,1868-5145.
4. C. K. Assaad, I. Ez-Zejjari, and L. Zan. "Root cause identification for collective anomalies in time series given an acyclic summary causal graph with loops". In: *International Conference on Artificial Intelligence and Statistics*. 2023, pp. 8395–8404.
5. 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* 132, 2022, p. 108945.
6. J. L. Augustin and O. Niggemann. "Self-Supervised Graph Structure Learning for Cyber-Physical Systems". *IFAC-PapersOnLine* 58, 4 2024. 12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2024, pp. 204–209. ISSN: 2405-8963.
7. K. Authors. *KServe*. Accessed: June 15, 2025. 2025. URL: <https://github.com/kserve/kserve>.
8. K. Authors. *Training Operator*. Accessed: June 11, 2025. 2025. URL: <https://github.com/kubeflow/training-operator>.
9. K.P. Authors. *KFP*. Accessed: June 15, 2025. 2025. URL: <https://www.kubeflow.org/docs/components/pipelines/>.
10. V.-G. M. und Automatisierungstechnik. *Cyber-Physical Systems: Chancen und Nutzen aus Sicht der Automation*. cit. on p. 3. 2013.

Bibliography

11. M. Awais, M. Naseer, S. Khan, R. M. Anwer, H. Cholakkal, M. Shah, M.-H. Yang, and F. S. Khan. "Foundation Models Defining a New Era in Vision: A Survey and Outlook". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 47, 4 2025, pp. 2245–2264.
12. S. Bai, J. Z. Kolter, and V. Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling". *arXiv preprint arXiv:1803.01271*, 2018.
13. K. Balzereit and O. Niggemann. "Sound and complete reconfiguration for a class of hybrid systems". In: *Proceedings of the 32nd international workshop on principle of diagnosis, Hamburg, Germany*. 2021, pp. 13–15.
14. M. Basseville and I. V. Nikiforov. *Detection of abrupt changes: theory and application*. Vol. 104. Prentice hall Englewood Cliffs, 1993.
15. L. Berberri, V. Kozlov, G. Nguyen, J. Sáinz-Pardo Díaz, A. Calatrava, G. Moltó, V. Tran, and Á. López García. "Machine learning operations landscape: platforms and tools". *Artificial intelligence review* 58, 6 17, 2025. ISSN: 0269-2821,1573-7462.
16. R. Bhagavathi, D. K. M. Kufoalor, and A. Hasan. "Digital Twin-Driven Fault Diagnosis for Autonomous Surface Vehicles". *IEEE Access* 11, 2023, pp. 41096–41104.
17. D. G. Bobrow and J. Whalen. "Community Knowledge Sharing in Practice: The Eureka Story". *Reflections* 4, 2 2002.
18. J. A. Bondy and U. S. R. Murty. *Graph theory*. Springer Publishing Company, Incorporated, 2008.
19. L. Bontemps, V. L. Cao, J. McDermott, and N.-A. Le-Khac. "Collective anomaly detection based on long short-term memory recurrent neural networks". In: *Future Data and Security Engineering*. Lecture notes in computer science. Springer International Publishing, Cham, 2016, pp. 141–152. ISBN: 9783319480565,9783319480572.
20. M. Bozzano, A. Cimatti, M. Gario, and A. Micheli. "SMT-Based Validation of Timed Failure Propagation Graphs". In: *Proceedings of the AAI Conf. on Artificial Intelligence*. Vol. 29. 2015.
21. D. Bundesregierung. *Strategie Künstliche Intelligenz der Bundesregierung*. cit. on p. 3. 2018.
22. A. Bunte, B. Stein, and O. Niggemann. "Model-Based Diagnosis for cyber-physical production systems based on machine learning and residual-Based Diagnosis models". *Proceedings of the AAI Conference on Artificial Intelligence*. *AAI Conference on Artificial Intelligence* 33, 01 17, 2019, pp. 2727–2735. ISSN: 2159-5399,2374-3468.

23. C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner. "Understanding disentangling in β -VAE". *arXiv [stat.ML]*, 10, 2018. arXiv: 1804.03599 [stat.ML].
24. D. Campos, T. Kieu, C. Guo, F. Huang, K. Zheng, B. Yang, and C. S. Jensen. "Unsupervised time series outlier detection with diversity-driven convolutional ensembles". *Proc. VLDB Endow.* 15, 3 2021, pp. 611–623. ISSN: 2150-8097.
25. E. Casalicchio and S. Iannucci. "The state-of-the-art in container technologies: Application, orchestration and security". *Concurrency and Computation: Practice and Experience* 32, 17 2020, e5668.
26. J. Cen, Z. Yang, X. Liu, J. Xiong, and H. Chen. "A review of data-driven machinery fault diagnosis using machine learning algorithms". *Journal of Vibration Engineering & Technologies* 10, 7 2022, pp. 2481–2507. ISSN: 2523-3920,2523-3939.
27. V. Chandola, A. Banerjee, and V. Kumar. "Anomaly detection: A survey". *ACM computing surveys (CSUR)* 41, 3 2009, pp. 1–58.
28. X. Chen, L. Deng, F. Huang, C. Zhang, Z. Zhang, Y. Zhao, and K. Zheng. "Daemon: Unsupervised anomaly detection and interpretation for multivariate time series". In: *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 2021, pp. 2225–2230.
29. Z. Chen, D. Chen, X. Zhang, Z. Yuan, and X. Cheng. "Learning graph structures with transformer for multivariate time-series anomaly detection in IoT". *IEEE internet of things journal* 9, 12 15, 2022, pp. 9179–9189. ISSN: 2327-4662,2372-2541.
30. L. H. Chiang, E. L. Russell, and R. D. Braatz. *Fault detection and diagnosis in industrial systems*. Springer Science & Business Media, 2000.
31. L. Chittaro and R. Ranon. "Hierarchical model-based diagnosis based on structural abstraction". *Artificial intelligence* 155, 1-2 2004, pp. 147–182. ISSN: 0004-3702,1872-7921.
32. K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. "Learning Phrase Representations Using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 2014, pp. 1724–1734.
33. T. Cody, S. Adams, P. Beling, and L. Freeman. "On Valuing the Impact of Machine Learning Faults to Cyber-Physical Production Systems". In: *2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*. 2022, pp. 1–6.

Bibliography

34. G. Coelho, P. Pereira, L. Matos, A. Ribeiro, E. C. Nunes, A. Ferreira, P. Cortez, and A. Pilastrri. "Deep dense and convolutional autoencoders for machine acoustic anomaly detection". In: *Artificial Intelligence Applications and Innovations: 17th IFIP WG 12.5 International Conference, AIAI 2021, Hersonissos, Crete, Greece, June 25–27, 2021, Proceedings* 17. 2021, pp. 337–348.
35. L. Console and O. Dressler. "Model-based diagnosis in the real world: lessons learned and challenges remaining". In: *IJCAI*. Vol. 99. 1999, pp. 1393–1400.
36. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2022.
37. E. Dai and J. Chen. "Graph-Augmented Normalizing Flows for Anomaly Detection of Multiple Time Series". In: *International Conference on Learning Representations*. 2022.
38. A. Deng and B. Hooi. "Graph neural network-based anomaly detection in multivariate time series". *Proceedings of the ... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence* 35, 5 18, 2021, pp. 4027–4035. ISSN: 2159-5399,2374-3468.
39. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *arXiv [cs.CL]*, 11, 2018. arXiv: 1810.04805 [cs.CL].
40. A. Diedrich, P. Deutschmann, and C. Junker. "ServiceNavigator - A Bayesian Assistance System for Diagnosing Industrial Production Systems". In: *2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS)*. 2022, pp. 1–6.
41. A. Diedrich and O. Niggemann. "On residual-based diagnosis of physical systems". *Engineering applications of artificial intelligence* 109, 104636 2022, p. 104636. ISSN: 0952-1976,1873-6769.
42. R. Diestel. *Graph Theory*. 5th. Springer Publishing Company, Incorporated, 2017. ISBN: 9783662536216.
43. J. J. Downs and E. F. Vogel. "A plant-wide industrial process control problem". *Computers & Chemical Engineering* 17, 3 1993. Industrial challenge problems in process control, pp. 245–255. ISSN: 0098-1354.
44. J. Doyé. "An advanced columbus thermal and environmental control system". In: *SpaceOps 2012*. 2012, p. 1292285.
45. B. Du, X. Sun, J. Ye, K. Cheng, J. Wang, and L. Sun. "GAN-Based Anomaly Detection for Multivariate Time Series Using Polluted Training Set". *IEEE Transactions on Knowledge and Data Engineering* 35, 12 2023, pp. 12208–12219.

46. M. Du, F. Li, G. Zheng, and V. Srikumar. "DeepLog". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. CCS '17: 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas Texas USA). ACM, New York, NY, USA, 30, 2017. ISBN: 9781450349468.
47. S. Duan, C. Zhao, and M. Wu. "Multiscale Partial Symbolic Transfer Entropy for Time-Delay Root Cause Diagnosis in Nonstationary Industrial Processes". *IEEE Transactions on Industrial Electronics* 70, 2 2023, pp. 2015–2025.
48. J. Ehrhardt, P. Overlöper, D. Vranjes, H. S. Steude, A. Diedrich, and O. Niggemann. "Using Modular Neural Networks for Anomaly Detection in Cyber-Physical Systems". In: *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2024, pp. 01–07.
49. B. Eiteneuer and O. Niggemann. "Lstm for model-based anomaly detection in cyber-physical systems". *arXiv preprint arXiv:2010.15680*, 2020.
50. T. Escobet, A. Bregon, B. Pulido, and V. Puig. *Fault diagnosis of dynamic systems*. Springer, 2019.
51. O. M. Ezeme, Q. H. Mahmoud, and A. Azim. "DReAM: Deep recursive attentive model for anomaly detection in kernel events". *IEEE access: practical innovations, open solutions* 7, 2019, pp. 18860–18870. ISSN: 2169-3536.
52. S. Falas, C. Konstantinou, and M. K. Michael. "Special Session: Physics-Informed Neural Networks for Securing Water Distribution Systems". In: *2020 IEEE 38th International Conference on Computer Design (ICCD)*. 2020, pp. 37–40.
53. A. Feldman, G. Provan, and A. Van Gemund. "Approximate model-based diagnosis using greedy stochastic search". *Journal of Artificial Intelligence Research* 38, 2010, pp. 371–413.
54. K. D. Forbus. "Qualitative Process Theory". *Artificial intelligence* 24, 1-3 1984, pp. 85–168.
55. A. S. Foundation. *Apache Parquet: A columnar storage format for the Hadoop ecosystem*. Accessed: June 15, 2025. 2025. URL: <https://parquet.apache.org/>.
56. C. N. C. Foundation. *Kubeflow brings MLOps to the CNCF Incubator*. Accessed: June 15, 2025. 2023. URL: <https://www.cncf.io/blog/2023/07/25/kubeflow-brings-mlops-to-the-cncf-incubator/>.
57. E. Frisk, M. Krysander, and D. Jung. "A toolbox for analysis and design of model based diagnosis systems for large scale models". *IFAC-PapersOnLine* 50, 1 2017, pp. 3287–3293. ISSN: 2405-8971,2405-8963.
58. M. Fullen, P. Schüller, and O. Niggemann. "Semi-supervised case-based reasoning approach to alarm flood analysis". In: *Machine Learning for Cyber Physical Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2020, pp. 53–61. ISBN: 9783662590836,9783662590843.

Bibliography

59. C. Gao, Z. Wang, Y. Guo, H. Wang, and H. Yi. "MPINet: Multiscale physics-informed network for bearing fault diagnosis with small samples". *IEEE transactions on industrial informatics* 20, 12 2024, pp. 14371–14380. ISSN: 1551-3203,1941-0050.
60. D. Garcia-Alvarez, A. Bregon, B. Pulido, and C. J. Alonso-Gonzalez. "Integrating PCA and structural model decomposition to improve fault monitoring and diagnosis with varying operation points". *Engineering applications of artificial intelligence* 122, 106145 2023, p. 106145. ISSN: 0952-1976,1873-6769.
61. A. Garg, W. Zhang, J. Samaran, R. Savitha, and C.-S. Foo. "An evaluation of anomaly detection and diagnosis in multivariate time series". *IEEE transactions on neural networks and learning systems* 33, 6 2022, pp. 2508–2517. ISSN: 2162-2388,2162-237X.
62. J. George, C. Gao, R. Liu, H. G. Liu, Y. Tang, R. Pydipaty, and A. K. Saha. *A Scalable and Cloud-Native Hyperparameter Tuning System*. 2020. arXiv: 2006.02085 [cs.DC].
63. M. S. Gill, T. Westermann, M. Schieseck, and A. Fay. "Integration of Domain Expert-Centric Ontology Design into the CRISP-DM for Cyber-Physical Production Systems". In: *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2023, pp. 1–8.
64. J. Goh, S. Adepu, K. N. Junejo, and A. Mathur. "A dataset to support research in the design of secure water treatment systems". *Critical Information Infrastructures Security*, 10, 2016, pp. 88–99.
65. A. Golestani and R. Gras. "Can we predict the unpredictable?" *Scientific reports* 4, 1 30, 2014, p. 6834. ISSN: 2045-2322.
66. D. Golubovic and R. Rocha. "Training and Serving ML workloads with Kube-flow at CERN". In: *EPJ Web of Conferences*. Vol. 251. 2021, p. 02067.
67. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger. Vol. 27. Orig GAN Paper. Curran Associates, Inc., 2014.
68. A. Goyal, A. Lamb, J. Hoffmann, S. Sodhani, S. Levine, Y. Bengio, and B. Schölkopf. "Recurrent Independent Mechanisms". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.
69. T. Granlund, V. Stirbu, and T. Mikkonen. "Towards regulatory-compliant MLOps: Oravio's journey from a machine learning experiment to a deployed certified medical product". *SN computer Science* 2, 5 2021, p. 342.
70. S. Grednev, H. S. Steude, S. Bronder, O. Niggemann, and A. Jung. "AI-assisted study of auxetic structures". *Acta Polytechnica CTU Proceedings* 42, 2023, pp. 32–36.

71. F. E. Grubbs. "Procedures for detecting outlying observations in samples". *Technometrics: a journal of statistics for the physical, chemical, and engineering sciences* 11, 1 1969, pp. 1–21. ISSN: 0040-1706,1537-2723.
72. D. Grünbaum, M. L. Stern, and E. W. Lang. "Quantitative probing: Validating causal models with quantitative domain knowledge". *Journal of Causal Inference* 11, 1 2023, p. 20220060.
73. A. Gu and T. Dao. "Mamba: Linear-Time Sequence Modeling with Selective State Spaces". In: *First Conference on Language Modeling*. 2024.
74. D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, and X. Bi. "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning". *arXiv preprint arXiv:2501.12948*, 2025.
75. D. M. Hawkins. *Identification of outliers*. Vol. 11. Springer, 1980.
76. K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Las Vegas, NV, USA). IEEE, 2016. ISBN: 9781467388511.
77. Z. He, Y. Chen, D. Zhang, and M. Abdulaal. "Vehicle anomaly detection by attention-enhanced temporal convolutional network". In: *2023 IEEE 6th International Conference on Industrial Cyber-Physical Systems (ICPS)*. 2023, pp. 1–6.
78. I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. "beta-vae: Learning basic visual concepts with a constrained variational framework". In: *ICLR*. 5th International Conference on Learning Representations. 2017.
79. K. Hightower, B. Burns, and J. Beda. *Kubernetes: Up and Running Dive into the Future of Infrastructure*. 1st. O'Reilly Media, Inc., 2017. ISBN: 9781491935675.
80. G. E. Hinton and R. R. Salakhutdinov. "Reducing the dimensionality of data with neural networks". *Science* 313, 5786 28, 2006, pp. 504–507. ISSN: 0036-8075,1095-9203.
81. S. Hochreiter and J. Schmidhuber. "Long short-term memory". *Neural computation* 9, 8 1997, pp. 1735–1780.
82. X. Hou, Y. Zhao, S. Wang, and H. Wang. "Model context protocol (mcp): Landscape, security threats, and future research directions". *arXiv preprint arXiv:2503.23278*, 2025.
83. N. Hranisavljevic, A. Maier, and O. Niggemann. "Discretization of hybrid CPPS data into timed automaton using restricted Boltzmann machines". *Engineering applications of artificial intelligence* 95, 1, 2020, p. 103826. ISSN: 0952-1976.

Bibliography

84. N. Hranisavljevic, T. Westermann, S. Plambeck, H. S. Steude, G. Benndorf, and O. Niggemann. "A Model Learning Perspective on the Complexity of Cyber-Physical Systems". In: *ML4CPS – Machine Learning for Cyber-Physical Systems* (Berlin). UB HSU, 2025.
85. E. Jang, S. Gu, and B. Poole. "Categorical Reparameterization with Gumbel-Softmax". *arXiv [stat.ML]*, 3, 2016. One of the two papers introducing categorical latent variables and the reparameterization trick. arXiv: 1611.01144 [stat.ML].
86. N. Jeffrey, Q. Tan, and J. R. Villar. "A review of anomaly detection strategies to detect threats to Cyber-Physical Systems". *Electronics* 12, 15 30, 2023, p. 3283. ISSN: 2079-9292,2079-9292.
87. Y. Jeong, E. Yang, J.H. Ryu, I. Park, and M. Kang. "Anomalybert: Self-supervised transformer for time series anomaly detection using data degradation scheme". *arXiv preprint arXiv:2305.04468*, 2023.
88. D. Jung. "Automated Design of Grey-Box Recurrent Neural Networks For Fault Diagnosis using Structural Models and Causal Information". In: *Learning for Dynamics and Control Conference*. 2022, pp. 8–20.
89. D. Jung, B. Kleman, H. Lindgren, and H. Warnquist. "Fault Diagnosis of Exhaust Gas Treatment System Combining Physical Insights and Neural Networks". *IFAC-PapersOnLine* 55, 24 2022. 10th IFAC Symposium on Advances in Automotive Control AAC 2022, pp. 97–102. ISSN: 2405-8963.
90. D. Jung, K. Y. Ng, E. Frisk, and M. Krysander. "Combining model-based diagnosis and data-driven anomaly classifiers for fault isolation". *Control engineering practice* 80, 2018, pp. 146–156. ISSN: 0967-0661,1873-6939.
91. H. Kang and P. Kang. "Transformer-based multivariate time series anomaly detection using inter-variable attention mechanism". *Knowledge-Based Systems* 290, 2024, p. 111507. ISSN: 0950-7051.
92. D. P. Kingma and M. Welling. "An Introduction to Variational Autoencoders". *arXiv [cs.LG]*, 6, 2019. arXiv: 1906.02691 [cs.LG].
93. D. P. Kingma and M. Welling. "Auto-Encoding Variational Bayes". *arXiv [stat.ML]*, 20, 2013. First paper on variational auto encoders!! arXiv: 1312.6114v10 [stat.ML].
94. J. de Kleer. "Dynamic domain abstraction through meta-diagnosis". In: *Lecture Notes in Computer Science*. Lecture notes in computer science. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 109–123. ISBN: 9783540735793,9783540735809.
95. G. Koutroulis, B. Mutlu, and R. Kern. "A causality-inspired approach for anomaly detection in a water treatment testbed". *Sensors* 23, 1 2022, p. 257.

96. M. Kravchik and A. Shabtai. "Detecting cyber attacks in industrial control systems using convolutional neural networks". In: *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and Privacy*. CCS '18: 2018 ACM SIGSAC Conference on Computer and Communications Security (Toronto Canada). ACM, New York, NY, USA, 15, 2018. ISBN: 9781450359924.
97. D. Kreuzberger, N. Kühl, and S. Hirschl. "Machine Learning Operations (MLOps): Overview, Definition, and Architecture". *IEEE Access* 11, 2023, pp. 31866–31879. ISSN: 2169-3536.
98. G. Lamperti and X. Zhao. "Diagnosis of higher-order discrete-event systems". In: *Availability, Reliability, and Security in Information Systems and HCI: IFIP WG 8.4, 8.9, TC 5 International Cross-Domain Conference, CD-ARES 2013, Regensburg, Germany, September 2-6, 2013. Proceedings* 8. 2013, pp. 162–177.
99. C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager. "Temporal convolutional networks for action segmentation and detection". In: *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 156–165.
100. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*. Vol. 86. 1998, pp. 2278–2324.
101. P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-T. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474.
102. D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S.-K. Ng. "MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks". In: *Artificial Neural Networks and Machine Learning – ICANN 2019: Text and Time Series*. Ed. by I. V. Tetko, V. Kůrková, P. Karpov, and F. Theis. Springer International Publishing, Cham, 2019, pp. 703–716. ISBN: 9783030304904.
103. S. Li, H. Zhang, Z. Jia, C. Zhong, C. Zhang, Z. Shan, J. Shen, and M. A. Babar. "Understanding and addressing quality attributes of microservices architecture: A Systematic literature review". *Information and Software Technology* 131, 2021, p. 106449. ISSN: 0950-5849.
104. S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala. "PyTorch distributed". *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases* 13, 12 2020, pp. 3005–3018. ISSN: 2150-8097.
105. Z. Li, Y. Zhao, J. Han, Y. Su, R. Jiao, X. Wen, and D. Pei. "Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding". In: *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 2021, pp. 3220–3230.

Bibliography

106. Y. Liang, H. Wen, Y. Nie, Y. Jiang, M. Jin, D. Song, S. Pan, and Q. Wen. "Foundation Models for Time Series Analysis: A Tutorial and Survey". In: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain). KDD '24. Association for Computing Machinery, New York, NY, USA, 2024, pp. 6555–6565.
107. S. Lin, R. Clark, R. Birke, S. Schonborn, N. Trigoni, and S. Roberts. "Anomaly detection for time series using VAE-LSTM hybrid model". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (Barcelona, Spain). IEEE, 2020, pp. 4322–4326. ISBN: 9781509066315.
108. C. Liu and J. Han. "Failure proximity: a fault localization-based approach". In: *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*. 2006, pp. 46–56.
109. J. Lucas, G. Tucker, R. B. Grosse, and M. Norouzi. "Don't blame the elbow! a linear vae perspective on posterior collapse". *Advances in Neural Information Processing Systems* 32, 2019.
110. A. Lundgren and D. Jung. "Data-driven fault diagnosis analysis and open-set classification of time-series data". *Control Engineering Practice* 121, 2022, p. 105006.
111. W. Ma, H. Wen, W. Hou, R. Yao, Y. Zhu, D. Yan, and Z. Zhao. "False data injection attacks detection in smart grids based on TCN prediction model". In: *International Conference on Optical Communication and Optoelectronic Technology (OCOT 2024)*. Vol. 13289. 2024, pp. 230–236.
112. L. van der Maaten and G. Hinton. "Visualizing Data using t-SNE". *Journal of machine learning research: JMLR* 9, 86 2008, pp. 2579–2605. ISSN: 1532-4435,1533-7928.
113. C. J. Maddison, A. Mnih, and Y. W. Teh. "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables". *arXiv [cs.LG]*, 2, 2016. One of the two papers introducing categorical latent variables and the reparameterization trick. arXiv: 1611.00712 [cs.LG].
114. P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. "Long short term memory networks for anomaly detection in time series". In: *ESANN 2015 proceedings, European symposium on artificial neural networks, computational intelligence and machine learning. Bruges (Belgium), 22-24 April 2015*, vol. 89. 2015, p. 94.
115. F. Martinez-Plumed, L. Contreras-Ochando, C. Ferri, J. Hernandez-Orallo, M. Kull, N. Lachiche, M. J. Ramirez-Quintana, and P. Flach. "CRISP-DM twenty years later: From data mining processes to data science trajectories". *IEEE transactions on knowledge and data engineering* 33, 8 1, 2021, pp. 3048–3061. ISSN: 1041-4347,1558-2191.

116. C. Meng, X.S. Jiang, X.M. Wei, and T. Wei. "A Time Convolutional Network Based Outlier Detection for Multidimensional Time Series in Cyber-Physical-Social Systems". *IEEE Access* 8, 2020, pp. 74933–74942. ISSN: 2169-3536.
117. S. Merkelbach, A. Diedrich, A. Szttyber-Betley, L. Travé-Massuyès, E. Chanthery, O. Niggemann, and R. Dumitrescu. "Using Multi-Modal LLMs to Create Models for Fault Diagnosis (Short Paper)". In: *DX*. 2024, 31:1–31:15.
118. L. Mescheder, Geiger, and S. Nowozin. *Which training methods GANs do actually converge? International Conference Machine Learning (ICML)*. PMLR, 2018, pp. 3481–3490.
119. M. Mesnier, G. R. Ganger, and E. Riedel. "Object-based storage". *IEEE Communications Magazine* 41, 8 2003, pp. 84–90.
120. A. Mezina, R. Burget, and C. M. Travieso-Gonzalez. "Network anomaly detection with temporal convolutional network and U-net model". *IEEE access: practical innovations, open solutions* 9, 2021, pp. 143608–143622. ISSN: 2169-3536.
121. Z. Mian, X. Deng, X. Dong, Y. Tian, T. Cao, K. Chen, and T. A. Jaber. "A literature review of fault diagnosis based on ensemble learning". *Engineering Applications of Artificial Intelligence* 127, 2024, p. 107357. ISSN: 0952-1976.
122. H. Min, X. Lei, X. Wu, Y. Fang, S. Chen, W. Wang, and X. Zhao. "Toward interpretable anomaly detection for autonomous vehicles with denoising variational transformer". *Engineering Applications of Artificial Intelligence* 129, 2024, p. 107601. ISSN: 0952-1976.
123. G. S. Misyris, A. Venzke, and S. Chatzivasileiadis. "Physics-Informed Neural Networks for Power Systems". In: *2020 IEEE Power & Energy Society General Meeting (PESGM)*. 2, 2020, pp. 1–5.
124. L. Moddemann, H. S. Steude, A. Diedrich, and O. Niggemann. "Discret2di-deep learning based discretization for model-based diagnosis". *IFAC-PapersOnLine* 58, 4 2024, pp. 640–645.
125. L. Moddemann, H. S. Steude, A. Diedrich, I. Pill, and O. Niggemann. "Extracting Knowledge using Machine Learning for Anomaly Detection and Root-Cause Diagnosis". In: *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2024, pp. 1–8.
126. A. Mohammadi, M. Krysander, and D. E. Jung. "Analysis of grey-box neural network-based residuals for consistency-based fault diagnosis". *IFAC-PapersOnLine* 55, 6 2022, pp. 1–6. ISSN: 2405-8971,2405-8963.
127. Z. A. Muhammad Farooq Siddique and J.-M. Kim. "Pipeline leak diagnosis based on leak-augmented scalograms and deep learning". *Engineering Applications of Computational Fluid Mechanics* 17, 1 2023, p. 2225577. eprint: <https://doi.org/10.1080/19942060.2023.2225577>.

Bibliography

128. K. P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.
129. S. Neupane, I. A. Fernandez, W. Patterson, S. Mittal, and S. Rahimi. "A temporal anomaly detection system for vehicles utilizing functional working groups and sensor channels". In: *2022 IEEE 8th International Conference on Collaboration and Internet Computing (CIC)*. 2022 IEEE 8th International Conference on Collaboration and Internet Computing (CIC) (Atlanta, GA, USA). IEEE, 2022.
130. M. Newman. *Networks*. Oxford university press, 2018.
131. O. Niggemann, B. Zimmering, H. S. Steude, J. L. Augustin, A. Windmann, and S. Multaheb. "Machine learning for cyber-physical systems". In: *Digital Transformation*. Ed. by B. Vogel-Heuser and M. Wimmer. Springer Berlin Heidelberg, Berlin, Heidelberg, 2023, pp. 415–446. ISBN: 9783662650035,9783662650042.
132. H. Nizam, S. Zafar, Z. Lv, F. Wang, and X. Hu. "Real-Time Deep Anomaly Detection Framework for Multivariate Time-Series Data in Industrial IoT". *IEEE Sensors Journal* 22, 23 2022, pp. 22836–22849.
133. OpenAI. *Scaling Kubernetes to 7,500 Nodes*. 2021. URL: <https://openai.com/index/scaling-kubernetes-to-7500-nodes/> (visited on 04/16/2025).
134. G. Pang, C. Shen, L. Cao, and A. Van Den Hengel. "Deep learning for anomaly detection: A review". *ACM computing surveys* 54, 2 31, 2022, pp. 1–38. ISSN: 0360-0300,1557-7341.
135. T. Park, K. Song, J. Jeong, and H. Kim. "Convolutional autoencoder-based anomaly detection for photovoltaic power forecasting of virtual power plants". *Energies* 16, 14 11, 2023, p. 5293. ISSN: 1996-1073.
136. P. Parodi, Z. Szigetvari, R. Muller, and A. Quaglia. "Columbus ECLSS: five years of operations and lessons learned". In: *43rd International Conference on Environmental Systems*. 43rd International Conference on Environmental Systems (Vail, CO). American Institute of Aeronautics and Astronautics, Reston, Virginia, 14, 2013. ISBN: 9781624102158.
137. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems* 32. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035.
138. P. Patarasuk and X. Yuan. "Bandwidth optimal all-reduce algorithms for clusters of workstations". *Journal of parallel and distributed computing* 69, 2 2009, pp. 117–124. ISSN: 0743-7315,1096-0848.
139. 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* 2, 11 1901, pp. 559–572.

140. F. Perrot and L. Travé-Massuyes. "Choosing abstractions for hierarchical diagnosis". In: *18th International Workshop on Principles of Diagnosis*. 2007, p. 42.
141. E. Pesola, K. Kolcio, M. Prather, and A. Ildefonso. "A Hybrid Model-Based and Data-Driven Framework for Automated Spacecraft Fault Detection". In: *Annual Conference of the PHM Society*. Vol. 15. 2023.
142. I. Pill and J. de Kleer. *Challenges for model-based Diagnosis*. Ed. by I. Pill, A. Natan, and F. Wotawa. Dagstuhl, Germany, 26, 2024.
143. M. Pota, G. De Pietro, and M. Esposito. "Real-time anomaly detection on time series of industrial furnaces: A comparison of autoencoder architectures". *Engineering Applications of Artificial Intelligence* 124, 2023, p. 106597. ISSN: 0952-1976.
144. M. R. G. Raman and A. P. Mathur. "A Hybrid Physics-Based Data-Driven Framework for Anomaly Detection in Industrial Control Systems". *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52, 9 2022, pp. 6003–6014.
145. J. Rehak, A. Sommer, M. Becker, J. Pfrommer, and J. Beyerer. "Counterfactual Root Cause Analysis via Anomaly Detection and Causal Graphs". In: *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*. IEEE, 18, 2023, pp. 1–7. ISBN: 9781665493130,9781665493147.
146. R. Reiter. "A theory of diagnosis from first principles". *Artificial intelligence* 32, 1 27, 1986, pp. 57–95. ISSN: 0004-3702,1872-7921.
147. F. Rewicki, J. Denzler, and J. Niebling. "Is It Worth It? Comparing Six Deep and Classical Methods for Unsupervised Anomaly Detection in Time Series". *Applied Sciences* 13, 3 2023. ISSN: 2076-3417.
148. D. Rezende and S. Mohamed. "Variational inference with normalizing flows". In: *International conference on machine learning*. 2015, pp. 1530–1538.
149. M. Rocklin. "Dask: Parallel computation with blocked algorithms and task scheduling". In: *Proceedings of the 14th python in science conference*. Vol. 130. 2015, p. 136.
150. P. Ruf, M. Madan, C. Reich, and D. Ould-Abdeslam. "Demystifying MLOps and Presenting a Recipe for the Selection of Open-Source Tools". *Applied Sciences* 11, 19 2021, p. 8861.
151. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning Representations by Back-Propagating Errors". *Nature* 323, 6088 1986, pp. 533–536. ISSN: 1476-4687.
152. J. Runge, A. Gerhardus, G. Varando, V. Eyring, and G. Camps-Valls. "Causal inference for time series". *Nature Reviews Earth & Environment* 4, 7 2023, pp. 487–505.

Bibliography

153. M. A. Saez, F. P. Maturana, K. Barton, and D. M. Tilbury. "Context-Sensitive Modeling and Analysis of Cyber-Physical Manufacturing Systems for Anomaly Detection and Diagnosis". *IEEE Transactions on Automation Science and Engineering* 17, 1 2020, pp. 29–40.
154. M. Sakurada and T. Yairi. "Anomaly detection using autoencoders with nonlinear dimensionality reduction". In: *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. MLSDA'14: Machine Learning for Sensory Data Analysis (Gold Coast Australia QLD Australia). ACM, New York, NY, USA, 2, 2014. ISBN: 9781450331593.
155. M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. "Diagnosability of discrete-event systems". *IEEE Transactions on Automatic Control* 40, 9 1995, pp. 1555–1575.
156. S. Schmidl, P. Wenig, and T. Papenbrock. "Anomaly detection in time series: a comprehensive evaluation". *Proceedings of the VLDB Endowment* 15, 9 2022, pp. 1779–1797.
157. D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. "Hidden technical debt in machine learning systems". *Advances in neural information processing systems*, 2015. ISSN: 1049-5258.
158. K. Senjab, S. Abbas, N. Ahmed, and A. u. R. Khan. "A survey of Kubernetes scheduling algorithms". *Journal of Cloud Computing* 12, 1 2023, p. 87.
159. H. Shafiei, A. Khonsari, and P. Mousavi. "Serverless computing: a survey of opportunities, challenges, and applications". *ACM Computing Surveys* 54, 11s 2022, pp. 1–32.
160. W. Shang, J. Qiu, H. Shi, S. Wang, L. Ding, Y. Xiao, and Y.-A. Tan. "An Efficient Anomaly Detection Method for Industrial Control Systems: Deep Convolutional Autoencoding Transformer Network". *Int. J. Intell. Syst.* 2024, 2024. ISSN: 0884-8173.
161. C. Shearer. "The CRISP-DM Model: The New Blueprint for Data Mining". *Journal of Data Warehousing* 5, 4 2000.
162. B. Shneiderman. "Human-centered artificial intelligence: Reliable, safe & trustworthy". *International journal of human-computer interaction* 36, 6 2, 2020, pp. 495–504. ISSN: 1044-7318,1532-7590.
163. L. M. V. da Silva, A. Köcher, F. Gehlhoff, and A. Fay. "On the Use of Large Language Models to Generate Capability Ontologies". In: *Proceedings of the 2024 IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*. Padova, Italy, 2024.

164. E. N. Spotorno Bieger, L. P. Horstmann, and A. A. Fröhlich. “An Analysis of LSTMs and CNNs Robustness for Early Battery End of Life Prediction on Multivariate Time Series Based on Non-Stationarity and Entropy”. In: *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2024, pp. 01–08.
165. H. S. Steude, D. Vranješ, J. L. Augustin, O. Niggemann, M. Lange-Hegermann, and D. Höche. “Use of neural networks for automated analysis and modeling of elasto-plastic material properties of metallic tensile specimens”. *Automation 2022 (VDI Berichte) 2399*, 2022, pp. 505–520.
166. H. S. Steude and A. Demeschkin. “Method and computer program product for monitoring a bleed air supply system of an aircraft”. Pat. 2022189235. National Phase Entries: US 18549599 (2024-03-21), EP 2022708959 (2024-01-17), KR 1020237034489 (2023-11-07), CN 202280034242.5 (2023-12-26). 15, 2022.
167. H. S. Steude, A. Diedrich, I. Pill, L. Moddemann, D. Vranješ, and O. Niggemann. *Data Driven Diagnosis for Large Cyber-Physical-Systems with Minimal Prior Information*. 2025. arXiv: 2506.10613 [cs.AI].
168. H. S. Steude and C. Geier. “Efficient Integration of Kubeflow Tools: An ISS Data Case Study on Anomaly Detection”. In: *KubeCon + CloudNativeCon Europe 2024 Co-located Events: Kubeflow Summit* (Paris Expo Porte De Versailles). Peer-reviewed presentation at CNCF-hosted co-located event. Paris, France, 19, 2024.
169. H. S. Steude and C. Geier. *Scaling Python: An End-to-End ML Pipeline for ISS Anomaly Detection with Kubeflow*. Conference Talk. Darmstadt, Germany, 2025.
170. H. S. Steude, C. Geier, L. Moddemann, M. Creutzenberg, J. Pfeifer, S. Turk, and O. Niggemann. “End-to-end MLOps integration: a case study with ISS telemetry data”. In: *ML4CPS – Machine Learning for Cyber-Physical Systems* (Berlin, 2024). UB HSU, 2024.
171. H. S. Steude, L. Moddemann, A. Diedrich, J. Ehrhardt, M. Kalech, and O. Niggemann. “Diagnosis Driven Anomaly Detection for CPS”. In: *Proceedings of the 34th International Workshop on Principle of Diagnosis*. Loma Mar, USA, 2023.
172. H. S. Steude, L. Moddemann, A. Diedrich, J. Ehrhardt, and O. Niggemann. “Diagnosis driven Anomaly Detection for Cyber-Physical Systems”. *IFAC-PapersOnLine* 58, 4 2024. 12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2024, pp. 13–18. ISSN: 2405-8971,2405-8963.

Bibliography

173. H. S. Steude, A. Windmann, and O. Niggemann. "Learning physical concepts in CPS: A case study with a three-tank system". *IFAC-PapersOnLine* 55, 6 1, 2022. 11th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2022, pp. 15–22. ISSN: 2405-8963,2405-8971.
174. M. Stumptner and F. Wotawa. "Diagnosing tree-structured systems". *Artificial Intelligence* 127, 1 2001. Part of this work has been published in preliminary form in the Proceedings of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI-97)., pp. 1–29. ISSN: 0004-3702.
175. A. Subramaniam and A. Subramaniam. "Automated Resource Scaling in Kubeflow through Time Series Forecasting". In: *2023 IEEE 5th International Conference on Cybernetics, Cognition and Machine Learning Applications (ICC-CMLA)*. 2023, pp. 173–179.
176. X. Sun, D. Wu, A. Zinflou, and B. Boulet. "Power System Anomaly Detection Via Ensemble of Encoder and Decoder Networks". In: *2022 IEEE Electrical Power and Energy Conference (EPEC)*. 2022, pp. 116–122.
177. G. Symeonidis, E. Nerantzis, A. Kazakis, and G. A. Papakostas. "MLOps - Definitions, Tools and Challenges". In: *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC) (Las Vegas, NV, USA). IEEE, 26, 2022.
178. M. Tappe, L. Moddemann, H. S. Steude, N. Hranisavljevic, S. Myschik, C. Geier, M. Creutzenberg, P. Grashorn, H. Ernst, and O. Niggemann. "A Supervised AI-Based Toolchain for Anomaly Detection, Diagnosis, and Reconfiguration for the Life Support System of the Columbus Module of the ISS". *CEAS Aeronautical Journal*, 2025. Accepted, but final submission still due.
179. T. pandas development team. *pandas-dev/pandas: Pandas*. Comp. software. Version latest. 2020.
180. L. Torvalds and J. C. Hamano. *Git*. Version control system, accessed on April 21, 2025. 2025.
181. S. Tuli, G. Casale, and N. R. Jennings. "TranAD: Deep Transformer Networks for Anomaly Detection in Multivariate Time Series Data". *arXiv [cs.LG]*, 18, 2022. arXiv: 2201.07284 [cs.LG].
182. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". *Advances in neural information processing systems* 30, 2017.
183. D. Vranješ, J. Ehrhardt, R. Heesch, L. Moddemann, H. S. Steude, and O. Niggemann. "Design Principles for Falsifiable, Replicable and Reproducible Empirical Machine Learning Research". In: *35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024)*. 2024, pp. 7–1.

184. D. Vranješ and O. Niggemann. "Enhancing Cyber-Physical System Analysis with Structure-Aware Modular Neural Networks". In: *2024 IEEE 7th International Conference on Industrial Cyber-Physical Systems (ICPS)*. 2024, pp. 1–8.
185. C. Wang, S. Xing, R. Gao, L. Yan, N. Xiong, and R. Wang. "Disentangled dynamic deviation transformer networks for multivariate time series anomaly detection". *Sensors (Basel, Switzerland)* 23, 3 18, 2023, p. 1104. ISSN: 1424-8220.
186. Y. Wang, Y. Wu, Q. Yang, and J. Zhang. "Anomaly detection of spacecraft telemetry data using temporal convolution network". In: *2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. 2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) (Glasgow, United Kingdom). IEEE, 17, 2021. ISBN: 9781728195391.
187. Z. Wang, W. Yan, and T. Oates. "Time series classification from scratch with deep neural networks: A strong baseline". *2017 International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 1578–1585.
188. J. Weber and F. Wotawa. "Diagnosing Dependent Failures—an Extension of Consistency-based Diagnosis". In: *18th Int. Work. Principles of Diagnosis*. 2007, p. 399.
189. T. Westermann, M. S. Gill, and A. Fay. "Representing Timed Automata and Timing Anomalies of Cyber-Physical Production Systems in Knowledge Graphs". In: *IECON 2023- 49th Annual Conference of the IEEE Industrial Electronics Society*. 2023, pp. 1–7.
190. M. Wielgosz, M. Mertik, A. Skoczeń, and E. De Matteis. "The model of an anomaly detector for HiLumi LHC magnets based on Recurrent Neural Networks and adaptive quantization". *Engineering applications of artificial intelligence* 74, 2018, pp. 166–185. ISSN: 0952-1976,1873-6769.
191. M. Wielgosz, A. Skoczeń, and M. Mertik. "Recurrent Neural Networks for anomaly detection in the Post-Mortem time series of LHC superconducting magnets". *arXiv [physics.ins-det]*, 2, 2017. arXiv:1702.00833 [physics.ins-det].
192. A. Windmann, H. S. Steude, D. Boschmann, and O. Niggemann. "Assessing Robustness for Prognostic Models on Cyber-Physical Systems". In: *Proceedings of the IEEE Emerging Technology and Factory Automation (ETFA)*. Not yet submitted. 2025.
193. A. Windmann, H. S. Steude, and O. Niggemann. "Robustness and Generalization Performance of Deep Learning Models on Cyber-Physical Systems: A Comparative Study". In: *IJCAI 2023 Workshop of Artificial Intelligence for Time Series Analysis (AI4TS)*. 2023.
194. B. Wu, Q. Xu, Z. Yao, Y. Tu, and Y. Chen. "VAE-TCN hybrid model for KPI Anomaly Detection". In: *2022 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS)*. 2022, pp. 1–6.

Bibliography

195. H. Wu, T. Hu, Y. Liu, H. Zhou, J. Wang, and M. Long. "TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis". In: *The Eleventh International Conference on Learning Representations*. 2023.
196. W. Wu, C. Song, J. Zhao, and Z. Xu. "Physics-informed gated recurrent graph attention unit network for anomaly detection in industrial cyber-physical systems". *Information Sciences* 629, 2023, pp. 618–633. ISSN: 0020-0255.
197. W. Wu, A. Pang, and W. Yang. "Heterogeneous Sensor Fault Detection for Networked Systems Based on a Graph Transformer". *IEEE Sensors Journal* 24, 4 2024, pp. 5266–5278.
198. L. Xia, Y. Liang, P. Zheng, and X. Huang. "Residual-Hypergraph Convolution Network: A Model-Based and Data-Driven Integrated Approach for Fault Diagnosis in Complex Equipment". *IEEE Transactions on Instrumentation and Measurement* 72, 2023, pp. 1–11.
199. X. Xie, B. Wang, T. Wan, and W. Tang. "Multivariate abnormal detection for industrial control systems using 1D CNN and GRU". *IEEE access: practical innovations, open solutions* 8, 2020, pp. 88348–88359. ISSN: 2169-3536.
200. C. Xu, G. Lv, J. Du, L. Chen, Y. Huang, and W. Zhou. "Kubeflow-based Automatic Data Processing Service for Data Center of State Grid Scenario". In: *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. 2021, pp. 924–930.
201. W. Yang, K. Zhang, and S. C. H. Hoi. "Causality-Based Multivariate Time Series Anomaly Detection". *CoRR abs/2206.15033*, 2022.
202. 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* 22, 9 2012, pp. 1567–1581. ISSN: 0959-1524,1873-2771.
203. A. B. Yoo, M. A. Jette, and M. Grondona. "Slurm: Simple linux utility for resource management". In: *Workshop on job scheduling strategies for parallel processing*. 2003, pp. 44–60.
204. F. Yu, V. Koltun, and T. Funkhouser. "Dilated residual networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 472–480.
205. M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, and M. Parkhe. "Accelerating the machine learning lifecycle with MLflow". *IEEE Data Eng. Bull.* 41, 4 2018, pp. 39–45.
206. M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, and M. J. Franklin. "Apache spark: a unified engine for big data processing". *Communications of the ACM* 59, 11 2016, pp. 56–65. ISSN: 0001-0782.

207. W. Zaman, M. F. Siddique, S. Ullah, F. Saleem, and J.-M. Kim. "Hybrid Deep Learning Model for Fault Diagnosis in Centrifugal Pumps: A Comparative Study of VGG16, ResNet50, and Wavelet Coherence Analysis". *Machines* 12, 12 2024, p. 905.
208. Z. Zamanzadeh Darban, G. I. Webb, S. Pan, C. Aggarwal, and M. Salehi. "Deep learning for time series anomaly detection: A survey". *ACM computing surveys*, 30, 2024. ISSN: 0360-0300,1557-7341.
209. C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, and N. V. Chawla. "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 2019, pp. 1409–1416.
210. P. Zhang, Z. Liu, N. Zhou, and H. Liu. "Fault Diagnosis for Smart Grids over Knowledge Graph and Abductive Reasoning". In: *2024 4th Power System and Green Energy Conference (PSGEC)*. 2024, pp. 1003–1008.
211. H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting". *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 12 2021, pp. 11106–11115. ISSN: 2374-3468,2159-5399.

Curriculum Vitae

Der Lebenslauf wird aus Gründen des Datenschutzes in der elektronische Fassung der Arbeit nicht veröffentlicht.