National Technical University of Athens
School of Electrical and Computer Engineering
Artificial Intelligence and Learning Systems Laboratory
Data Science and Machine Learning

Development of a Complete Marine Data Science Pipeline: From Data Collection to Model Deployment

A thesis written in the Artificial Intelligence and Learning Systems Laboratory in partial fulfillment of the requirements for the completion of the Data Science and Machine Learning Postgraduate Studies Programme.

Written by
Spyros Rigas

Supervised by
Prof. S. Kollias

Co-Supervised by
Dr. P. Tzouveli

Athens, March 2023

National Technical University of Athens
School of Electrical and Computer Engineering
Artificial Intelligence and Learning Systems Laboratory
Data Science and Machine Learning

Development of a Complete Marine Data Science Pipeline: From Data Collection to Model Deployment

Written by
Spyros Rigas

Supervised by
Prof. S. Kollias

Co-Supervised by
Dr. P. Tzouveli

Examined and evaluated by the Advisory Committee on March 3rd, 2023.

(Signature)	*(Signature)*	*(Signature)*
. . . .	. . . .	. . . .
Prof. S. Kollias	Prof. G. Stamou	Asst. Prof. A. Voulodimos

Athens, March 2023

If I were to give a summary of the tendency of our times, I would say, Quantity. The multitude, the mass spirit, dominates everywhere, destroying quality. Quantity, instead of adding to life's comforts and peace, has merely increased man's burden.

E. Goldman, 111 years ago

Περιληψη

Η εφαρμογή τεχνικών της επιστήμης δεδομένων και της μηχανικής μάθησης στη ναυτιλία έχει αναδειχθεί ως ένας πολλά υποσχόμενος ερευνητικός τομέας με σημαντικό δυναμικό για τη βελτίωση της απόδοσης των πλοίων και τη διασφάλιση ασφαλών ναυτιλιακών επιχειρήσεων, λόγω της πληθώρας των δεδομένων που δημιουργούνται από διάφορους αισθητήρες και συστήματα. Παρά το δυναμικό αυτό, η ναυτιλία έχει καθυστερήσει να υιοθετήσει την ψηφιοποίηση και την εφαρμογή προηγμένων τεχνικών ανάλυσης. Η παρούσα εργασία αποτελεί ένα βήμα προς την κατεύθυνση της τεχνολογικής εξέλιξης και της αυτοματοποίησης στη ναυτιλία, παρουσιάζοντας μια προσέγγιση για την ανίχνευση ανωμαλιών στα συστήματα των πλοίων μέσω εφαρμογής της μηχανικής μάθησης.

Πιο συγκεκριμένα, εξετάζεται η εφαρμογή αλγορίθμων ανίχνευσης μονομεταβλητών και πολυμεταβλητών ανωμαλιών, με στόχο την παρακολούθηση των συστημάτων των πλοίων και την ανίχνευση πιθανών προβλημάτων πριν αυτά καταστούν κρίσιμα. Συγκεκριμένα, χρησιμοποιείται ένα μοντέλο συνελικτικού νευρωνικού δικτύου με Spectral Residuals (SR-CNN) για την ανίχνευση μονομεταβλητών ανωμαλιών, ενώ για την ανίχνευση πολυμεταβλητών ανωμαλιών χρησιμοποιείται μια προηγμένη έκδοση του μοντέλου ανίχνευσης πολυμεταβλητών ανωμαλιών σε χρονοσειρές μέσω νευρωνικών δικτύων γράφων με attention (MTAD-GAT) της Microsoft.

Στο πλαίσιο της μελέτης αναπτύσσονται τρία διαφορετικά είδη συνόλων δεδομένων για την εκπαίδευση και την αξιολόγηση της απόδοσης των μοντέλων: ένας τύπος συνόλων δεδομένων με τεχνητά εισαγόμενες ανωμαλίες που χρησιμοποιούνται για την προ-εκπαίδευση και την αξιολόγηση των μοντέλων πριν την έκδοσή τους (deployment), ένας τύπος συνόλων συνθετικών δεδομένων χρονοσειρών τα οποία χρησιμοποιούνται για την αξιολόγηση της εκφραστικότητας των μοντέλων, χρησιμοποιώντας είδη ανωμαλιών με τα οποία τα μοντέλα δεν έχουν έρθει σε επαφή, και ένας τύπος συνόλων δεδομένων που περιέχουν τα δεδομένα που εκπέμπονται από τους εν λειτουργία αισθητήρες των πλοίων.

Στην εργασία αναλύεται επίσης η διαδικασία data engineering που ακολουθείται για την κατασκευή ενός πλήρους pipeline, το οποίο περιλαμβάνει την εισαγωγή δεδομένων, όλους τους απαραίτητους υπολογισμούς, την αποθήκευση των δεδομένων και την έκδοση των μοντέλων. Αυτό επιτυγχάνεται αξιοποιώντας τις προηγμένες δυνατότητες των υπολογισμών σε υπολογιστικό νέφος (cloud computing) και ειδικότερα τις υπηρεσίες που παρέχονται από το οικοσύστημα υπολογιστικού νέφους Azure της Microsoft.

Τα αποτελέσματα που προκύπτουν από την εκπαίδευση και αξιολόγηση των μοντέλων στο σύνολο δεδομένων με τις τεχνητά εισαγόμενες ανωμαλίες δείχνουν πόσο πολλά υποσχόμενη είναι η προτεινόμενη προσέγγιση στην ανίχνευση ανωμαλιών σε διαφορετικούς τύπους δεδομένων και συνθηκών. Αξιοσημείωτο είναι το γεγονός πως σε δύο περιπτώσεις τα μοντέλα καταφέρνουν να ανιχνεύσουν ανώμαλη συμπεριφορά σε δύο συστήματα πλοίων χρησιμοποιώντας την πραγματική ροή από τους αισθητήρες τους: κατόπιν επιθεωρήσεων από μηχανικούς που ακολούθησαν μερικές ημέρες μετά τις προβλέψεις των μοντέλων, αυτές συσχετίστηκαν με παρατηρήσιμες βλάβες στα αντίστοιχα συστήματα των πλοίων. Η εργασία αυτή παρέχει αξιόλογες πληροφορίες για τη δυνατότητα της μηχανικής μάθησης να επανασχεδιάσει τον τρόπο με τον οποίο πραγματοποιούνται οι ναυτιλιακές επιχειρήσεις και προσφέρει κατευθύνσεις για μελλοντικές έρευνες στον τομέα.

Λέξεις κλειδια

ναυτιλιακές επιχειρήσεις, δεδομένα χρονοσειρών, ανίχνευση ανωμαλιών, SR-CNN, νευρωνικά δίκτυα γράφων με attention, μεταβαλλόμενοι αυτοκωδικοποιητές, κατασκευή συνόλου δεδομένων, υπολογισμοί στο cloud, data engineering

Abstract

The application of data science and machine learning techniques to the maritime industry has emerged as a promising field with significant potential for improving vessel performance and ensuring safe marine operations, due to the abundance of data generated by various sensors and systems. Despite this potential, the industry has been slow to embrace digitalization and the application of advanced analytics. This thesis represents a step towards the direction of automation in the maritime industry, presenting an approach for detecting anomalies in shipboard systems through the application of machine learning.

More specifically, the deployment of univariate and multivariate anomaly detection algorithms is studied, with the aim of monitoring the health of vessel systems and identifying potential issues before they become critical. The Spectral Residual Convolutional Neural Network model is used for univariate anomaly detection, while an advanced version of Microsoft's Multivariate Time-Series Anomaly Detection via Graph Attention Network model is used for multivariate anomaly detection.

As part of the study, three separate dataset types are developed to train and evaluate the performance of the anomaly detection models: a family of datasets with artificially induced anomalies that is used for the pre-training and evaluation of the models before deployment, a family of synthetic time-series datasets which is utilized to evaluate the expressivity of the trained models using unseen types of anomalies and a family of datasets corresponding to the operational sensor data feed.

The work also includes a comprehensive analysis of the data engineering process followed for the construction of a complete pipeline, which involves the ingestion of data, all necessary computations, data storing and models serving. This is achieved by leveraging the advancements in cloud computing and in particular the features and services provided by Microsoft's Azure Cloud Ecosystem.

The results achieved from training and evaluating the models in the artificially induced anomalies dataset demonstrate how promising the proposed approach in detecting anomalies in different types of data and settings is. Importantly, two instances of the deployed models are able to detect anomalous behaviour in two vessel systems using their sensors' live feed and consequent inspections by engineers on-board validate the models' predictions. This thesis provides valuable insights into the potential for machine learning to revolutionize the way marine operations are conducted, and offers a roadmap for future research in the field.

Keywords

`marine operations, time-series data, anomaly detection, SR-CNN, graph attention networks, variational autoencoders, dataset construction, cloud computing, data engineering`

Acknowledgements

The completion of the present thesis would have been nothing but fiction, were it not for some people to whom I owe my heartfelt gratitude.

First and foremost, I would like to thank my main supervisor, Prof. S. Kollias, for trusting me with the thesis' topic and agreeing to supervise it. My thesis would not have been possible without the guidance and mentorship of my Co-supervisor, Dr. P. Tzouveli, whose insights and encouragement have been invaluable. I would also like to thank the members of my thesis committee, Prof. G. Stamou and Asst. Prof. A. Voulodimos, who generously shared their time to read and review this work.

I am indebted to my friends and family for their unwavering support and especially my partner, Pinelopi. I am also thankful to Prof. P. Mavropoulos; although our relationship began as a purely academic one during my earlier Physics studies, Phivos has become a dear friend of mine, who has helped and guided me numerous times along the road. This list could not leave out the wonderful friends I made during my studies in the DSML programme, to whom I am grateful for their companionship throughout this journey.

Finally, I am truly indebted to SeaQuest Marine Technologies for trusting me to conceptualize and organize the project presented here and for providing me with the resources required for its realization. In addition, I owe a special thanks to Alex, my colleague and friend, who provided insightful suggestions about color combinations and other aspects related to the figures, which were tailor-made for the purposes of the present thesis. Last but not least, I would like to thank "Captain Nick" for his constant support, guidance and friendship.

Contents

1 Introduction

The maritime sector is a crucial player in global trade and commerce, with ships being the backbone of the transportation of goods and passengers across oceans on a daily basis. Evidently, ensuring the smooth operation of these vessels is paramount to the efficiency and reliability of the industry. With the growth by leaps and bounds that the fields of Internet of Things (IoT), Big Data and Machine Learning have seen in the past few years [1,2], there is enormous potential to optimize the performance of vessels and improve safety, thus fundamentally transforming the way marine operations are conducted. Despite these advances in technology, the maritime sector has been slow to embrace digitalization [3], missing out on opportunities to improve efficiency and reduce operational costs.

A modern ship is a highly complex system, comprising hundreds of systems and sub-systems located in different locations and serving a variety of purposes. As such, one of the most promising ways to harness the power of technology is to automate the processes that monitor the health condition of these systems and alert engineers and stakeholders whenever certain actions need to be taken for their maintenance or possible repairs. In the past, this task was performed through periodic inspections and engineering checks, a highly non-scalable practice that could only be parallelized by increasing the number of engineers on board [4]. To address this limitation, some vessels began incorporating sensors to automate the monitoring process and established rule-based alerting systems [5,6] based on sensor readings. For instance, by defining a certain width threshold, a sensor monitoring a steel pipe's width is able to transmit an alert if its readings drop below said threshold. Of course, this rule-based approach has its own limitations, including the inability to analyze multiple signals and make decisions based on combined component behavior. Additionally, this approach is very naive and requires a prior understanding of the system's behaviour in each specific environment (the rules cannot be set arbitrarily), which is almost never the case. Finally, data from normal operating conditions may be continuously collected, however data from known faults are substantially rarer, especially if they are labeled.

A more advanced way to automate this process of monitoring and alerting is through the implementation of anomaly detection machine learning algorithms [7]. These algorithms are designed to identify deviations from a learnable normal behavior, which could indicate potential problems before they cause costly disruptions in the operation of the vessel. A major advantage of this approach is that it can be done through the analysis of single sensors, as well as by monitoring the health of multiple sensors that may be located on a single system. Additionally, no prior knowledge of the systems is required, as long as there are available data, and the data vectors themselves do not necessarily need to be labeled. Of course, there are still challenges that need to be addressed, such as the handling of noisy and incomplete data, or ensuring the privacy and security of collected data. The application of anomaly detection in the marine industry is still in its early stages. Nevertheless, there is increasing interest in using machine learning to analyze data from ships and identify potential problems before they occur[8–14]. The digitalization of the shipping industry and the application of machine learning in marine operations holds great promise, and some may argue that it is long overdue.

1.1 Thesis Scope & Structure

The present thesis investigates the application of machine learning in the field of marine operations. The focus is on the application of algorithms and machine learning models to detect anomalies in data collected from sensors installed on vessels, which provide various parameters related to the health of systems, such as temperature and vibration readings. The objective of the study is to streamline the process of anomaly detection, making it faster and more accurate through the use of machine learning models. The study aims to demonstrate the feasibility of using machine learning in a real-world scenario, where large amounts of data are generated continuously at large scales. The scope of the thesis includes the design and implementation of a complete pipeline that ingests, pre-processes and stores the data collected from the sensors, as well as trains and deploys the machine learning algorithms. This pipeline is based on cloud services, which allow for scalability and efficient processing of the data. The study also includes a thorough evaluation of the performance of the machine learning models and provides insights into the challenges and opportunities in this field.

The structure of the thesis is organized in a clear and concise manner to provide a comprehensive understanding of the subject. Chapter 2 lays the foundation by introducing the theoretical framework of anomaly detection, covering both univariate and multivariate approaches. It provides a detailed examination of the machine learning models and algorithms used throughout this work, including a formal description of each method. Chapter 3 focuses on the data used in the experiments. It provides an overview of the sensors used to collect the data, the data collection process, as well as the resulting datasets used to train and evaluate the machine learning models. Chapter 4 covers the technical aspects of the machine learning operations and data engineering. It describes the cloud services and pipelines used for all the involved stages, from data ingestion and pre-processing to the deployment of the trained models. Chapter 5 delves into the implementation and training of the machine learning models. It presents the experiments performed with the trained models, and provides a detailed discussion of the results. This chapter is critical in understanding the models' performance and the validity of the applied methods. Finally, Chapter 6 provides a conclusion to the study, summarizing the results and their implications. It also includes an outlook for future work, proposing potential avenues for further research and development.

1.2 Related Works

The use of anomaly detection algorithms in monitoring systems on vessels is an evolving field. Despite the growing interest in this area, a large volume of related work has been done by companies operating in the maritime sector, rather than academic institutions. As a result, much of the research and development being conducted in this field is proprietary, and not readily available to the public. For this reason, there are limited publicly available publications on the subject of monitoring systems on vessels and alerting through anomaly detection algorithms in real-world scenarios.

Among the publicly available ones, some studies have demonstrated the effectiveness of anomaly detection in the maritime industry by utilizing a series of different algorithms. Ellefsen et al.[10] train a Variational Autoencoder on carefully pre-processed data in order to classify velocity and acceleration measurements from autonomous ferries as anomalous or regular. Brandsæter et al.[8] employ Auto Associative Kernel Regression (AAKR)[15] for signal reconstruction and the sequential probability ratio test[16] for anomaly detection on data from sensors that cover different aspects of vessel operations. The same authors also develop a cluster based version of the same algorithms[9] to perform anomaly detection on diesel engine-related data, such as engine speed and power or bearing temperature. Last but not least, Kim et al.[11] propose an ensemble-based method for anomaly detection, including a series of different algorithms, such as Local Outlier Factor (LOF)[17] or Locally Selective Combination in Parallel Outlier Ensembles (LSCP)[18].

Other studies do not focus on the machine learning models as their primary research point, but rather on the different types of shipboard systems they can monitor and detect anomalies on, such as engines or generators. Qu et al.[13] focus on diesel engines, which are the main driving force for various civil ships, and perform anomaly detection on vibration data using deep learning models such as Recurrent Neural Networks or Autoencoders. Velasco-Gallego and Lazakis[14] perform anomaly detection on data composed of 14 features and acquired from sensors on diesel generators, using similar machine learning models.

Finally, very recently researchers turned their eye to explainable anomaly detection in maritime environments. Acknowledging the existence of several different anomaly detection algorithms, most of which are unsupervised due to the limited volume of available labeled data, they try to combine anomaly detection with explainable AI in order to be able to explain why models classify specific data instances as anomalies and measure the influence of individual features to the classification. For instance, Kim et al.[12] adopt the Shapley Additive exPlanations[19] (SHAP) framework and combine it with clustering methods to identify anomalies and specify which sensor is mainly responsible for them in each case.

It is worth mentioning that a considerable volume of studies have been performed on similar topics or fields and can be readily applied to the maritime sector. For instance, research has been done on the use of machine learning algorithms for predictive maintenance and anomaly detection in other industrial sectors, such as aviation[20], automotive[21] and power generation[22]. These studies provide valuable insights into the potential benefits and challenges of implementing these technologies in a complex, real-world environment.

2 Theoretical Framework & Algorithms

As explained in the introductory part of the present thesis, through continuous monitoring and analysis it is possible to identify unexpected events or rare occurrences in data that are obtained from sensors installed on various systems on vessels. These may indicate decreased efficiency or lead to equipment failures, resulting in significant financial losses for stakeholders. The early detection of these "anomalies" enables prompt action to be taken in order to mitigate or completely prevent the occurrence of issues. For example, the detection of an abnormal increase in a motor's temperature at an early stage allows an on-site engineer to activate redundancy measures, thereby preventing further overheating and potential equipment failure.

This process of identifying patterns or behaviors in data that deviate from the norm is known as anomaly detection (or outlier detection). The earliest forms of anomaly detection were based on simple statistical methods, such as the use of moving averages and standard deviation-based tests[23–25]. However, with the growth of big data, more sophisticated methods for anomaly detection were developed in order to allow for the analysis of large and complex datasets[7].

One of the main challenges with the task of anomaly detection is that anomalies are by definition rare events, and it is often difficult to find a sufficient number of labeled examples in order to perform it in a supervised manner. Furthermore, the definition of what constitutes an anomaly can vary depending on the application or the context, making it difficult to create a consistent set of labeled examples. For this reason, especially in industry applications where a single business scenario might require millions of labeled time-series data, unsupervised approaches are usually preferred. Nonetheless, when the availability of data allows anomaly detection algorithms to be trained in a supervised manner, their results tend to be more accurate.

This chapter presents a thorough analysis of the algorithms utilized for anomaly detection on the datasets that are presented in the following chapter. These algorithms are classified into two main categories, based on the number of features they analyze: univariate anomaly detection (UVAD) algorithms, which are used to identify anomalies in the time-series of a single feature, and multivariate anomaly detection (MVAD) algorithms, which are used to detect anomalies in the time-series of multiple features. While the UVAD algorithm of choice is unsupervised by default, the MVAD algorithm can be applied in both an unsupervised and supervised setting.

2.1 Univariate Anomaly Detection

The class of UVAD algorithms consists of models that attempt to identify anomalies in a single time-series. The problem can be formally stated as follows: given an input vector $\mathbf{x} = [x_1, x_2, \dots, x_T] \in \mathbb{R}^T$, a UVAD algorithm produces an output vector $\mathbf{y} = [y_1, y_2, \dots, y_T]$, where $y_i \in \{0, 1\}$, indicating whether the value x_i constitutes an anomaly ($y_i = 1$) or not ($y_i = 0$).

In the context of marine operations, UVAD is applied to the readings of each individual sensor. As such, a network of sensors would in general require a network of UVAD models - one for each sensor. The UVAD algorithm used for the purposes of the present thesis is known as the Spectral Residual Convolutional Neural Network (SR-CNN)[26].

2.1.1 Spectral Residuals & Saliency Maps

Saliency maps are frequently used in the domain of computer vision[27,28] and can be obtained by calculating the spectral residuals (SR)[29] of images. Images are equivalent to time-series, in the sense that an image is an ordered sequence of pixels, while a time-series is an ordered sequence of values at specific timestamps. Therefore, methods that have proved to be efficient in one domain can usually be used in the other, mutatis mutandis. The main idea of the SR UVAD is that the tasks of UVAD and visual saliency detection share a lot of common elements, since the anomaly points are usually salient in the visual perspective.

Given an input time-series $\mathbf{x}$, the first step in acquiring the corresponding saliency map is the calculation of its Discrete Fourier Transform (DFT) using the Fast Fourier Transform (FFT) algorithm. Supposing that $\mathcal{T}$ is the DFT operator and $\boldsymbol{\omega}$ is the frequencies vector, the Fourier spectrum's amplitude and phase spectra can be calculated as

$$A(\boldsymbol{\omega}) = \mathrm{Re}\left[\mathcal{T}(\mathbf{x})\right] \quad \text{and} \quad P(\boldsymbol{\omega}) = \mathrm{Im}\left[\mathcal{T}(\mathbf{x})\right], \tag{2.1}$$

respectively. Next, the log-representation of the amplitude spectrum is given by

$$\mathcal{L}(\boldsymbol{\omega}) = \log\left[A(\boldsymbol{\omega})\right], \tag{2.2}$$

where amplitudes that are equal to zero are ignored (for example, set equal to zero). Using the log-representation, the averaged spectrum, $\mathcal{A}(\boldsymbol{\omega})$, can be approximated(1) as

$$\mathcal{A}(\boldsymbol{\omega}) = \mathbf{h}_q \circledast \mathcal{L}(\boldsymbol{\omega}), \tag{2.3}$$

i.e. the convolution of $\mathcal{L}(\boldsymbol{\omega})$ by a filter $\mathbf{h}_q$, where

$$\mathbf{h}_q = \frac{1}{q}\left[1, 1, 1, \dots, 1\right] \in \mathbb{R}^q. \tag{2.4}$$

(1) Strictly speaking, Eq. (2.3) is only an approximation. However, the averaged curve usually indicates a local linearity and therefore adopting a local average filter to approximate its shape is more than reasonable.

The filter $\mathbf{h}_q$ is represented as a $q \times 1$ vector in Eq. (2.4), since $\mathcal{L}(\boldsymbol{\omega})$ is the log-representation of the amplitude spectrum of a sequence vector (time-series). In the equivalent computer vision problem, where the sequence would correspond to an image, the filter would be represented as a $q \times q$ matrix. The parameter q corresponds to one of the model's hyper-parameters.

The spectral residual, $\mathcal{R}(\boldsymbol{\omega})$, of the spectrum is obtained by subtracting the averaged spectrum from the log-representation:

$$\mathcal{R}(\boldsymbol{\omega}) = \mathcal{L}(\boldsymbol{\omega}) - \mathcal{A}(\boldsymbol{\omega}) . \tag{2.5}$$

In the context of image data, the spectral residual is supposed to contain the "innovation" of an image, in the sense that all redundant visual information has been removed (or at least significantly reduced) by the subtraction of Eq. (2.5). For this reason, in terms of time-series data, the spectral residual contains the "novelties" of the time-series expressed as discrete values in the frequency domain. The saliency map is then simply the corresponding information as expressed in the spatial domain:

$$S(\mathbf{x}) = \left\| \mathcal{F}^{-1} \left\{ \exp\left[\mathcal{R}(\boldsymbol{\omega}) + \mathrm{i} P(\boldsymbol{\omega}) \right] \right\} \right\| \tag{2.6}$$

Evidently, $\mathcal{F}^{-1}$ in Eq. (2.6) is the inverse DFT operator. Note that the phase spectrum of the time-series is preserved during this process and that all information regarding anomalies comes from the amplitude spectrum. Figure 2.1 depicts an example of how saliency maps highlight anomalies in a time-series.

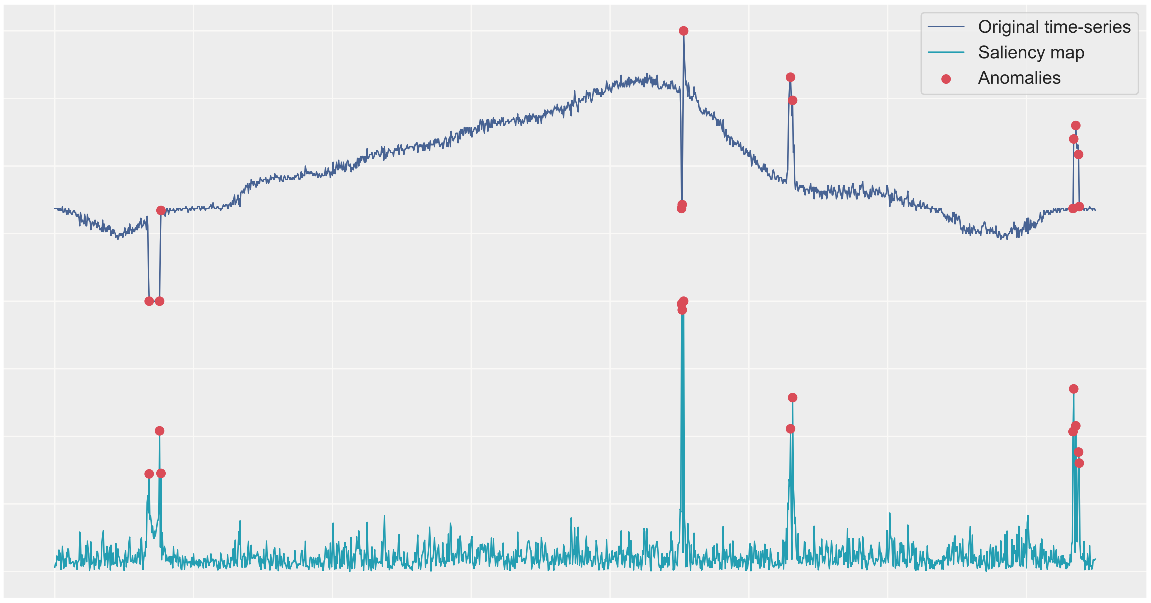


Figure 2.1: The upper line corresponds to the original time-series, while the lower line corresponds to its saliency map (both lines have been min-max scaled). Anomalies are marked with red.

The upper line of Fig. 2.1 corresponds to a time-series containing a small number of peaks and valleys that can be identified as anomalies even by visual inspection, while the lower line corresponds to its saliency map, obtained as explained through Eqs. (2.1) - (2.6). The saliency map is mainly composed of a somewhat homogeneous region of values, but also shows a small number of peaks, the locations of which coincide with the locations of the original time-series' peaks and valleys.

The way the peaks of the saliency map of Fig. 2.1 are classified as anomalies follows a simple, yet robust rule:

$$y_i = \begin{cases} 0, & \text{if } \frac{S(x_i)-\bar{S}(x_i)}{\bar{S}(x_i)} \leq \tau \\ 1, & \text{else} \end{cases}, \tag{2.7}$$

where

$$\bar{S}(\mathbf{x}) = \mathbf{h}_z \circledast S(\mathbf{x}) \tag{2.8}$$

is defined as the averaged saliency, similarly to how the averaged spectrum was defined in Eq. (2.3). The rule of Eq. (2.7) states that if the percentage difference between the saliency $S(x_i)$ at point x_i and the averaged saliency $\bar{S}(x_i)$ is higher than a selected threshold, τ, then this point is classified as an anomaly. Note that in this case $z \neq q$ is another hyper-parameter which determines the number, z, of preceding points to be considered for a local averaging of the saliency (in general, $z > q$). As for the threshold, τ, it is the last of the model's hyper-parameters and must be calibrated independently for each sensor's readings, depending on the desired sensitivity.

2.1.2 The SR-CNN Model

While the threshold utilized by the SR algorithm can be fine-tuned per sensor to yield results which are more than satisfactory, an even more sophisticated decision rule can be constructed using discriminative models. The main idea is to use the unlabeled time-series data in order to automatically construct a synthetic labeled dataset and then use the latter for supervised learning (a process that shares common elements, but is not equivalent to self-supervised learning).

More specifically, the original time-series is scanned using a sliding time-window of width w and a step β such that the step is smaller than the time-window ($\beta < w$), resulting in overlapping sliding windows. The data points within each window are treated as normal values and up to[2] κ points are randomly chosen so that they can be artificially transformed into anomalies using the transformation

$$x_i \to x_i + (\bar{x}_i + \mu) \cdot (1 + \sigma^2) \cdot r, \tag{2.9}$$

where $r \sim \mathcal{N}(0,1)$, μ and σ are each window's mean and standard deviation, respectively, and

$$\bar{x}_i = [\mathbf{h}_w \circledast \mathbf{x}]_i \tag{2.10}$$

is the local average of all points preceding x_i. This process, which is illustrated in Fig. 2.2, creates a labeled dataset of time-series data which are segments of the original input that contain artificially injected anomalies.

[2] For example, if $\kappa = 20$, then one time window may contain 17 artificial anomalies, while another one might contain 3.

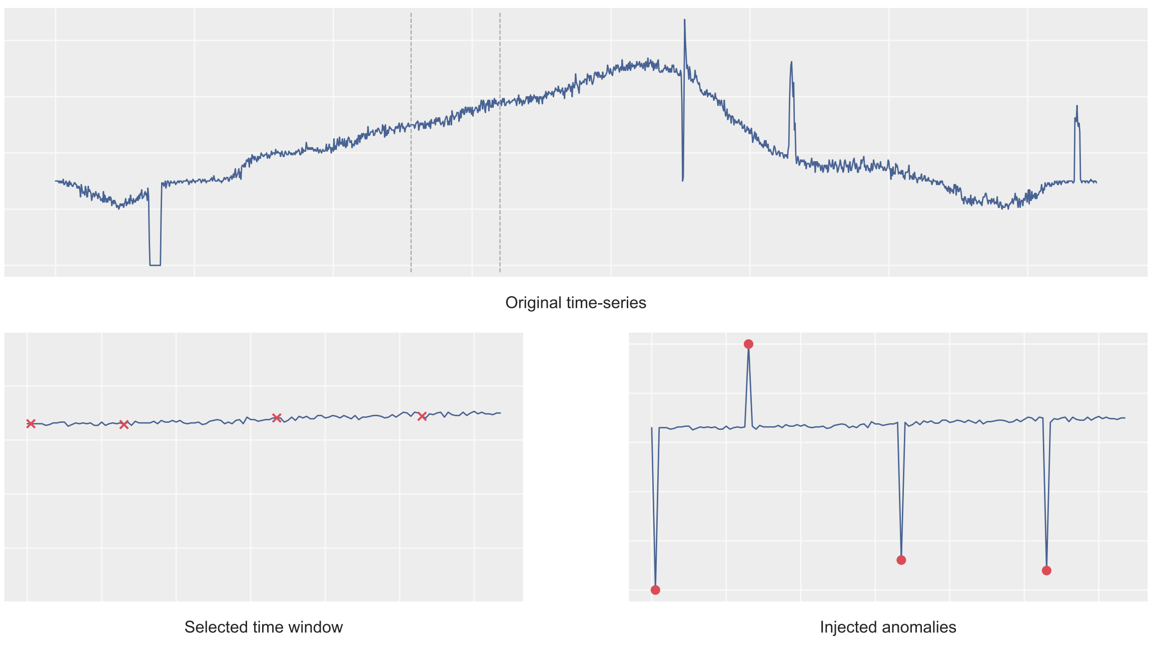


Figure 2.2: Top: the original time-series. Bottom: The selected time window's data (left) and the injected anomalies within this window (right).

The features that are extracted from this dataset are the saliency maps, using the steps described in Eqs. (2.1) - (2.6), while the labels are 1 for the injected anomalies and 0 for all other points. All that is left to do at this point is the supervised training of a discriminative model and among the best candidates for this, with proven efficiency on saliency detection on image data[30–32], are Convolutional Neural Networks (CNNs). While one single CNN architecture cannot be expected to yield optimal results for all types of sensor data, there are a few universal rules: in all cases, the first 1-dimensional convolutional layer's kernel's size must be equal to w, i.e. the sliding window's length. In addition, fully connected linear layers should be stacked at the end of the convolutional layers, with the final layer (of size w) having a sigmoid activation. The main components of the complete SR-CNN model described in the present section are schematically represented in the flowchart of Fig. 2.3.

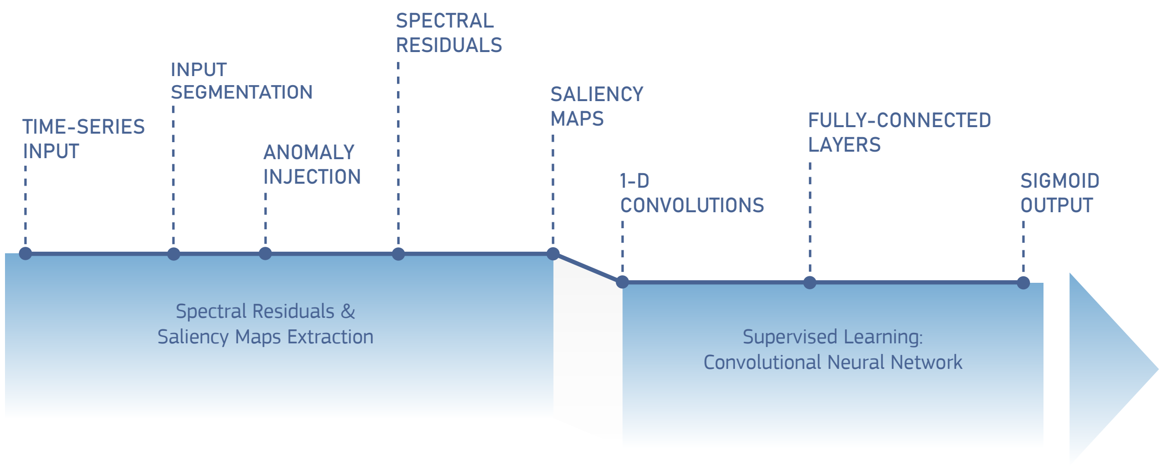


Figure 2.3: A complete flowchart depicting the complete SR-CNN model, from the saliency maps' extraction, to the CNN's supervised learning.

2.2 Multivariate Anomaly Detection

When it comes to uncorrelated sensors (i.e. the monitoring of uncorrelated features), while UVAD is a perfectly reasonable approach, its scalability becomes a significant challenge as the number of sensors increases. Even if their lifecycle maintenance is handled by good MLOps practices, one instance of a UVAD model for each sensor in a network of hundreds or thousands of sensors is computationally costly and inefficient. One could argue that this issue can be addressed by grouping sensors together based on the system they are monitoring, or depending on the type of feature they are recording. An example of the former approach would be a vessel with 6 engines, where an ensemble of UVAD models deployed using the feed of one engine's sensors could possibly be sufficient to make accurate predictions on the readings from the other 5 engines' sensors as well. As for the latter approach, by grouping sensors of different types together, one universal UVAD model per type could provide satisfactory results (for example, one UVAD model for all thermal sensors).

Despite the potential for reducing computational costs and "model bookkeeping" by implementing these ideas, and even if this is accomplished without making compromises on model accuracy, a significant limitation still arises in the case of correlated sensors. A fundamental principle in Statistical Physics states that a system's expected behaviour cannot be determined by examining the influence of each of its components individually, without taking their interactions into account. In the words of P. W. Anderson, *more is different* [33]. Based on this idea, in a network of sensors that monitor a system, the health status of the entire system is not reflected by the health status of each individual sensor, but rather by mutual information about the whole network. For instance, sudden perturbations in a single sensor's readings do not necessarily imply changes on the system as a whole and vice versa. This is where Multivariate Anomaly Detection (MVAD) enters the stage.

MVAD aims at detecting anomalies in multidimensional data by analyzing dependencies among sequences of multiple features. It solves both the issue of bookkeeping when handling a large number of models as well as the physical problem of missing the interactions between a number of features that are not taken into consideration when analyzing each feature individually. While there is a plethora of approaches when it comes to algorithms designed to solve the problem of MVAD [34–36], one of the most effective among them is perhaps also the most intuitive: the process of decomposing a system into its constituents along with their interactions can be represented by a graph. Consequently, Graph Neural Networks (GNNs) are a perfect candidate for solving this problem. Note that the final MVAD model used for the purposes of the present study is not itself a GNN, however GNNs are a very important part of its architecture. For this reason, the following sub-sections are devoted to a short overview of the constituents of the final MVAD model, starting with GNNs.

2.2.1 Graph Neural Networks

The motivation for the development of GNNs came from the abundance of problems that involve data whose structure can be represented in the form of graphs: social networks analysis, molecular structure prediction and language syntax trees, to name but a few. The first application of neural networks to graph-structured data involved the use of recursive neural networks to process data represented as directed acyclic graphs[37]. However, the term "Graph Neural Network" was not coined until almost a decade later, when Gori et al.[38] introduced GNNs as a generalization of recursive neural networks that can deal with a broader set of families of graphs, such as cyclic or undirected graphs. Scarselli et al.[39] and Gallicchio et al.[40] further elaborated on these ideas, with their collective work falling today into the category of Recurrent Graph Neural Networks (RecGNNs)[41]. RecGNNs consist of an iterative process, where neighbour information is propagated in an iterative manner until equilibrium (message passing), with the purpose of learning a target node's representation.

While the ideas that led to the development of RecGNNs were revolutionary, the models themselves have numerous limitations, with scalability being perhaps the most significant among them[42]. RecGNNs are based on iterative processes that are computationally demanding and involve a number of parameters that increases as the graph grows. Additionally, RecGNNs suffer from limited expressivity, as they fail to capture more complex relationships between nodes in a graph[43]. These observations, along with the desire to expand the notion of convolution to data that cannot be represented in grid-like structures, led to the development of Graph Convolutional Networks (GCNs). Besides, CNNs had already seen tremendous success in a range of problems spanning from image classification[44] to machine translation[45]. Research on GCNs is often divided into two main streams: the spectral-based and the spatial-based approaches.

As far as spectral-based approaches are concerned, Bruna et al.[46] and their pioneering work introduced a graph convolution based on spectral graph theory. More specifically, the convolution operation is performed in the spectral domain, i.e. on the eigenvectors of the graph Laplacian which are used to define the graph's Fourier basis. This approach is not without issues, however most of them were addressed by subsequent works. For example, the original spectral GCN led to spatially non-localized filters. Furthermore, the explicit eigendecomposition of the graph Laplacian imposed certain computational constraints. Nonetheless, it was found that the filters can be spatially localized either by proper reparameterizations[47], or by restricting the filters to operate in a nearest-neighbour setting[48], while approximations of the Laplacian could remove the need for solving the corresponding eigenvalues-eigenvectors problem exactly[49]. Despite these efforts and without denying the success of spectral GCN models on various tasks[50–52], a problem that appears to be inherent in this spectral-based approach is that it leads to filters that depend on the Fourier basis, which in turn depends on the graph itself. Consequently, spectral GCNs have inherent limitations when it comes to their capacity, as they cannot easily generalize to graphs with different structures.

When it comes to spatial-based approaches, their research started earlier compared to spectral-based ones, with the work of Micheli[53]. In spatial GCNs, convolutions are defined directly on the graph, operating on groups of nodes within given neighbourhood sizes. Obviously, this creates the problem of properly defining operators that combine the weight sharing property of CNNs and the ability to work with neighbourhoods of different sizes. Various suggestions have been made regarding this problem, for instance separating weight matrices for different nodes depending on their degree[54], or sampling a neighbourhood of fixed size for each node and then performing specific aggregations over it[55]. These approaches (especially the latter example) have indeed yielded impressive performance scores across numerous tasks, however a significant limitation of them is that the neighbours of each node are treated in a uniform manner, i.e. without taking into account cases where nodes are not influenced equally by all neighbours.

2.2.1.1 Graph Attention Networks

It becomes evident that, despite the fact that GCNs made several steps towards extending the notion of convolutions from grids to graphs, each approach faces its own shortcomings. By combining their issues, one may define the "ideal" convolution-like operation as one having the following properties: (1) *satisfactory complexity* (especially when it comes to temporal complexity); (2) *learning scalability* (the cardinality of the algorithm's set of parameters should not depend on the graph's size); (3) *high capacity* (a trained model should be able to generalize to unseen graphs of varying structures); (4) *spatial localization* (the convolution operation should be performed on a node's neighbourhood); and (5) *expressivity* (the algorithm should be able to capture more complex relationships between neighbouring nodes and would therefore need to assign different influence scores on each neighbour). Fig. 2.4 provides a schematic representation of what one such desirable operation would look like.

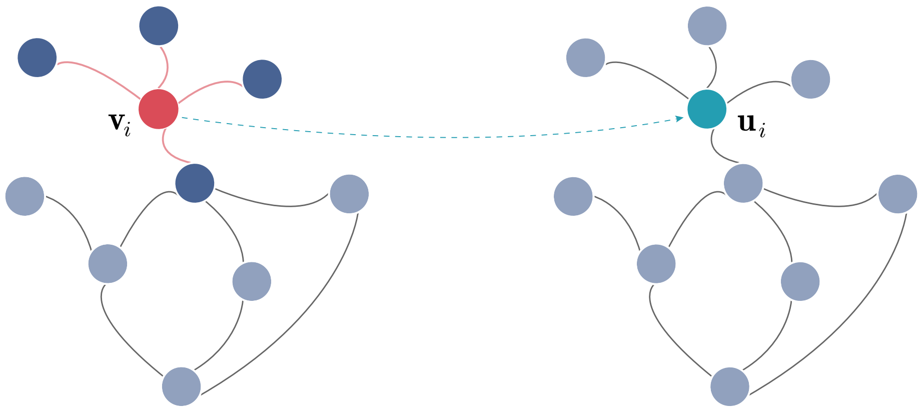


Figure 2.4: The desirable equivalent of convolutional operations on graph nodes.

Heavily influenced by the unprecedented breakthroughs of attention mechanisms[56,57] on virtually all sequence-based tasks, Veličković et al.[58] introduced an attention-based graph model, in an attempt to define the aforementioned ideal convolution-like operation. The building block of this architecture is known as a graph attentional layer and 1 or more such layers stacked together construct a Graph Attention Network (GAT).

The input to a graph attentional layer is a set of node vectors, $\mathbf{v} = \{\mathbf{v}_1, \dots, \mathbf{v}_k\}$, with $\mathbf{v}_i \in \mathbb{R}^T$, where k is the number of nodes and T is the number of elements per node vector. In the context of time-series data, each node may correspond to a single feature and T is equal to the total number of time steps. This input layer produces a new set of node vectors, $\mathbf{u} = \{\mathbf{u}_1, \dots, \mathbf{u}_k\}$, with $\mathbf{u}_i \in \mathbb{R}^{T'}$, where T' is in general different from T, but the cardinalities of $\mathbf{v}$ and $\mathbf{u}$ are equal. These new node vectors essentially correspond to higher-level features, similarly to how a CNN transforms its input in the context of grid-structured data. Given the graph's adjacency matrix, **A**, the original node vectors are transformed as

$$\mathbf{u}_i = \sigma\left(\mathbf{W}\cdot\sum_{j=1}^{k} A_{ij}\alpha_{ij}\mathbf{v}_j\right), \tag{2.11}$$

where $\sigma\left(\mathbf{x}\right)$ is the logistic sigmoid function and $\mathbf{W} \in \mathbb{R}^{T'\times T}$ is the weight matrix that corresponds to one of the graph attentional layer's learnable parameters. The presence of the adjacency matrix's elements in Eq. (2.11) ensures that only linked nodes are taken into account for this convolution-like operation, since $A_{ij} = 1$ only if (i, j) is an existing edge and 0 otherwise. As for α_{ij}, it corresponds to the neighbourhood-normalized version of the self-attention coefficients and it is given by

$$\alpha_{ij} = \text{softmax}\left(e_{ij}\right) = \frac{\exp\left(e_{ij}\right)}{\sum_m A_{im}\exp\left(e_{im}\right)}, \tag{2.12}$$

where e_{ij} are the original self-attention coefficients. These indicate the importance of node j's features to node i and are usually calculated as

$$e_{ij} = A_{ij}\ \text{LeakyReLU}\left(\mathbf{a}\cdot\left[\mathbf{W}\cdot\mathbf{v}_i \oplus \mathbf{W}\cdot\mathbf{v}_j\right]\right), \tag{2.13}$$

where $\oplus$ denotes vector concatenation and $\mathbf{a} \in \mathbb{R}^{2T'}$ is a weight vector which parameterizes the feed-forward neural network that serves as the attention mechanism. In fact, due to the appearance of A_{ij} in all equations, the attention is known as masked attention, since it allows each node to attend only to its linked nodes, without dropping the structural information about the graph. A bias vector may also be included before applying the LeakyReLU nonlinearity, however bias vectors are omitted throughout this analysis for brevity. Additionally, the negative slope of the LeakyReLU is one of the model's hyper-parameters, though in literature it is usually set equal to 0.2. In general, the self-attention coefficients can occur through any attention mechanism of the form

$$e_{ij} = f\left(\mathbf{w}_l\cdot\mathbf{v}_i, \mathbf{w}_r\cdot\mathbf{v}_j\right), \tag{2.14}$$

where $f : \mathbb{R}^{T'} \times \mathbb{R}^{T'} \to \mathbb{R}$. Nonetheless, ever since Veličković et al. suggested Eq. (2.13), most implementations and related works have been using this specific attention mechanism, which became synonymous with graph attention.

The aforementioned equations can be generalized in order to account for multi-head attention, which acts as a regularization factor: by extending the weight matrix to a tensor $\mathbf{W} \in \mathbb{R}^{T'\times T\times M}$, where M is the number of attention mechanisms that are employed, Eq. (2.13) becomes

$$e_{ij}^{n} = A_{ij}\ \text{LeakyReLU}\left(\mathbf{a}\cdot\left[\mathbf{W}^{m}\cdot\mathbf{v}_i \oplus \mathbf{W}^{m}\cdot\mathbf{v}_j\right]\right), \tag{2.15}$$

where $\mathbf{W}^{m} \in \mathbb{R}^{T'\times T}$ is the weight matrix associated with the m-th attention mechanism. Similarly, the normalized attentions are given by applying a softmax filter to the results of Eq. (2.15). As for how the node vectors are updated, Eq. (2.11) assumes the form

$$\mathbf{u}_i = \bigoplus_{m=1}^{M} \left[\sigma \left(\mathbf{W}^m \cdot \sum_{j=1}^{k} A_{ij} \alpha_{ij}^m \mathbf{v}_j \right) \right], \tag{2.16}$$

where now $\mathbf{u}_i \in \mathbb{R}^{T' \times M}$, as long as the graph attentional layer is not the final layer of the GAT, since it is more sensible to perform some kind of aggregation (mainly averaging) instead of concatenation on the prediction layer's outputs. In this case, the transformed node vectors are given by

$$\mathbf{u}_i = \sigma \left(\frac{1}{M} \sum_{m=1}^{M} \mathbf{W}^m \cdot \sum_{j=1}^{k} A_{ij} \alpha_{ij}^m \mathbf{v}_j \right). \tag{2.17}$$

These two different ways of transforming the original node vectors through multi-head attention are depicted in Fig. 2.5.

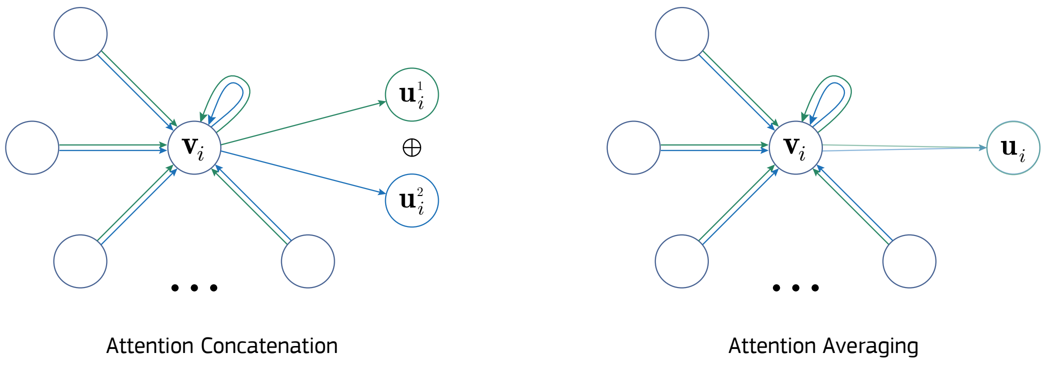


Figure 2.5: Depiction of concatenation (left) and averaging (right) in the case of multi-head attention for $M = 2$.

Apart from multi-head attention, it is worth noting that applying dropout[59] to the self-attention coefficients also serves as an efficient regularizer, especially in problems with small training datasets[58].

Returning to the five properties discussed regarding "ideal" convolution-like operations on graph, it becomes clear that the attention-based approach of Veličković et al. respects all of them: (1) the time complexity of a graph attentional layer with single-head attention is $\mathcal{O}(N_V T T' + N_E T')$, where N_V and N_E are the number of graph nodes and edges, respectively, the self-attention layer's operation is parallelizable across all edges and the computation of output features is parallelizable across all nodes; (2) the number of parameters is fixed and independent of the graph's node/edge count; (3) the attention mechanism does not depend on global graph structure and is therefore applicable to any type of graph; (4) the operation is spatially localized by default, since only linked nodes attend to each other; and (5) the resulting model is more expressive than GCNs, thanks to its inherent ability to assign different influence scores (attention) to different neighbours.

2.2.1.2 Static vs Dynamic Attention

As was the case with other models before it, the application of this self-attention mechanism to GNNs led to breakthroughs in a number of different tasks and datasets[60,61]. Additionally, it paved the road for the development of novel GAT-like architectures to tackle new, challenging problems[62–64]. It was not until 2022, when Brody et al.[65] presented their work, that the expressivity of GATs was first characterized as limited, despite being better than its predecessors'.

The authors argue that GATs do not compute the expressive type of attention they call *dynamic* attention, but rather a restricted, *static* form of attention. Their argument is that, given a specific query, any attention-based model should be able to "focus" on the most relevant input, which is only possible by "decaying" other inputs, i.e. assigning a lower score to these inputs. By adopting the terminology of Vaswani et al.[57], the authors prove that the attention mechanism utilized by GATs leads to a selection of keys regardless of the query. As a result, this mechanism hinders the model's expressivity in cases where the relevance of each key highly depends on the corresponding query. The formal proof of these statements along with a mathematical formulation of the concepts of static and dynamic attention can be found in Appendix A.

More specifically, the issue in GATs can be traced in Eq. (2.13), where the layers **W** and **a** are applied consecutively and could therefore be collapsed into a single linear layer without any deformational impact on the model's architecture. As a solution to this issue, Brody et al. suggest a simple modification in the order of internal operations: the input node vectors are concatenated before applying the **W** layer and the nonlinearity is applied before applying the **a** layer. Eq. (2.13) can then be rewritten as

$$e_{ij} = A_{ij}\, \mathbf{a} \cdot \mathrm{LeakyReLU}\left(\mathbf{W} \cdot \left[\mathbf{v}_i \oplus \mathbf{v}_j\right]\right), \tag{2.18}$$

where now $\mathbf{W} \in \mathbb{R}^{2T' \times 2T}$ [(3)]. While this change in order of operations may seem trivial at first sight, drawing the neural network that corresponds to each attention mechanism reveals their key differences (see Fig. 2.6).

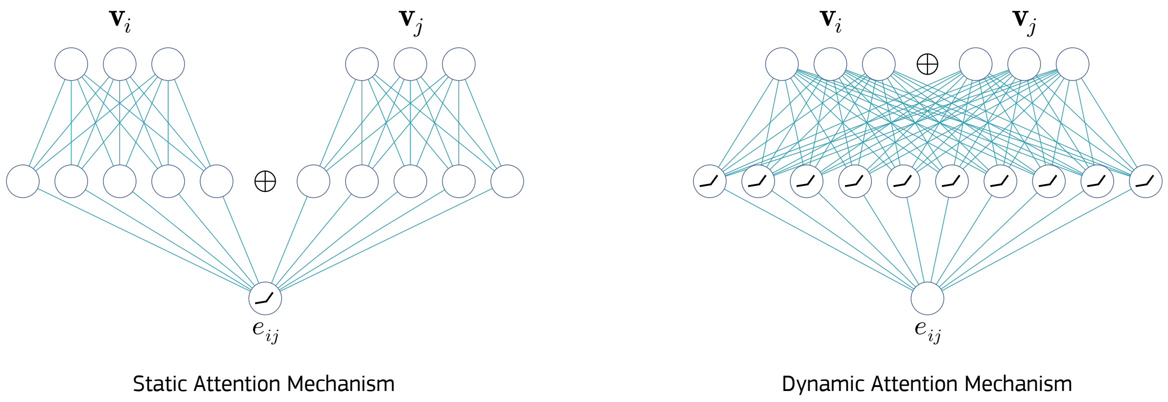


Figure 2.6: Feedforward network for the calculation of static attention (left) and dynamic attention (right) in the case where $T = 3$ and $T' = 5$.

The neural network that corresponds to the static attention mechanism proposed in the GAT architecture is indeed equivalent to a single layer, while the neural network that corresponds to the dynamic

(3) In fact, there is no practical reason in setting the number of rows of **W** equal to $2T'$, since T' is an arbitrary dimension independent of the original input. By renaming $2T' \to T'$, one simply makes a change of variables with absolutely no impact on the framework. Nonetheless, this notation makes it easier to draw parallels between GAT and GATv2 and this is why it is adopted.

attention mechanism is equivalent to a Multiple Layer Perceptron (MLP). As a result, the latter attention mechanism is strictly more expressive, since this advanced GAT - which the authors named GATv2 - is a universal approximator[66].

An issue with Eq. (2.18) is that it does not allow a direct inference of the new form of Eq. (2.11), i.e. the new transformation rule for the original node vectors, since the dimensions of **W** in GATv2 are different compared to the ones in GAT. For this reason, Eq. (2.18) can be equivalently written as

$$e_{ij} = A_{ij}\, \mathbf{a} \cdot \text{LeakyReLU}\left(\mathbf{w}_l \cdot \mathbf{v}_i + \mathbf{w}_r \cdot \mathbf{v}_j\right), \tag{2.19}$$

where $\mathbf{w}_r, \mathbf{w}_l \in \mathbb{R}^{2T' \times T}$ and **W** is simply the augmentation of $\mathbf{w}_l$ and $\mathbf{w}_r$, i.e. $\mathbf{W} = (\mathbf{w}_l | \mathbf{w}_r)$. Now, one can readily rewrite Eq. (2.11) as

$$\mathbf{u}_i = \sigma\left(\mathbf{w}_r \cdot \sum_{j=1}^{k} A_{ij}\alpha_{ij}\mathbf{v}_j\right). \tag{2.20}$$

For completeness, it is mentioned that in the case of multi-head attention, Eq. (2.19) is generalized to

$$e_{ij}^m = A_{ij}\, \mathbf{a} \cdot \text{LeakyReLU}\left(\mathbf{w}_r^m \cdot \mathbf{v}_i + \mathbf{w}_r^m \cdot \mathbf{v}_j\right) \tag{2.21}$$

and Eq. (2.20) is generalized to

$$\mathbf{u}_i = \bigoplus_{m=1}^{M}\left[\sigma\left(\mathbf{w}_r^m \cdot \sum_{j=1}^{k} A_{ij}\alpha_{ij}^m\mathbf{v}_j\right)\right] \tag{2.22}$$

for intermediate layers of a GATv2 and to

$$\mathbf{u}_i = \sigma\left(\frac{1}{M}\sum_{m=1}^{M}\mathbf{w}_r^m \cdot \sum_{j=1}^{k} A_{ij}\alpha_{ij}^m\mathbf{v}_j\right) \tag{2.23}$$

for the output layer of a GATv2.

2.2.2 Gated Recurrent Unit Networks

Apart from GNNs, another integral part of the final MVAD model's architecture is the Recurrent Neural Network (RNN)[67,68] that utilizes Gated Recurrent Unit (GRU) cells. Before diving into the specifics of GRUs, it's important to give a short overview of RNNs, which are a type of neural network designed to process sequential data such as speech or text - a problem that can't be handled by conventional feedforward neural networks. They achieve this via the use of feedback connections that allow them to maintain an internal state known as the hidden state that can be updated at each time step. This allows RNNs to take into account the context of the input, making them well suited for tasks involving sequential data.

The basic structure of the RNN starts with an input $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T) \in \mathbb{R}^{T\times F}$, where $\mathbf{x}_t \in \mathbb{R}^F$ is the input vector's element at time step t, which is simply a F-dimensional feature vector. This input is sequentially processed by the so-called cell (in the case of vanilla RNNs, the cell consists of a single smooth, bounded function, such as the logistic sigmoid or hyperbolic tangent) in order to produce a hidden state vector $\mathbf{h} = (\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T) \in \mathbb{R}^{T \times d}$, where d is the dimension of each hidden state vector element, also known as hidden dimension. The hidden dimension is often one of the hyper-parameters in the network's architecture. Note that, optionally, the RNN may include an output apart from the hidden state vector. In order to illustrate how the RNN processes the input sequence step by step, a helpful visualization trick involves unrolling it through time, as shown in Fig. 2.7.

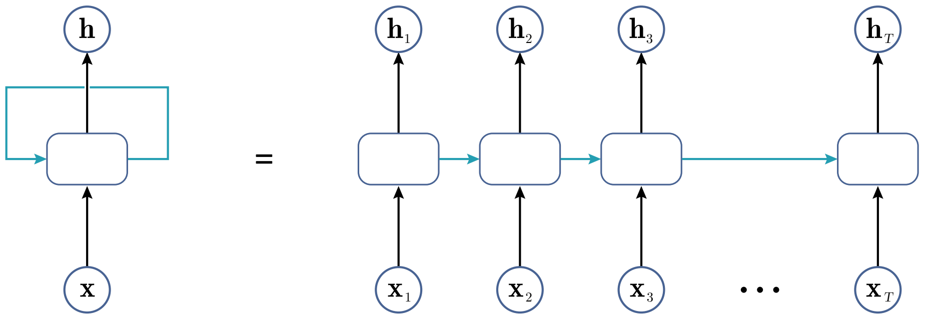


Figure 2.7: Unrolling Recurrent Neural Networks (RNNs) through time.

At each time step t, the RNN takes the $\mathbf{x}_t$ element from the input vector and updates its hidden state using this input and the previous time step's hidden state. The update rule for the vanilla RNN is

$$\mathbf{h}_t = f\Big(\mathbf{W} \cdot [\mathbf{x}_t \oplus \mathbf{h}_{t-1}]\Big), \tag{2.24}$$

where $\mathbf{W} \in \mathbb{R}^{d \times (F+d)}$ is the matrix containing the learnable weights and f is an activation function (usually the logistic sigmoid or hyperbolic tangent). Note that a bias can also be incorporated in the rule of Eq. (2.24) by adding a vector $\mathbf{b} \in \mathbb{R}^d$ in the activation function's argument, however in what follows biases will be omitted for brevity, as was done in previous sections. Another important remark is that $\mathbf{h}_0$ is set equal to the null vector in order to calculate $\mathbf{h}_1$ at time step $t = 1$.

The major among the shortcomings of vanilla RNNs is that they cannot be trained to capture long-term dependencies using gradient-based optimization strategies, due to the problem of vanishing gradients[69]. This is a consequence of the fact that the content of the hidden state is completely overwritten at each time step, even though the hidden state at time step t is calculated based on the hidden state at time step $t - 1$. As a result, the effect of long-term dependencies is hidden by the effects of short-term dependencies[70]. This issue can be addressed either by devising different learning algorithms[71], or by introducing gating mechanisms inside the cell that allow the network to selectively update or preserve its hidden state, thus allowing it to effectively capture both short-term and long-term dependencies. As far as the latter approach is concerned, the two most well-known gated cells are the Long Short-Term Memory (LSTM)[72] and the Gated Recurrent Unit (GRU)[73].

The main difference between LSTMs and GRUs is the number of gates they use. LSTMs use three gates, an input gate, an output gate, and a forget gate, while GRUs use only two gates, an update gate and a reset gate. In a nutshell, the LSTM's gates are used to control the flow of information into, out of, and within the cell state of the LSTM, while the update gate of the GRU controls how much of the previous hidden state to retain, and its reset gate controls how much of the new input to let through. Both cells have been successful in a wide range of applications such as natural language processing[74,75], speech recognition[76], and machine translation[77,78], so the choice of which one to use always depends on the specific task and the type of data being used. Experimentally, it has been found that LSTMs tend to perform better on tasks that highly prioritize remembering long-term dependencies, while GRUs tend to be more computationally efficient[70] and perform well on tasks that require the model to quickly adapt to changing input[79].

Based on this comparison and for reasons that become clearer in Section 2.2.4, the final MVAD model utilizes the RNN with a GRU cell as part of its architecture. Fig. 2.8 depicts the GRU cell and the update rule it follows to perform updates on the hidden state at each time step.

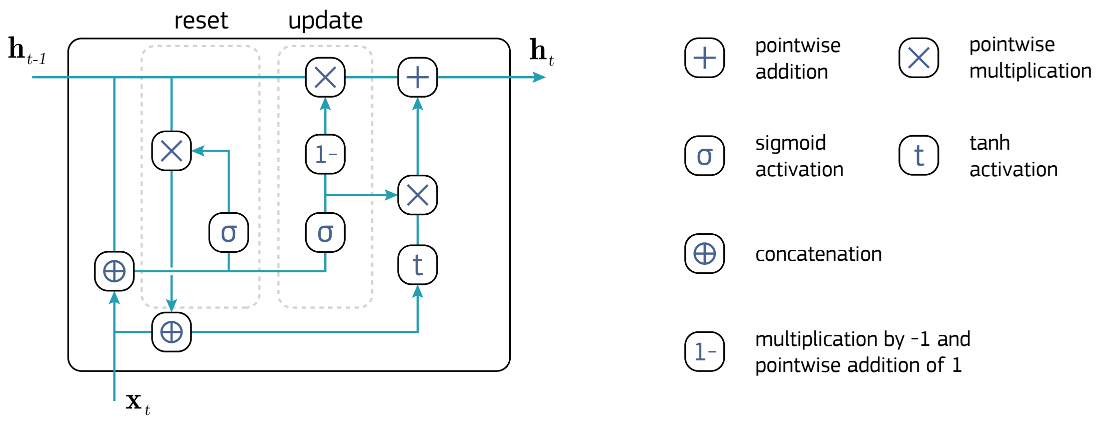


Figure 2.8: Schematic representation of the Gated Recurrent Unit (GRU) cell.

At time step t, the update and reset gates are calculated as

$$\mathbf{z}_t = \sigma\Big(\mathbf{W}^z \cdot [\mathbf{x}_t \oplus \mathbf{h}_{t-1}]\Big) \quad \text{and} \quad \mathbf{r}_t = \sigma\Big(\mathbf{W}^r \cdot [\mathbf{x}_t \oplus \mathbf{h}_{t-1}]\Big), \tag{2.25}$$

respectively, where $\mathbf{W}^{z/r} \in \mathbb{R}^{d\times(F+d)}$ is the matrix containing the learnable weights associated with the update/reset gate. Using these gates, a candidate hidden state vector element is defined as

$$\mathbf{n}_t = \tanh\Big(\mathbf{W}^n \cdot [\mathbf{x}_t \oplus \mathbf{r}_t \circ \mathbf{h}_{t-1}]\Big), \tag{2.26}$$

where $\circ$ denotes the Hadamard product (pointwise multiplication) and $\mathbf{W}^n \in \mathbb{R}^{d\times(F+d)}$ is the matrix containing the learnable weights associated with the candidate hidden state vector element. The reason why this quantity is known as the candidate hidden state vector element is because the actual hidden state vector element is given by

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \circ \mathbf{h}_{t-1} + \mathbf{z}_t \circ \mathbf{n}_t, \tag{2.27}$$

which is an update rule analogous to the one of Eq. (2.24).

Eq. (2.27) shows that the hidden state vector element at time step t is a linear interpolation between the previous time step's hidden state vector element and the current step's candidate hidden state vector element. In addition, this equation clarifies why $\mathbf{z}_t$ is called the update gate, since it is the vector that quantifies how much the previous hidden state vector element needs to update its content. In the extreme case where $\mathbf{z}_t = \mathbb{0}$, no update occurs and therefore all hidden state vector elements are equal to the null vector (since $\mathbf{h}_0 = \mathbb{0}$ by convention). Similarly, the reset gate quantifies how much the previous hidden state vector element is taken into account for the calculation of the candidate hidden state vector element. In the extreme case where $\mathbf{r}_t = \mathbb{0}$, the network effectively becomes a feedforward one, since each time the cell encounters a new item from the sequence, it treats it as if it is the first one, ignoring all previous input. Finally, in the extreme case where $\mathbf{z}_t = \mathbb{1}$ and $\mathbf{r}_t = \mathbb{1}$, the network is effectively equivalent to a vanilla RNN and the vanishing gradients problem resurfaces.

2.2.3 Variational Autoencoders

The last missing piece of the MVAD model's architecture is a structure known as the Variational Autoencoder[80,81] (VAE), however it is first worth providing a short summary of "traditional" Autoencoders[82] (see Fig. 2.9).

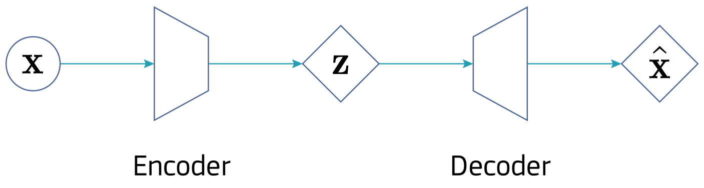


Figure 2.9: Schematic representation of an autoencoder. Deterministic variables are placed inside rhombi.

Autoencoders are deep neural network architectures whose main purpose is to learn a representation for a set of input data, in order to perform a series of tasks, such as feature engineering[83] (for example, dimensionality reduction), compression[84] (for example, in the case of image data), or data generation[85] (for example, to enrich limited datasets), to name but a few. Given an input vector $\mathbf{x} \in \mathbb{R}^d$, the left part of the autoencoder (known as the encoder) learns a low-dimensional latent representation $\mathbf{z} \in \mathbb{R}^L$ for it, where typically $L \ll d$. This low-dimensional layer is also known as the bottleneck. Then, the right part of the autoencoder (known as the decoder) tries to reconstruct $\mathbf{x}$ from $\mathbf{z}$, since the objective of the model is to create an output $\hat{\mathbf{x}} \in \mathbb{R}^d$ as close as possible to the original input: $\hat{\mathbf{x}} \approx \mathbf{x}$. It is stressed that these structures are presented herein in a completely abstract way, precisely because they do not correspond to specific architectures; their architecture depends on the type of input data and task. For instance, in a generative autoencoder for image data, the encoder usually uses CNN layers followed by linear layers, while the decoder utilizes linear layers followed by CNN layers.

It is important to note at this point that the traditional autoencoder is a purely deterministic model: feeding the model with a specific input $\mathbf{x}_0$, the output will always be $\hat{\mathbf{x}}_0$. In this sense, the VAE can be seen as the probabilistic extension of an autoencoder, where the encoder's and decoder's outputs are not single values, but rather parameters of probability distributions from which the $\mathbf{z}$ and $\hat{\mathbf{x}}$, respectively, are *sampled*. A schematic representation of the VAE can be seen in Fig. 2.10, where $\mathbf{z}$ and $\hat{\mathbf{x}}$ are not deterministic anymore (and therefore not placed inside rhombi), but instead are sampled from the distributions that are parameterized by the *probabilistic* encoder's and decoder's outputs (a procedure represented with dashed lines). Following Girin et al.[86], the formal presentation of the VAE is split into two main parts: one including the details of its decoder, which acts as a generative model, and one including the details of its encoder, which corresponds to an inference model.

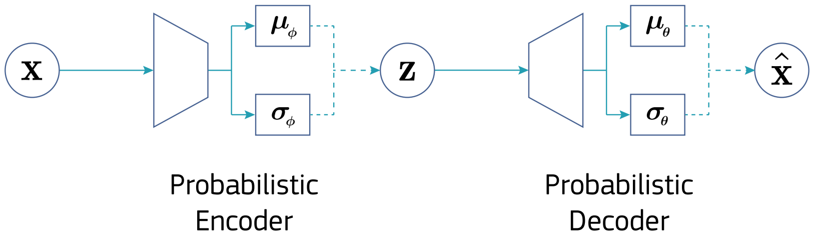


Figure 2.10: Schematic representation of a VAE. Dashed lines represent a sampling process. The general VAE architecture is not limited to specific probability distributions, however in the present example the encoder and decoder outputs are parameters for Gaussian distributions, indexed by ϕ and θ, respectively.

2.2.3.1 VAE Decoder: A Generative Model

The VAE decoder corresponds to a Deep Latent Variable Model (DLVM), i.e. a latent variable model whose distribution is parameterized by a deep neural network. It is a generative model, since it takes care of the reconstruction of the probability distribution of the input vector, based on the value of the latent variable. In fact, it can be considered as a generalization of generative mixture models with continuous latent variables instead of discrete ones[81]. More formally, it is defined by

$$p_\theta\left(\mathbf{x}, \mathbf{z}\right) = p_\theta\left(\mathbf{x}|\mathbf{z}\right) p\left(\mathbf{z}\right), \tag{2.28}$$

where θ denotes the set of parameters of the conditional distribution $p_\theta\left(\mathbf{x}|\mathbf{z}\right)$(4) and $p\left(\mathbf{z}\right)$ corresponds to the prior distribution of the latent variable. While the latent variable's prior distribution can in general also depend on its own set of parameters, the most common choice for it is an isotropic L-dimensional Gaussian,

$$p\left(\mathbf{z}\right) = \mathcal{N}\left(\mathbf{z}; \mathbb{0}, \mathbb{1}\right). \tag{2.29}$$

This is due to the fact that, as long as the functions parameterized by the θ-parameters are expressive enough (which is always the case with deep neural networks), the shape of the prior distribution does not really matter[87]. As for the conditional distribution, any probability distribution function can be used(5), depending on the nature of the input vector, $\mathbf{x}$. Nonetheless, a common choice is a Gaussian distribution with a diagonal covariance matrix, $\boldsymbol{\Sigma}_\theta\left(\mathbf{z}\right) = \operatorname{diag}\left(\left\{\sigma_\theta^{(i)\,2}\left(\mathbf{z}\right)\right\}_{i=1}^{d}\right)$:

$$p_\theta\left(\mathbf{x}|\mathbf{z}\right) = \mathcal{N}\left(\mathbf{x}; \boldsymbol{\mu}_\theta\left(\mathbf{z}\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{z}\right)\right) = \prod_{i=1}^{d} \mathcal{N}\left(x^{(i)}; \mu_\theta^{(i)}\left(\mathbf{z}\right), \sigma_\theta^{(i)\,2}\left(\mathbf{z}\right)\right), \tag{2.30}$$

where $\boldsymbol{\mu}_\theta : \mathbb{R}^L \to \mathbb{R}^d$ and $\boldsymbol{\sigma}_\theta : \mathbb{R}^L \to \mathbb{R}_+^d$ are deep neural network functions of the latent variable (for example, MLPs) parameterized by the θ-parameters (for example, weights and biases).

(4) Strictly speaking, the decoder is defined by $p_\theta\left(\hat{\mathbf{x}}, \mathbf{z}\right)$, since using the same variable name at both the input and output is an abuse of notation. However, it is justifiable in the present case, where the objective is to reproduce the input variable's distribution in the output.

(5) For example, it has been found that Gamma distributions are a good choice in the case of audio data[88].

When it comes to the training of the decoder, it amounts to optimizing the parameters θ so that the Kullback-Leibler (KL) divergence between the original data distribution, $p^*(\mathbf{x})$, and the model distribution, $p_\theta(\mathbf{x})$ is minimized:

$$\begin{aligned}\min_\theta \left\{D_{\mathrm{KL}}\Big(p^*(\mathbf{x}) \parallel p_\theta(\mathbf{x})\Big)\right\} &\iff \min_\theta \left\{\mathbb{E}_{p^*(\mathbf{x})}\left[\log p^*(\mathbf{x}) - \log p_\theta(\mathbf{x})\right]\right\} \\ &\iff \max_\theta \left\{\mathbb{E}_{p^*(\mathbf{x})}\left[\log p_\theta(\mathbf{x})\right]\right\} \qquad (2.31)\end{aligned}$$

Eq. (2.31) indicates that the minimization of the KL divergence is equivalent to a Maximum Likelihood Estimation (MLE) for the model distribution, i.e. the marginal likelihood:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})\, p(\mathbf{z})\; d\mathbf{z}. \qquad (2.32)$$

In practice, the original data distribution $p^*(\mathbf{x})$ is never known, however one can assume that the training dataset $\mathbf{X} = \{\mathbf{x}_n \in \mathbb{R}^d\}_{n=1}^N$ consists of N independent and identically distributed (i.i.d.) data vectors sampled from $p_\theta(\mathbf{x})$. In this case, the expectation of Eq. (2.31) can be replaced by the Monte Carlo estimate

$$\max_\theta \left\{\frac{1}{M}\sum_{m=1}^{M} \log p_\theta(\mathbf{x}_m)\right\}. \qquad (2.33)$$

The issue with this expression is that the marginal likelihood which is present in Eq. (2.33) is analytically intractable, since the integral of Eq. (2.32) involves a highly non-linear function of $\mathbf{z}$ (because the parameters of the conditional distribution are generated from a deep neural network). That said, the standard approach is to utilize Expectation-Maximization (EM) variational algorithms[89]: these leverage the latent variable nature of the model to maximize a lower bound of the intractable marginal log-likelihood[90], which depends on the posterior distribution of the latent variable.

More formally, let $\mathcal{F}$ denote a variational family defined as a set of probability distribution functions over the latent variable $\mathbf{z}$. For any variational distribution $q(\mathbf{z}) \in \mathcal{F}$, one may re-write the marginal log-likelihood as

$$\begin{aligned}\log p_\theta(\mathbf{x}_n) = \mathbb{E}_{q(\mathbf{z})}\left[\log p_\theta(\mathbf{x})\right] &= \mathbb{E}_{q(\mathbf{z})}\left[\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\right)\right] \\ &= \mathbb{E}_{q(\mathbf{z})}\left[\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\frac{q(\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\right)\right] \\ &= \underbrace{\mathbb{E}_{q(\mathbf{z})}\left[\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)\right]}_{\mathcal{L}(\theta, q(\mathbf{z}); \mathbf{x})} + \underbrace{\mathbb{E}_{q(\mathbf{z})}\left[\log\left(\frac{q(\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\right)\right]}_{D_{\mathrm{KL}}\left(q(\mathbf{z}) \parallel p_\theta(\mathbf{z}|\mathbf{x})\right)}, \qquad (2.34)\end{aligned}$$

where the second term is the KL divergence between $q(\mathbf{z})$ and the posterior $p_\theta(\mathbf{z}|\mathbf{x})$, and the first term, $\mathcal{L}(\theta, q(\mathbf{z}); \mathbf{x})$, is referred to in literature as the evidence lower bound (ELBO). Based on the fact that any KL divergence is non-negative, it becomes evident that the ELBO is a lower bound on the marginal log-likelihood, i.e.

$$\log p_\theta(\mathbf{x}) \geq \mathcal{L}(\theta, q(\mathbf{z}); \mathbf{x}), \tag{2.35}$$

with the equality holding only when the variational distribution $q(\mathbf{z})$ is exactly equal to the posterior. Based on Eq. (2.35), instead of focusing on maximizing the marginal log-likelihood, one may focus on maximizing the ELBO, first with respect to $q(\mathbf{z})$ (E-step) and then with respect to θ (M-step). Of course, Eq. (2.34) indicates that the E-step involves finding the variational distribution $q(\mathbf{z})$ in the family $\mathcal{F}$ that best approximates the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$ according to the KL divergence:

$$q^*(\mathbf{z}) = \arg\max_{q\in\mathcal{F}} \mathcal{L}(\theta, q(\mathbf{z}); \mathbf{x}) = \arg\min_{q\in\mathcal{F}} D_{\mathrm{KL}}(q(\mathbf{z}) \parallel p_\theta(\mathbf{z}|\mathbf{x})). \tag{2.36}$$

If the variational family $\mathcal{F}$ is unconstrained, the solution to the E-step is trivially given by the exact posterior distribution, i.e. $q^*(\mathbf{z}) = p_\theta(\mathbf{z}|\mathbf{x})$. Note that this optimal distribution over $\mathbf{z}$ is actually conditioned on $\mathbf{x}$, therefore henceforth the notation $q(\mathbf{z}|\mathbf{x})$ will be used instead of $q(\mathbf{z})$. However, the intractability of the marginal likelihood leads to an intractability of the posterior distribution, which in turn hinders any attempt at solving the E-step analytically. To tackle this issue, the variational family $\mathcal{F}$ is constrained by being re-defined as a set of variational distributions with a certain parametric form $q_\lambda(\mathbf{z}|\mathbf{x})$, where the λ-parameters govern the shape of the distribution. This is known as variational inference[90]. The ELBO thus becomes a function of both the decoder's parameters, θ, as well as the variational parameters, λ:

$$\mathcal{L}(\theta, \lambda; \mathbf{x}) = \mathbb{E}_{q_\lambda(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\lambda(\mathbf{z}|\mathbf{x})\right]. \tag{2.37}$$

What is more, the E-step of Eq. (2.36) reduces to the problem of optimizing the λ-parameters as

$$\lambda^* = \arg\max_{\lambda} \mathcal{L}(\theta, \lambda; \mathbf{x}) = \arg\min_{\lambda} D_{\mathrm{KL}}(q_\lambda(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}|\mathbf{x})). \tag{2.38}$$

2.2.3.2 VAE Encoder: An Inference Model

In general, given a dataset of i.i.d. data vectors $\mathbf{X} = \{\mathbf{x}_n \in \mathbb{R}^d\}_{n=1}^N$, one needs to find the parameters $\Lambda = \{\lambda_n\}_{n=1}^N$ of the variational distributions $q_{\lambda_n}(\mathbf{z}_n|\mathbf{x}_n)$, for $n = 1, \dots, N$. This is done by maximizing the total ELBO

$$\mathcal{L}(\theta, \Lambda; \mathbf{X}) = \sum_{n=1}^{N} \mathcal{L}(\theta, \lambda_n; \mathbf{x}_n), \tag{2.39}$$

which is the sum of the local ELBO defined in Eq. (2.37) over each vector in $\mathbf{X}$. Evidently, this procedure is costly for large datasets, which is why an even stronger assumption can be made when defining the variational family: all variational distributions $q_{\lambda_n}(\mathbf{z}_n|\mathbf{x}_n)$ share a common set of parameters, ϕ, which is introduced through an inference model f_ϕ that satisfies

$$\lambda_n = f_\phi\left(\mathbf{x}_n\right). \tag{2.40}$$

This is known as amortized variational inference and it introduces a single model which is used to map each observation $\mathbf{x}_n$ to a local variational parameter, λ_n. This stronger assumption effectively transforms the variational distributions as well as the ELBO itself into functions of ϕ:

$$\mathcal{L}\left(\theta, \phi; \mathbf{X}\right) = \sum_{n=1}^{N} \mathbb{E}_{q_\phi\left(\mathbf{z}_n|\mathbf{x}_n\right)}\left[\log p_\theta\left(\mathbf{x}_n, \mathbf{z}_n\right) - \log q_\phi\left(\mathbf{z}_n|\mathbf{x}_n\right)\right]. \tag{2.41}$$

Therefore, the optimization of the set of local variational parameters, Λ, is equivalent to the optimization of the shared set of inference model parameters, ϕ. This inference model corresponds to the VAE's encoder and, similar to the decoder, it corresponds to a DLVM. A common (however not the only) choice for it is

$$q_\phi\left(\mathbf{z}|\mathbf{x}\right) = \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_\phi\left(\mathbf{x}\right), \boldsymbol{\Sigma}_\phi\left(\mathbf{x}\right)\right) = \prod_{i=1}^{L} \mathcal{N}\left(z^{(i)}; \mu_\phi^{(i)}\left(\mathbf{x}\right), \sigma_\phi^{(i)\,2}\left(\mathbf{x}\right)\right), \tag{2.42}$$

i.e. a Gaussian distribution with $\boldsymbol{\Sigma}_\phi\left(\mathbf{x}\right) = \mathrm{diag}\left(\left\{\sigma_\phi^{(i)\,2}\left(\mathbf{x}\right)\right\}_{i=1}^{d}\right)$. In this expression, $\boldsymbol{\mu}_\phi : \mathbb{R}^d \to \mathbb{R}^L$ and $\boldsymbol{\sigma}_\phi : \mathbb{R}^d \to \mathbb{R}_+^L$ are deep neural network functions of the input variable (for example, MLPs) parameterized by the ϕ-parameters (for example, weights and biases).

Returning to the ELBO of Eq. (2.41), by using Bayes' theorem it can be equivalently written as

$$\begin{aligned}\mathcal{L}\left(\theta, \phi; \mathbf{X}\right) &= \sum_{n=1}^{N} \mathbb{E}_{q_\phi\left(\mathbf{z}_n|\mathbf{x}_n\right)}\left[\log p_\theta\left(\mathbf{x}_n|\mathbf{z}_n\right) + \log p\left(\mathbf{z}_n\right) - \log q_\phi\left(\mathbf{z}_n|\mathbf{x}_n\right)\right] \\ &= \underbrace{\sum_{n=1}^{N} \mathbb{E}_{q_\phi\left(\mathbf{z}_n|\mathbf{x}_n\right)}\left[\log p_\theta\left(\mathbf{x}_n|\mathbf{z}_n\right)\right]}_{\text{Reconstruction accuracy}} - \underbrace{\sum_{n=1}^{N} D_{\mathrm{KL}}\Big(q_\phi\left(\mathbf{z}_n|\mathbf{x}_n\right) \parallel p\left(\mathbf{z}_n\right)\Big)}_{\text{Regularization term}}.\end{aligned} \tag{2.43}$$

The first term of Eq. (2.43) is the reconstruction accuracy that represents the average accuracy of reconstructing the distribution parameters of the original input's distribution, while the second term acts as a regularizer that enforces the approximate posterior distribution to be close to the prior. In practice, when using Gaussian priors as in Eq. (2.29), this term forces different entries of $\mathbf{z}$ to be independent and encode a different characteristic of the input data[86]. For most commonly chosen distributions, the regularization term has an analytical expression as a function of θ and ϕ (see Appendix B for the example of Gaussian prior and posterior), though the same can't be said for the reconstruction accuracy term, where the expectation taken with respect to the approximate posterior renders it analytically intractable. As a result, it is approximated using a Monte Carlo estimate with M i.i.d. samples from $q_\phi\left(\mathbf{z}_n|\mathbf{x}_n\right)$, similarly to what was done in Eq. (2.33):

$$\mathcal{L}\left(\theta, \phi; \mathbf{X}\right) \simeq \sum_{n=1}^{N} \frac{1}{M} \sum_{m=1}^{M} \log p_\theta\left(\mathbf{x}_n|\mathbf{z}_{n,m}\right) - \sum_{n=1}^{N} D_{\mathrm{KL}}\Big(q_\phi\left(\mathbf{z}_n|\mathbf{x}_n\right) \parallel p\left(\mathbf{z}_n\right)\Big). \tag{2.44}$$

In the standard VAE methodology[80] the approximated ELBO of Eq. (2.44) is optimized using gradient-based optimization techniques to learn the generative and inference model parameters: the encoder and decoder DLVMs are cascaded and the sets of θ- and ϕ-parameters are jointly estimated from the training dataset(6). In terms of the gradient descent technique, the gradient of $\mathcal{L}\left(\theta, \phi; \mathbf{X}\right)$ with respect to θ can be easily computed using the backpropagation algorithm. However, backpropagation cannot be applied for the gradient with respect to ϕ, because the aforementioned sampling operation from the approximate posterior distribution is not differentiable. To bypass this issue, Kingma and Welling[80] proposed a so-called reparameterization trick: the sample $\mathbf{z}_{n,m}$ is reparameterized as

$$\mathbf{z}_{n,m} = \boldsymbol{\mu}_{\phi}\left(\mathbf{x}_n\right) + \operatorname{diag}\left\{\boldsymbol{\sigma}^2_{\phi}\left(\mathbf{x}_n\right)\right\}^{1/2} \boldsymbol{\varepsilon}_m, \tag{2.45}$$

where $\boldsymbol{\varepsilon}_m$ is a sample drawn from an L-dimensional standard Gaussian distribution,

$$\boldsymbol{\varepsilon}_m \sim \mathcal{N}\left(\mathbb{0}, \mathbb{1}\right), \tag{2.46}$$

which does not depend on ϕ. This trick essentially transfers the stochastic nature of the procedure from $\mathbf{z}_{n,m}$ to a new stochastic node, since $\mathbf{z}_{n,m}$ can be seen as a deterministic function of ϕ, $\mathbf{x}_n$ and $\boldsymbol{\varepsilon}_m$. As such, the ELBO becomes differentiable with respect to ϕ and the backpropagation algorithm can be utilized. This procedure is illustrated in the computation graphs of Fig. 2.11.

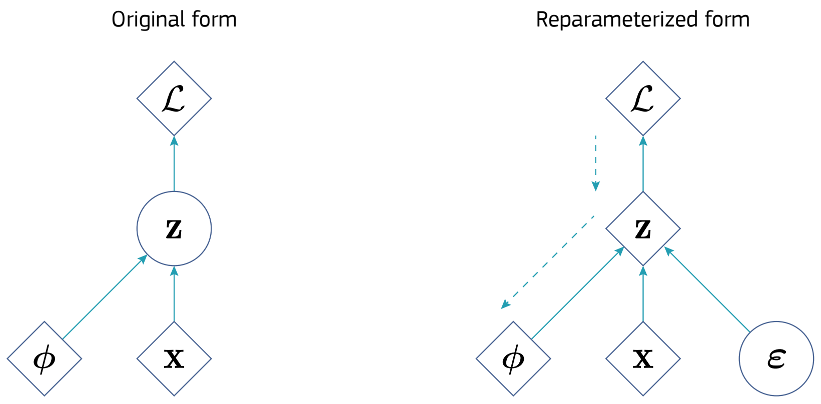


Figure 2.11: Illustration of the reparameterization trick performed to enable backpropagation, where deterministic variables are placed inside rhombi and stochastic variables are placed inside circles. The dashed arrows correspond to the calculation of gradients with respect to ϕ during backpropagation, which can only be done in the reparameterized form.

Closing the discussion on VAEs, it is worth noting that the gradient of the approximated ELBO given by Eq. (2.44) with respect to ϕ is an unbiased estimate of the gradient of the exact ELBO[81]. This allows using very few samples during the training of the VAE (even setting $M = 1$ is a valid choice[80]), as long as the training is done in a mini-batch setting (for example, with stochastic gradient descent) and a sufficiently large number of mini-batches is used.

(6) Note that this is different from an EM algorithm, which would alternatively optimize the ELBO with respect to ϕ and θ in the E- and M-step, respectively.

2.2.4 Piecing Everything Together: the MTAD-GATv2 Model

Having formally defined all the constituents of the final MVAD model, this chapter's final section is devoted to a thorough presentation of its full architecture. The model is based on Microsoft's Multivariate Time-Series Anomaly Detection via Graph Attention Network (MTAD-GAT) [91] and is called MTAD-GATv2. In a nutshell, the model treats each univariate time-series in a sensor network as an individual feature and tries to perform anomaly detection by modelling both the correlations between different features, as well as the temporal dependencies within each time-series.

The length of the multivariate time-series, T, is taken to be equal to the total number of timestamps available for the longest univariate time-series and all operations involved in the model's architecture are performed in a sliding window setting, with constant length Δt. First, the pre-processed time-series for each feature (details on the pre-processing are provided in the following chapter) passes through a 1-dimensional convolutional layer, in order to extract high-level information as a type of local feature engineering. The outputs of this convolutional layer are then processed by two parallel GATv2 layers, hence the name of the model(7): a feature-oriented GATv2 layer and a time-oriented GATv2 layer.

For the feature-oriented GATv2 layer, the multivariate time-series within a sliding window is considered as a complete graph where each node corresponds to a certain feature and each edge represents the relationship between two features. This allows the GATv2 layer to detect multivariate correlations without any prior knowledge, as the relationships between nodes are carefully captured through the dynamic attention mechanism. On the other hand, the time-oriented GATv2 layer considers all the timestamps within each sliding window as a complete graph, with each node corresponding to a feature vector at a given timestamp. This allows the model to capture all temporal dependencies in the time-series, similarly to how a transformer [56] utilizes self-attention to model a sequence of words.

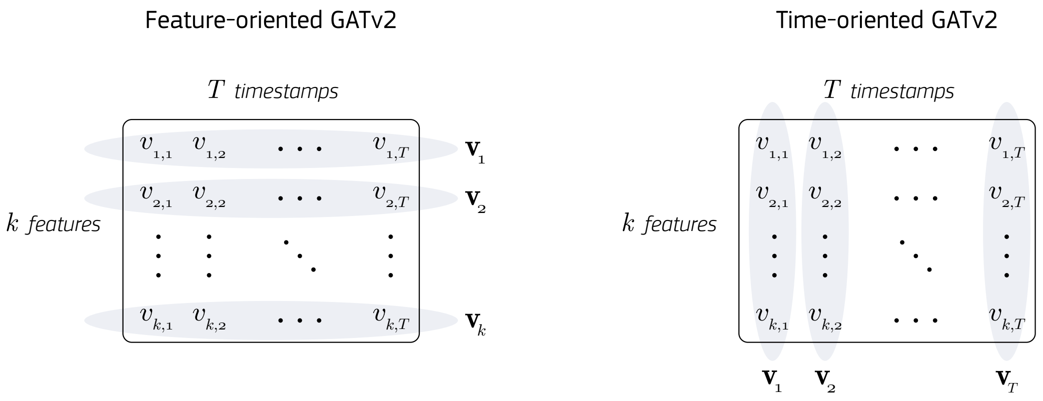

Feature-oriented GATv2
T timestamps
k features
$v_{1,1}$ $v_{1,2}$ $\cdots$ $v_{1,T}$
$v_{2,1}$ $v_{2,2}$ $\cdots$ $v_{2,T}$
$v_{k,1}$ $v_{k,2}$ $\cdots$ $v_{k,T}$
$\mathbf{v}_1$
$\mathbf{v}_2$
$\mathbf{v}_k$
Time-oriented GATv2
T timestamps
k features
$\mathbf{v}_1$ $\mathbf{v}_2$ $\mathbf{v}_T$

Figure 2.12: The node vectors corresponding to the feature-oriented (left) and time-oriented (right) GATv2 layers. For this illustration $\Delta t = T$ has been assumed.

More formally, in the feature-oriented GATv2 layer, each node is represented by a node vector $\mathbf{v}_i = \{v_{i,t}\}_{t=1}^{\Delta t}$, where $v_{i,t}$ is the value for the i-th feature at time step t. For a total of k nodes,

(7) In the original work by Zhao et al. [91] regular GAT layers are applied, which have the limitations discussed in Section 2.2.1.2 due to the calculation of static instead of dynamic attention

the final output of this layer is a $k \times T$ matrix, where each row is a T-dimensional vector representing the GATv2 layer's output for a single node. In the time-oriented GATv2 layer, each node is represented by a node vector $\mathbf{v}_t = \{v_{t,i}\}_{i=1}^{k}$, where $v_{t,i}$ is the value at time step t corresponding to the i-th feature. For a total of T timestamps, the final output of this layer is a $T \times k$ matrix, where each row is a T-dimensional vector representing the GATv2 layer's output for a single node(8). The node vectors utilized by each GATv2 layer can be seen in Fig. 2.12.

The transposed $k \times T$ output of the feature-oriented GATv2 layer, the $T \times k$ output of the time-oriented GATv2 layer and the original input of the layers (the 1-dimensional convolutional layer's output) are then concatenated to a single $T \times F$ matrix, where $F = 3k$. Each row of this matrix corresponds to a F-dimensional feature vector which fuses the original input's information with the feature-based and temporal dynamic attention scores. This concatenated product is then fed to a GRU layer of hidden dimension d, in order to capture sequential patterns and long-term dependencies in the time-series. As was discussed in Section 2.2.2, LSTM networks are better at capturing such dependencies. Nevertheless, GRU networks are computationally more efficient and, most importantly, they perform better when the model needs to adapt to changing input. In the present case, the parameters associated with the attention mechanisms of the GATv2 layers are learnable, therefore the output of these layers - which is part of the GRU layer's input - changes with every training epoch.

The final part of the MTAD-GATv2 architecture corresponds to a forecasting-based model and a reconstruction-based model, which work in parallel using the GRU's output. The former is utilized so that the model can make predictions for the value at the next timestamp, while the latter is used to capture the data distribution of the entire multivariate time-series. The chosen models are a MLP for the forecasting process and a VAE with bottleneck dimension L for the reconstruction. Note that in this case the VAE's objective is not to reconstruct a vector similar to the GRU's output, but rather to reconstruct a vector similar to the input time-series' k-dimensional feature vector. For this reason, while the GRU's output is d-dimensional, the dimension of the VAE's decoder's output must be taken equal to k(9). The architecture of the full MTAD-GATv2 model is depicted in Fig. 2.13.

During the training process, the learnable parameters from both models are updated simultaneously and the loss function is defined as the sum of both optimization targets:

$$\mathcal{L} = \mathcal{L}_f + \mathcal{L}_r, \tag{2.47}$$

where $\mathcal{L}_f$ and $\mathcal{L}_r$ are the loss functions associated with the forecasting and the reconstruction model, respectively. During inference, at a given timestamp, the MLP provides a prediction $\hat{\mathbf{x}} = \{\hat{x}_i\}_{i=1}^{k}$, where $\hat{x}_i$ is the predicted value for the i-th feature. On the other hand, the VAE receives the actual

(8) Note that, following the original paper presenting the MTAD-GAT, in the present work the mapping $\mathbb{R}^{T/k} \to \mathbb{R}^{T/k}$ is also chosen for the feature-/time-oriented GATv2 layer, respectively. Nonetheless, this is not a limitation by design of the GATv2 as seen in Section 2.2.1.1, which means that the mappings can be generalized to $\mathbb{R}^{T} \to \mathbb{R}^{T'}$ and $\mathbb{R}^{k} \to \mathbb{R}^{k'}$.

(9) Another viewpoint would be to consider the full MTAD-GATv2 model as a VAE, where the encoder consists of everything other than the forecasting model and the decoder model. Then, the dimension of the VAE's input would be equal to the dimension of its output.

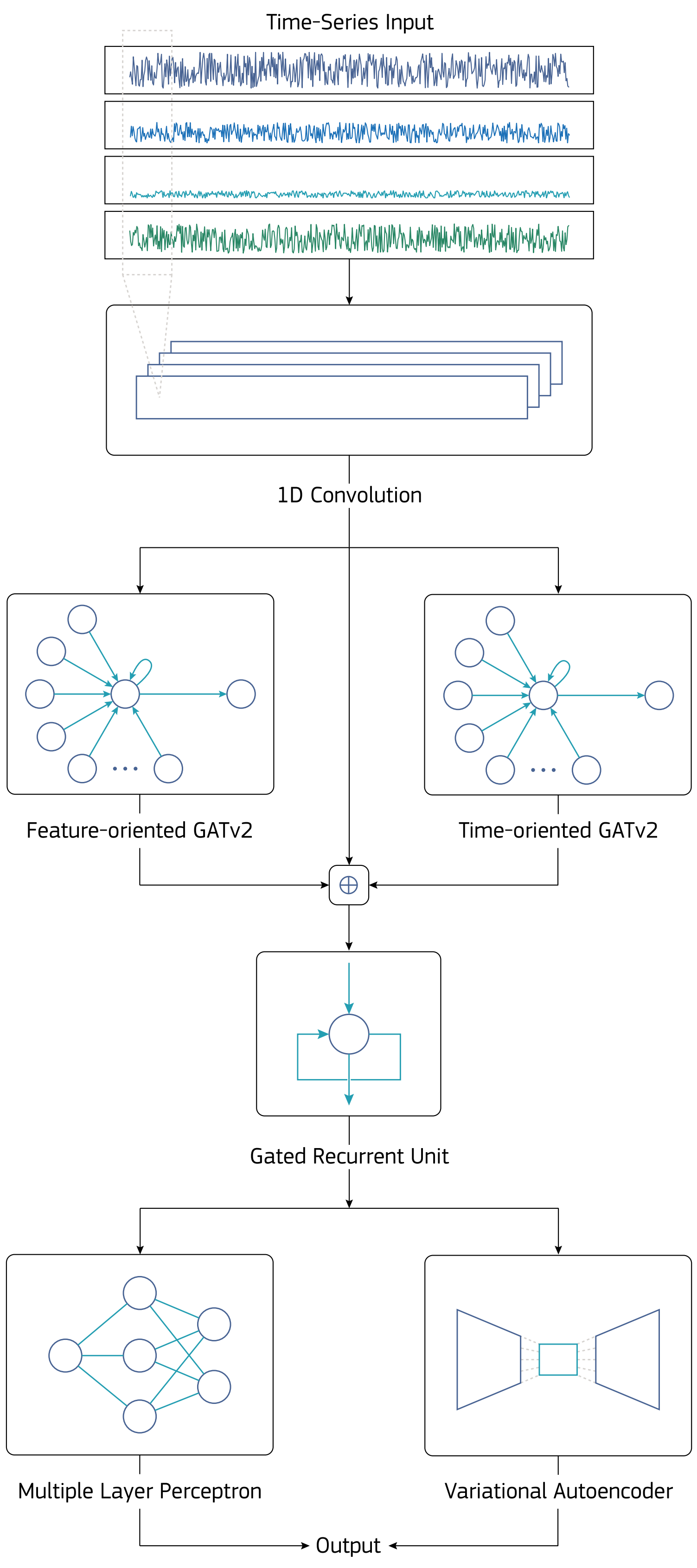


Figure 2.13: Architecture of the MTAD-GATv2 model.

measured value of each feature, x_i, and tries to reconstruct it, with its decoder producing an output vector $\mathbf{r} = \{r_i\}_{i=1}^{k}$, where r_i is the reconstructed value for the i-th feature. The final output of the model is an overall inference score, which is the sum of the inference score per feature, given by

$$s = \sum_{i=1}^{k} s_i = \sum_{i=1}^{k} \frac{|\hat{x}_i - x_i| + \gamma\,|x_i - r_i|}{1+\gamma}, \tag{2.48}$$

where γ is a hyper-parameter that weights the relative contribution of the reconstruction and forecasting process to the final score. This score takes into account the squared error between the prediction of the forecasting model and the actual measurement, in order to quantify the former's deviation from the latter. Additionally, it considers the absolute error between the measurement and the reconstructed value, to quantify the probability of generating a non anomalous value, given the approximated distribution of the time-series. Finally, if the overall score surpasses a certain threshold, then the value at the corresponding timesmtamp is classified as an anomaly.

While this threshold can be considered a hyper-parameter and an optimal value for it can be investigated by experimentation, a Peak Over Threshold (POT)[92] approach is utilized to choose the threshold automatically. In a nutshell, the POT algorithm starts by choosing an initial threshold to identify anomalies in the data, using either domain knowledge or statistical methods (the 95-percentile of the original data is used for the purposes of this work). Given this threshold, the algorithm identifies the excesses over the threshold and then models them using a Generalized Pareto Distribution (GPD). The GPD is used to model the tail behavior of the distribution, which is where the "extreme values" are expected to be located. Given the estimated parameters of the GPD, the anomaly threshold is re-set (the 95-th percentile of the estimated distribution is chosen). This process is repeated iteratively until convergence; the convergence criterion chosen here is $\delta\tau \leq 5 \cdot 10^{-3}$, where $\delta\tau$ is the difference between two consecutive threshold choices. The code implementation of the POT algorithm used in the MTAD-GATv2 model is the one written by A. Siffer[93].

3 Data Collection & Datasets

The marine industry delivers a rich playground for machine learning and data science, as it generates large amounts of data from various sources, including sensor data, navigation data, and environmental data. These data come in different forms and structures, providing diverse datasets that can be used to train and evaluate machine learning models. Among the different types of data generated by the marine industry, sensor data is particularly important for the safe and efficient operation of vessels and offshore platforms. Sensor data includes, but is not limited to, time-series data from various sources such as temperature, pressure, vibration, and other environmental sensors. These data are used to monitor the performance of equipment and detect any potential issues before they become critical.

As explained in the previous chapters, collecting such data is crucial in order to perform anomaly detection, identify potential issues and provide early warning signs before they escalate into serious problems. The purpose of the present chapter is to present the different types of sensors used to collect the data that are utilized for the training and evaluation of the anomaly detection algorithms discussed in Chapter 2. In addition, a short overview of the data collection process is given. Finally, the three different custom datasets developed for the purposes of the study are discussed in detail.

3.1 Sensors and Studied Systems

The availability of large amounts of data is ideal for the development and improvement of predictive models. However, in practical applications, it is not always possible or cost-effective to gather data from every possible source. In the context of monitoring vessel systems, it is necessary to carefully consider which systems and parameters to monitor, and how many sensors to use, in order to optimize the balance between data availability and cost. To this end, it is important to focus on systems for which there is a good understanding of the common problems that occur, and to prioritize the monitoring of key parameters that are known to have the greatest impact on the performance and reliability of these systems. By doing so, it is possible to gather the most relevant data and use them to develop effective predictive models that can help to improve the performance and safety of vessel operations. Based on this, the scope of the work presented herein covers the monitoring of four system types(10): Motor-Pump systems, Stern tubes, Pipes and Scrubber Towers.

(10) They are referred to as "system types" instead of "systems", since a single instance thereof may be present in several locations of the vessel. In other words, monitoring system A means that all instances of this system on the vessel are monitored, regardless of their location or the larger system that they are a part of.

Motor-pump systems refer to the combination of a motor and a pump that work together to transfer fluids or other materials. They are closely related and interdependent components, which is why it is necessary to study them together in order to understand their performance. Motor-pump systems are essential in a vessel and are used for a variety of purposes, including fuel transfer, bilge pumping, cooling systems, and fire-fighting systems. Their monitoring is important in order to ensure their proper functioning and detect potential issues as soon as possible, thus maintaining the safety and performance of the vessel. The fact that motor-pump systems are components of numerous other larger shipboard systems further justifies why they correspond to one of the four monitored system types. A basic outline of a motor-pump system can be seen in Fig. 3.1[(11)].

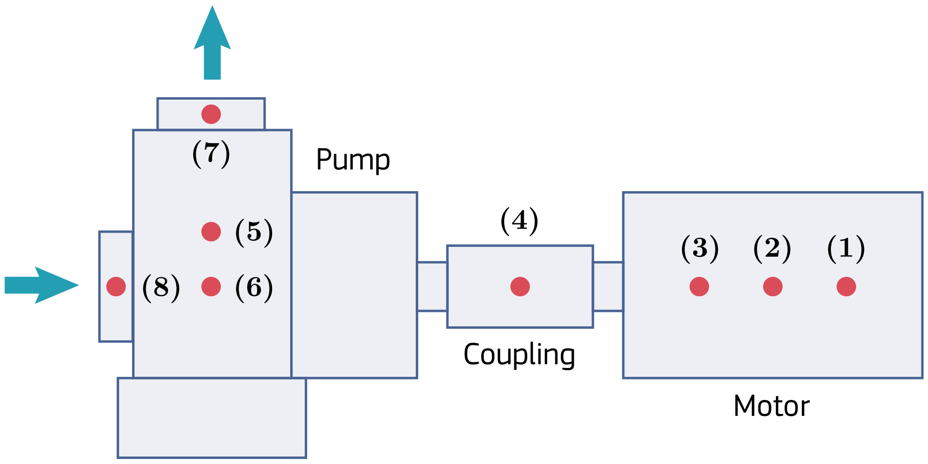


Figure 3.1: Outline of a motor-pump system connected through a coupling. Arrows typically depict the fluids' flow direction. Red points indicate the locations where sensors are placed.

The motor-pump system depicted in the figure above is connected through a coupling which serves as a flexible joint transmitting power from the motor to the pump while allowing for small misalignments between the two components. This type of system typically involves the motor and pump being mounted on separate bases, with the motor providing the necessary power to drive the pump and the coupling helping absorb vibrations and prevent wear and tear.

The red circles in the figure indicate the approximate locations where the sensors used to monitor each motor-pump system are placed. A vibration sensor is placed on the motor's shaft (1), followed by a temperature sensor (2) and a current sensor (3). Their outputs are units of displacement (specifically millimeters), degrees in the Fahrenheit scale and amperage, respectively. Vibration sensors are also placed on the coupling between the motor and the pump (4), as well as on the pump's casing (5). Additionally, a tachometer is placed near the pump's impeller (6), in order to measure its speed (in rpm). Finally, two Coriolis flow meters are placed in the pump's discharge (7) and suction sides (8), yielding measurements of fluid flow in liters per minute.

The second monitored system type is the stern tube, an outline of which can be seen in Fig. 3.2. Many vessels have a single stern tube, however several cargo ships and oil tankers may have multiple

(11) Note that Fig. 3.1, as well as other similar figures, depicts oversimplifications of the actual systems, whose architecture is obviously more complex.

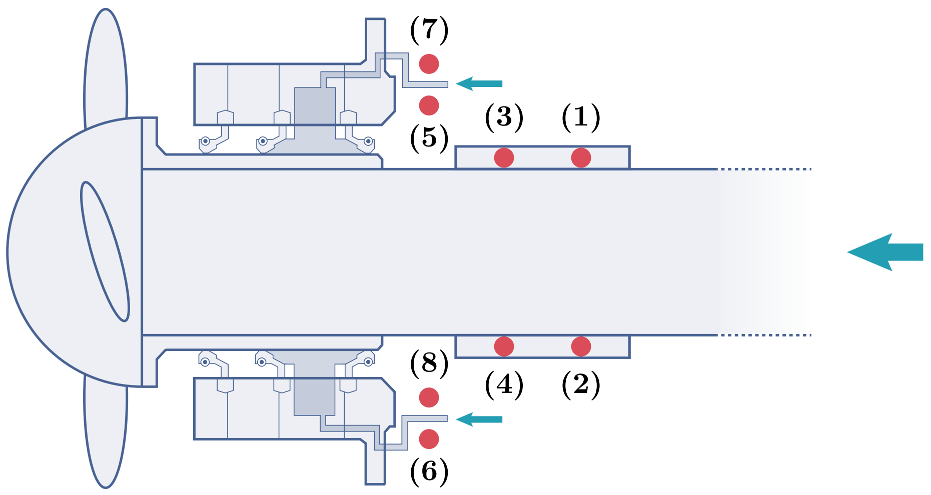


Figure 3.2: Outline of a vessel's stern tube, with emphasis on its oil supply line. Arrows depict the direction of the oil's flow. Red points indicate the locations where sensors are placed.

stern tubes to accommodate multiple propeller shafts, which can provide increased propulsion power and redundancy in case of failures. A stern tube system is a component of the propulsion system that supports the propeller shaft and provides a sealed surface for its rotation. It typically consists of one or more pairs of bearings, which are used to support the propeller shaft and reduce friction between the shaft and the stern tube, allowing the propeller to rotate freely. Additionally, a stern tube contains seals, with the aim of preventing water from entering the stern tube, providing a barrier between the sea and the ship's hull. Another very important component is the lubrication system, which helps to reduce friction and wear on the bearings and the propeller shaft, ensuring that the propeller can rotate smoothly and efficiently.

A pair of temperature sensors (1), (2) and vibration sensors (3), (4) is placed on each pair of bearings, in order to monitor the alignment of the stern tube's shaft. A rather severe issue that can occur due to several reasons, such as leaks in the shaft seals, leaks in the oil supply system, or failures of the oil return system, is water ingress into the stern tube. Measurements of oil pressure and oil flow rate can help detect water ingress into the stern tube system, through changes in these parameters' measurements. For instance, if the oil pressure decreases or the oil flow rate decreases, this could indicate that water is entering the system and displacing the oil. For this reason, a pressure transducer (5), (6) and a Coriolis flow meter (7), (8) are placed in the oil supply line of the lubrication system, in order to measure the oil's pressure (in atm) and flow (in liters per minute), respectively.

Another system type that is monitored for the purposes of the work presented herein is the common steel pipe, which is obviously a component of most shipboard systems (see Fig. 3.3). The main issue with steel pipes is the occurrence of corrosion, which can lead to severe leaks and damage if left unchecked. It is therefore crucial to monitor the width of the pipe using corrosion sensors, which can measure both the external (1) and internal (2) width of the pipe (in millimeters). It has been found that corrosion usually originates from areas of the pipe where the pipe's radius changes, due to the abrupt change of the fluid's velocity (continuity equation), which is why it suffices to place corrosion sensors only in such areas. Other important features of steel pipes that require monitoring are the fluid's flow

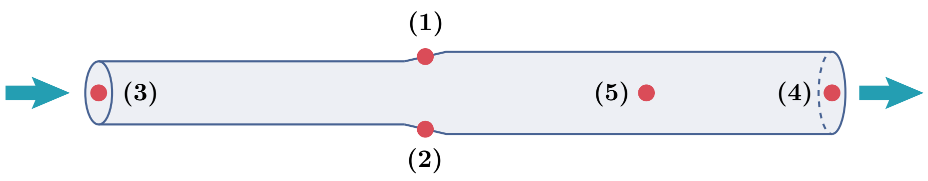


Figure 3.3: Outline of a generic steel pipe with variable radius. Arrows typically depict the fluids' flow direction. Red points indicate the locations where sensors are placed.

rate and its pH, in order to ensure that there are no leaks into or out of the pipe, in pipes carrying hazardous materials or pipes feeding other sensitive systems that could be damaged from the influx of acidic fluids. This is why two Coriolis flow meters are placed in the pipe's inlet (3) and outlet (4), and a pH-meter is placed in the interior of the pipe (5). It is worth mentioning that thermistors were also considered, due to their low cost compared to more expensive sensors used to measure temperature, such as the thermocouples used in stern tubes and motors. Nevertheless, their use was dismissed after realizing how vulnerable they are when exposed to somewhat extreme environments that are usually found in several vessel locations.

The final system type studied is the scrubber tower (commonly referred to simply as "scrubber"), an outline of which can be seen in Fig. 3.4. Scrubber towers are commonly used in maritime and industrial applications to remove pollutants from exhaust gases. They work by passing the exhaust gases through a series of chambers or scrubber elements, where the pollutants are captured and removed from the gas stream. This helps to reduce the emissions of harmful pollutants, such as sulfur oxides (SOx), nitrogen oxides (NOx) and particulate matter, which are produced by the combustion of fuels.

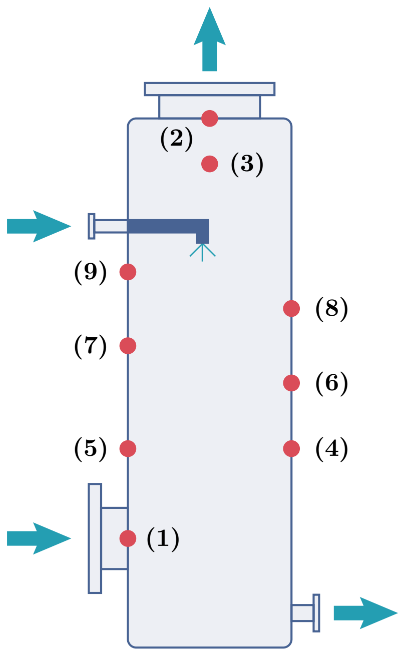


Figure 3.4: Outline of a cylindrical scrubber tower. Arrows typically depict the fluids' flow direction. Red points indicate the locations where sensors are placed.

In order to detect possible fouling or scaling in the interior of the scrubber, 2 pressure sensors are placed near its inlet (1) and outlet (2): clogging due to scaling and fouling can be detected by measuring pressure gradients. For the same reason, a conductivity sensor (yielding measurements of µS/cm) is placed in the scrubber tower's interior (3). One of the most common problems in scrubbers are thermal shocks, which occur when there is a sudden change in temperature within the scrubber system. They can be caused by a variety of factors, including changes in the temperature of the scrubbing solution or fluctuations in the temperature of the exhaust gases. The sudden expansion or contraction of the scrubber components due to temperature changes can cause stress that exceeds the material's strength, leading to cracks and other types of damage. In order to monitor the function of the scrubbers, 6 thermocouples (4)-(9) are placed on its shell: 2 closer to its bottom part and the remaining 4 in a helix around its center (Fig. 3.4 is a 2-dimensional projection, so the helix is not visible).

3.2 Data Collection

Unlike in many research cases where datasets are readily available, the collection of data in an industrial setting is a complex and challenging task that requires specialized knowledge and expertise. Real-world scenarios involve a significant amount of effort and resources to gather information, which is often collected through the deployment of properly configured data acquisition systems, ensuring accurate and reliable data collection. This section gives a short overview of this process, a schematic representation of which can be seen in Fig. 3.5.

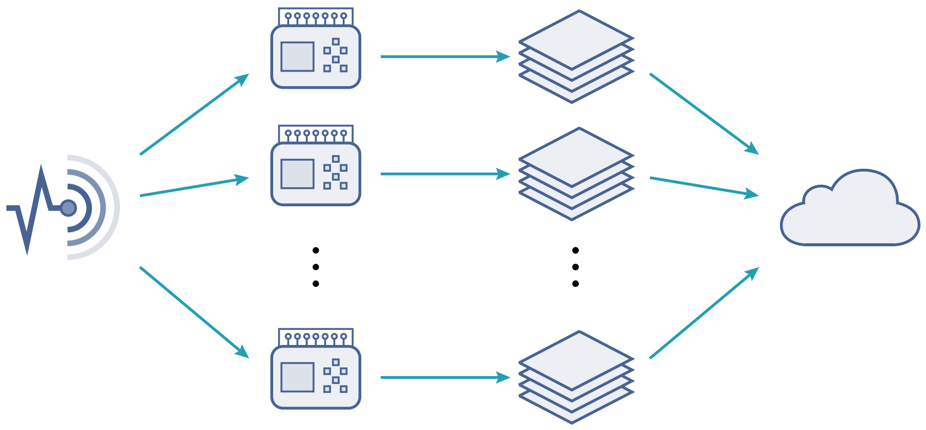


Figure 3.5: Schematic representation of the data collection process, from receiving and properly processing the sensor readings, to transmitting them to the cloud.

The first step is the ingestion of sensor readings on the egde, i.e. on the vessel. This is done by using several programmable logic controllers (PLCs), on the input/output (I/O) modules of which the sensors are physically connected. A PLC is a type of computer that is designed to control industrial processes, such as production lines or automation systems. PLCs are widely used in industrial control applications due to their ability to handle a wide range of inputs and outputs, perform complex control algorithms and communicate with other devices such as sensors and human-machine interfaces. They are also known for their ruggedness, reliability and ability to operate in harsh environments, which is why they are an ideal choice for marine operations. PLCs are programmed using specialized programming languages, and they typically include a number of built-in functions that can be used to control and monitor industrial processes. Examples of such programming languages are ladder logic[94], function block diagrams (FBD)[95], or sequential function charts (SFC)[96].

Once a network of sensors is connected to a PLC's I/O modules (typically one PLC is used per monitored system), these modules are configured to the appropriate input type, range and scaling to match the sensors being used. This involves setting up the I/O modules to use the appropriate voltage or current levels and performing all necessary signal conditioning to ensure accurate and reliable readings from the sensors. Then, the PLC is programmed to read the data from the sensors and perform any necessary processing before storing them in appropriate data structures. As far as this processing is concerned, there are two types of required operations that the PLC must perform.

To explain the first type of operation, it is important to note at this point that for many monitored systems some features are measured by two sensors instead of one. This is done to ensure availability in case of sensor failures, which is more common for some sensor types than others and hence is taken as a preventive measure for these specific sensor types (vibration measurements are the most typical example of this). In these cases, the PLC is programmed to process the input sensor data, either by logging the mean value of the two readings as a single measurement in case both sensors are operational, or by isolating the failed sensor's measurements and logging only the operational sensor's readings in case one of them has malfunctioned. As for the second type of operation, it corresponds to adding required metadata to each sensor's readings, apart from the timestamp on which they were taken and their value. These metadata include the name of the sensor that yielded the corresponding measurement(12), as well as an ON/OFF indication. This indication is necessary so that when the data are analyzed, they can be filtered depending on whether the system was operational during the measurement (ON indication) or not (OFF indication)(13). The ON/OFF indication is acquired by having a separate status input (a digital input connected to a simple switch) connected to the PLC.

In industrial control systems, the method of data transmission is a crucial consideration: one must determine whether the data will be transmitted in real-time via a streaming scenario or collected and sent in batches. Streaming involves the continuous transfer of data in real-time, while batch transmission involves the collection of data over a certain period of time, and then sending the accumulated data in one go. For vessels, which may not always have a constant internet connection, batch transmission is the preferred option. In this scenario, the PLC collects and stores the data and sends it out in batches at predetermined intervals or when an internet connection is available. This ensures that data is still captured and transmitted, even if a continuous connection is not present.

With the data properly processed and stored in batches using the discussed method, the final step is their transmission into the cloud. As discussed in more detail in Chapter 4, the ingestion of the data once they reach the cloud is performed by Microsoft Azure's IoT Hub[97] service. For this

(12) Theoretically, this is not necessary, as a PLC's channel is in a one-to-one correspondence with the feature that it logs from the connected sensor. Nonetheless, for reasons that will become clearer in Chapter 4, it is considerably easier to create data pipelines on the cloud when the data explicitly include information about the sensor's name.

(13) Shipboard systems are not always operational, however they are still monitored even when offline. For example, if specific valves are closed, there is no fluid flow from a pipe and therefore no valuable flow measurements.

reason, a secure and reliable connection is established between all PLCs and the Azure IoT Hub, using the MQTT protocol. MQTT stands for Message Queuing Telemetry Transport and it is a lightweight publish/subscribe messaging protocol that is commonly used for transmitting data from IoT devices to servers. One advantage of using MQTT is its efficiency and low overhead, making it well suited for use in resource-constrained environments such as those found in many IoT devices. Additionally, MQTT provides a flexible and scalable way to transmit data from a large number of devices to a centralized hub like Azure IoT Hub. In the case presented herein, where data is sent in batches from a PLC to Azure IoT Hub, the PLC is configured as an MQTT client and uses the MQTT protocol to publish messages containing the sensor readings and other relevant data to separate devices entities of the IoT Hub. The IoT Hub, acting as an MQTT broker, receives these messages and corresponds to the entry point of a cloud-based data pipeline.

3.3 Constructed Datasets

Closing the present chapter, an overview of the three dataset types used in this study is provided. It must be stated at this point that the term "dataset type" is used to refer to three different "families" of datasets. For example, the first "dataset type" described in what follows actually consists of four separate datasets. However, these datasets' use is different compared to the use of datasets that are discussed later, which is why they are grouped into a single dataset type.

3.3.1 Dataset Type #1: Controlled Anomaly Simulation

The first dataset type consists of four individual datasets which are called *pre-training datasets*. Each of these datasets corresponds to one monitored system from the ones discussed in Section 3.1. Their features along with their descriptions can be seen in Tables 3.1 - 3.4.

Feature	**Description**
`VibM`	vibration on motor's shaft
`TempM`	motor temperature
`CurrM`	motor current
`VibC`	vibration on coupling
`VibP`	vibration on pump's casing
`Speed`	pump impeller speed
`FlowOut`	flow from pump discharge
`FlowIn`	flow into pump suction

Table 3.1: Pre-training dataset for motor-pump systems.

Obviously, a machine learning model is best trained using historical instances of the data that it is asked to provide inference results for. When historical data are not available, a common practice is

Feature	Description
Bear1Temp1	first temperature measurement of the first bearing pair
Bear1Temp2	second temperature measurement of the first bearing pair
Bear1Vib1	first vibration measurement of the first bearing pair
Bear1Vib2	second vibration measurement of the first bearing pair
Pressure1	oil pressure in first inlet of lubrication system
Pressure2	oil pressure in second inlet of lubrication system
Flow1	oil flow in first inlet of lubrication system
Flow2	oil flow in second inlet of lubrication system

Table 3.2: Pre-training dataset for stern tube systems.

Feature	Description
ExtWidth	external measurement of pipe width
IntWidth	internal measurement of pipe width
FlowIn	fluid flow in the pipe's inlet
FlowOut	fluid flow in the pipe's outlet
pH	Measurement of fluid pH

Table 3.3: Pre-training dataset for pipe systems.

Feature	Description
PressureIn	pressure near scrubber tower's inlet
PressureOut	pressure near scrubber tower's outlet
Cond	conductivity measurement in scrubber tower's interior
TempB1	first temperature measurement near the scrubber tower's base
TempB2	second temperature measurement near the scrubber tower's base
TempC1	first temperature measurement of the helix
TempC2	second temperature measurement of the helix
TempC3	third temperature measurement of the helix
TempC4	fourth temperature measurement of the helix

Table 3.4: Pre-training dataset for scrubber tower systems.

to collect data until a significant volume is concentrated, manually label them and then use them for training. However, in an industrial setting, this approach is impractical for two reasons: firstly, it requires a significant amount of time, which is translated into loss of profit for a company's stakeholders. Secondly, and most importantly, anomalies are by definition rare events, which means that in order to collect timestamps that are labeled as anomalous, the aforementioned time is multiplied by a large factor.

To mitigate this challenge, before installing sensors on vessels and deploying machine learning models for anomaly detection, one instance of each system discussed in Section 3.1 is purchased and sensors are installed on it to collect data as explained in the previous sections. This is done in dry dock(14) facilities, thus providing a controlled environment in which the systems can operate in conditions that simulate (as best as possible) real-world vessel conditions, while also allowing engineers to induce artificial anomalies, so that labeled datasets can be constructed. It goes without saying that the data acquired in this manner are not ideal. Isolating each system from its environment artificially removes possible correlations between different systems. Additionally, stressing the systems to induce artificial anomalies in order to acquire timestamps that are labaled as anomalous has its limitations; for instance, a pipe cannot be artificially corroded in a given time window and a scrubber tower cannot suffer a thermal shock and then return to a normal operating condition without repairs. Finally, caution is required so as not to create temporal correlations that do not occur in real-world scenarios (for instance, inducing a series of anomalies once every day can lead to the creation of some sort of unrealistic periodicity). Nonetheless, pre-training models on data acquired from the exact same sensors that are installed on vessels and provide the real-time data feed for anomaly detection is significantly "better than nothing".

As far as dataset details are concerned, the systems are monitored for a total of 17 days, with a measurement granularity (frequency of sensor measurements) of 30 seconds for every feature. This leads to the creation of time-series with 48960 labeled timestamps. The following table (Table 3.5) presents the contamination factor (the percentage of anomalies) for each system's dataset.

System	**Number of induced anomalies**	**Contamination Factor**
Motor-pump system	2306	4.71%
Stern tube	1842	3.76%
Pipe	2893	5.91%
Scrubber tower	2566	5.24%

Table 3.5: Contamination factor of each multivariate time-series dataset.

Note that these contamination factors correspond to the multivariate time-series' labelling and not that of individual features. For example, in a pipe system, even if the `ExtWidth` feature's time-series appears normal in a given time window (its label is 0 in this window), if the readings for the `pH` feature's

(14) Dry docks are large structures that are used to lift vessels out of the water so that maintenance and repairs can be carried out on their hulls, propellers and other parts.

time-series correspond to anomalous events (its label is 1 in this window), then the pipe system is considered to be in an anomalous state (thus the multivariate time-series' label is 1 in this window). This example is depicted in Fig. 3.6. Of course, the labels for each feature individually are also included in the datasets, in order to allow for the evaluation of both univariate and multivariate anomaly detection algorithms. The feature-wise distribution of anomalies over each multivariate time-series can be found in Appendix C.

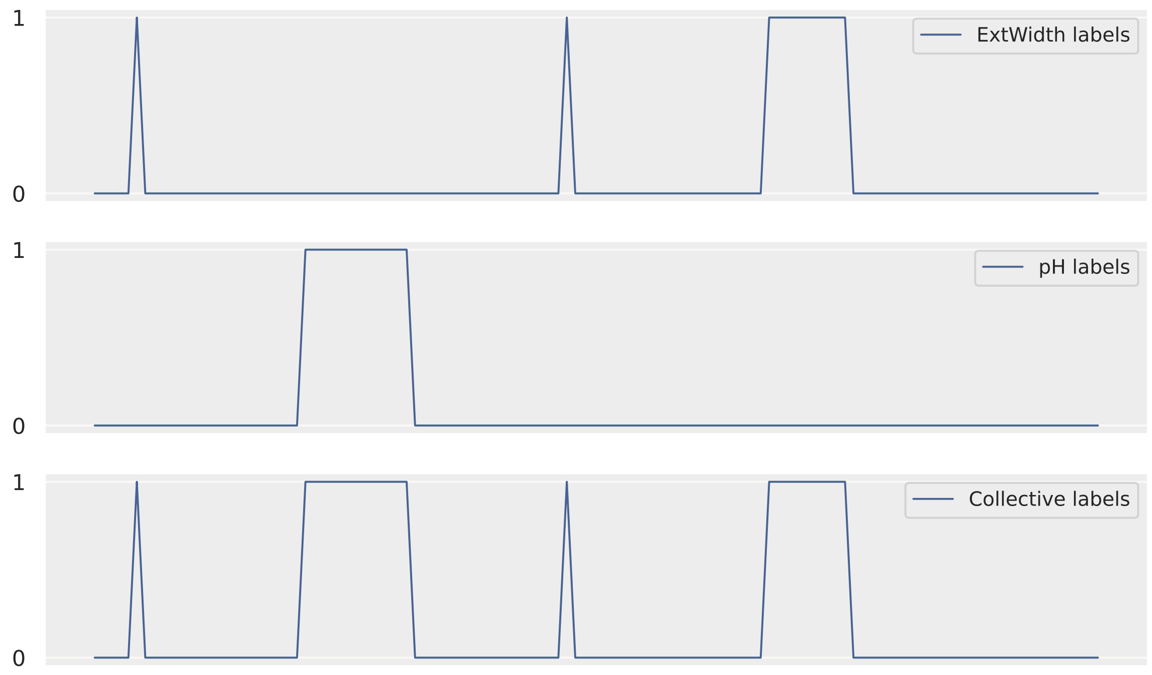


Figure 3.6: Example of how individual features' labels are used to create labels for the entire multivariate time-series corresponding to a system.

3.3.2 Dataset Type #2: Synthetically Generated Time-Series

The second dataset type comprises univariate and multivariate time-series datasets of 10000 timestamps each, which are used to evaluate the expressivity of the models trained on the previously introduced pre-training datasets. These datasets' time-series are synthetically generated by using noisy periodic functions (namely sine and cosine waves, square pulses and ECG-like pulses, with added Gaussian noise) as a base time-series model and injecting them with different types of anomalies in order to test the algorithms' performance on data that are both unseen and generated from distributions which are different compared to the ones of the training data. This is why periodic functions are used as a base model: most of the features of the monitored shipboard systems do not exhibit periodic behaviour or show patterns of periodicity. Note that extremum anomalies are included in all datasets.

To create the datasets of this type, scripts based on the GutenTAG tool of the TimeEval project[98] are developed. There are 8 different anomaly types which are injected into the base time-series: (1) amplitude and (2) frequency anomalies (which alter the wave's/pulse's amplitude and frequency, respectively), (3) extremum anomalies (which correspond to global or local minima or maxima injected

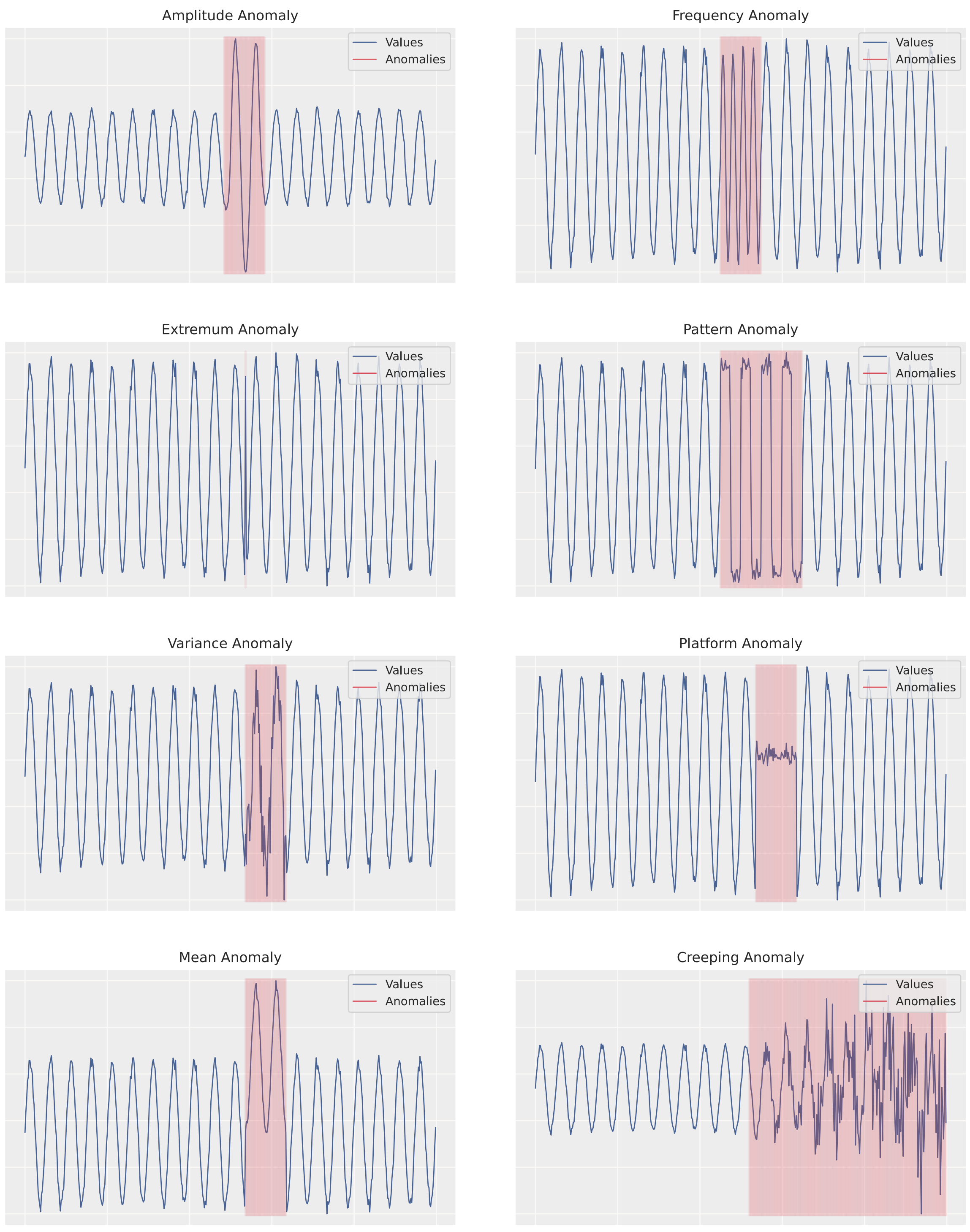


Figure 3.7: Examples of each anomaly type taken into consideration for the development of the synthetically generated time-series datasets.

in the normal time-series), (4) pattern anomalies (which inject patterns of waves/pulses that are different from the ones used in the base time-series), (5) variance anomalies (where the Gaussian noise is increased), (6) platform anomalies (which create plateaus somewhere in the time-series), (7) mean anomalies (which correspond to vertical displacements of the time-series) and (8) creeping anomalies (which introduce effects that gradually alter the base time-series' pattern over time). These different anomaly types are depicted in Fig. 3.7, where the base time-series corresponds to sine waves. Additionally, table 3.6 provides details for the different datasets constructed for this purpose, including a dataset ID, the number of its features (thus making a distinction between univariate and multivariate time-series), the periodic function used as a base model, the types of injected anomalies, as well as the contamination factor.

ID	Features	Base Model	Anomalies	Contamination Factor
U1	1	sine wave	(1), (2), (5)	5%
U2	1	cosine wave	(1), (4), (6)	5%
U3	1	square pulse	(4), (7)	5%
U4	1	ECG pulse	(6), (8)	8%
M1	8	all	(1), (2), (4), (6)	5.2%
M2	8	all	(1), (2), (4), (5)	4.5%
M3	5	all	(6), (8)	7.6%
M4	9	all	(6), (7), (8)	7.8%

Table 3.6: Datasets of synthetically generated time-series.

It is stressed at this point that these datasets are developed to test the expressivity of already trained models. As a result, the number of features for each multivariate dataset is fixed, depending on the type of model that is to be tested: datasets of 8, 8, 5 and 9 features are developed for MVAD models pre-trained on the dataset of motor-pump systems, stern tubes, pipes, and scrubber towers, respectively. Note that the granularity of data is not relevant in this case, however it is taken to be equal to 30 seconds, for uniformity reasons.

3.3.3 Dataset Type #3: Operational Vessel Data

The final dataset type consists of datasets constructed from the real-world data that are transmitted from sensors installed on operational vessels. For the purposes of the present thesis, the data of a single vessel are taken into account, however the project described herein concerns in general numerous vessels, of different types, from several companies. The studied vessel is a cruise liner with 6 diesel engines and therefore has 6 scrubber towers. Additionally, 19 different pipe systems are monitored (including the pipes that lead to the scrubber towers), as well as 28 different motor-pump systems (the vessel has more than 100). Including the single stern tube, this amounts to a total of 54 datasets, which are continuously appended with new data as they arrive in batches. By the time of writing the present thesis, each dataset contains features with more than 170000 entries, corresponding to more than 2 months of constant data feed. The sensors' measurement granularity is 10 seconds, however, as explained in the next chapter, by taking the median of 3 consecutive measurements as a single data point, the granularity effectively becomes equal to 30 seconds, as in the case of the pre-training data.

4 Cloud Data Engineering

In the field of machine learning and data science, having access to well-curated datasets is critical to the development and evaluation of models. However, in real-world applications, collecting and preparing data for analysis can present significant challenges. This is particularly true in the context of maritime operations, where the use of sensors to gather information about the health condition of vessel systems requires a robust and efficient data pipeline.

The process of managing data in a production environment is referred to as Data Engineering. It encompasses the design, development and maintenance of the necessary infrastructure and processes for collecting, transforming and preparing data for use with machine learning models or other algorithms. The challenge of Data Engineering is to efficiently handle the complexities of large and diverse datasets, making sure that the data is processed in a way that is suitable for analysis.

The use of cloud infrastructure, such as the Azure Cloud Ecosystem[99] which is used in the work presented herein, can simplify the process of Data Engineering by providing scalability and reliability for handling the complexities of data ingestion, transformation, and modeling. This allows Data Engineering to be streamlined, enabling the creation of a robust and efficient data pipeline. Presenting the followed practices and used services for the construction of this pipeline as a unified Azure solution is the aim of this chapter.

4.1 Azure Cloud Ecosystem & Solution Architecture

The Azure Cloud Ecosystem, developed by Microsoft, is a comprehensive cloud computing platform that provides organizations with scalable and reliable services for their data processing and analysis needs. With its wide range of services for data analysis and machine learning, one can build and deploy complex data-driven applications in a flexible and efficient manner.

One of the most significant benefits of the Azure Cloud Ecosystem is its scalability. Azure provides a flexible infrastructure that can accommodate changing demands, making it ideal for organizations with dynamic data processing needs. This is exactly the case with maritime operations, where the number of sensors, vessels or even client companies can change drastically in a single day. Regardless of the size of one's data processing needs, Azure provides the necessary infrastructure to support these demands. Additionally, Azure offers a range of data storage options, including structured and unstructured data, making it easy to store and access large and complex datasets, such as sensor data from vessel systems. With the ability to scale as needed, organizations can avoid the costs and limitations of traditional on-premises data centers and focus on their core business operations.

Another critical aspect of the Azure Cloud Ecosystem is its reliability. Azure provides high availability and disaster recovery features, ensuring that data is always available and protected, which are particularly important for organizations that rely on their data for critical business operations. This is exactly the case with the studied problem, where anomaly detection is performed to identify potential malfunctions and system failures or indications thereof. Azure also offers a wide range of security features, including data encryption, access control and network security, to ensure that data is protected from unauthorized access. With the assurance of data security and availability, organizations can focus on their core business operations and avoid the costs and complexities of managing their own data centers.

Its open and flexible architecture is another key feature of the Azure Cloud Ecosystem. Azure supports a wide range of programming languages and platforms, making the integration of existing systems and tools with the Azure platform easy. Additionally, Azure provides numerous Application Programming Interfaces (APIs) and Software Development Kits (SDKs), making it easy for organizations to automate their data processing and analysis tasks. In the present work, the Python SDK is utilized to create and manage a series of different services. This allows efficiently organizing all data processing and analysis workflows for maritime operations, reducing the time and resources required to manage these processes.

The Azure Cloud Ecosystem also provides several services for data analysis and machine learning, which can be used to build and deploy complex data-driven applications. Additionally, Azure provides services for data ingestion, pre-processing and modeling, making it easy to create end-to-end data pipelines for the analysis of sensor data from vessel systems, streamlining all aspects of data operations. One key feature of Azure that supports all of these functionalities is the integration of Databricks[100], a powerful and collaborative platform for big data processing and machine learning. The Databricks platform offers a user-friendly interface, collaborative features, and the ability to scale to meet dynamic demands, making it ideal when working with large and complex datasets. Azure Databricks[101] integrates with other Azure services and tools, offering organizations a seamless and efficient solution for their data processing and analysis needs. With its comprehensive capabilities and robust architecture, Azure Databricks is a valuable tool for organizations seeking to gain valuable insights into their operations and optimize their processes.

The full architecture of the solution developed in Azure for the requirements of the work presented in this thesis can be seen in Fig. 4.1. It can be divided into four main groups of resources, each serving a specific purpose in the data pipeline: *Data Ingestion* (a group handling the ingestion of the data transmitted from the vessel), *Data Storage* (a group controlling the storage of the data into different tiers), *Computations* (a group containing the services that handle the orchestration of computations with the processed data) and *Serving* (a group for the services that serve the final results). In addition to these resource groups, there is an extra group seen in Fig. 4.1, titled *Logging and Security*.

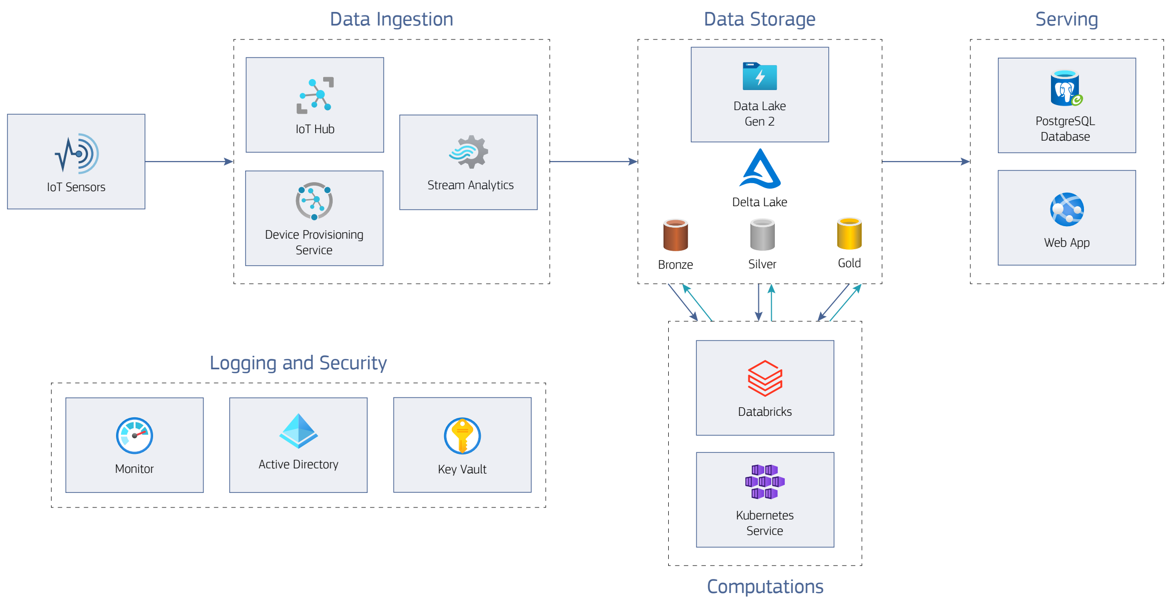


Figure 4.1: Architecture of the Azure solution.

This group includes three main Azure components that are used globally, outside the scope of individual resource groups. The first among them is Azure Monitor[102], which is a monitoring service allowing the collection, analysis and action on telemetry data received from a variety of resources and applications, in order to optimize the performance and availability of their workloads. The second is the Azure Active Directory[103], which is an access management service allowing organizations to securely manage user access and authentication to cloud-based resources and applications. The third is the Azure Key Vault[104], which allows organizations to store and manage sensitive information, such as cryptographic keys, passwords, and certificates, in a centralized and highly secure manner.

4.2 Data Ingestion

The solution's architecture is described in the order in which the data pipeline's events take place, starting from the transmitting of sensor data from the vessel all the way to serving the final results. This means that the first resource group under consideration is the one handling the ingestion of the transmitted data, which includes the Azure IoT Hub, the Azure Stream Analytics[105] and the Azure IoT Hub Device Provisioning Service[106] components.

As mentioned in the previous chapter, the Azure IoT Hub service is the entry point of the data pipeline where the batch data collected from the sensors and transmitted from the PLC devices are queued for ingestion and storing. It provides secure communication channels for devices to communicate with the cloud and also provides secure storage for device identities and credentials. This ensures that only authorized devices can communicate with the cloud, and that communications between devices and the cloud are protected from unauthorized access. Another important feature of Azure IoT Hub is its scalability, as it is designed to handle large numbers of devices, and provides features such as partitioning and load balancing to ensure that the IoT Hub can handle the scale and complexity of large IoT deployments.

For the purposes of the work presented here, a single IoT Hub component is used per company. The partition number, which determines the number of parallel processing nodes that are used to handle the workload of the IoT Hub, is set equal to 10, however it can be increased or decreased as needed to accommodate changes in the scale of the deployment. All messages received by the IoT Hub are retained for 7 days (which is also configurable) before being automatically deleted, as a backup measure in case a part relevant to the storing of the data further down the data pipeline fails. Finally, using built-in endpoints, IoT Hub's message routing feature allows all received messages to be directed to storage layers (containers inside Azure Blob Storage [107]), in a time-optimized and secure manner.

More specifically, the message routing feature ingests data, transforms them into avro(15) files and then routes them to blob storage containers to be saved temporarily (for 7 days) using the format

```
{iothub}/{partition}/{YYYY}/{MM}/{DD}/{HH}/{mm},
```

where `{iothub}` corresponds to the company's IoT Hub, `{partition}` corresponds to the partition number and `{YYYY}/{MM}/{DD}/{HH}/{mm}` is a datetime format. Unfortunately, this format used from IoT Hub's message routing options is not ideal: in order to perform analytics and train machine learning models, a single sensor's(16) data need to be isolated from data points obtained from other sensors. Additionally, a directory-like hierarchy needs to be applied, separating the time-series of different sensors from different systems, belonging to different system types and different vessels, thus optimizing the way queries can be performed on the data during pre-processing or analysis. A way to achieve this would be to include additional fields in the above format, such as metadata from the received data messages. Nonetheless, at the time of writing the present thesis, there is no way of specifying one such custom format. For this reason, the Azure Stream Analytics (ASA) service is used to reorganize the ingested data and save them in the desired hierarchical format in another container.

ASA is a fully managed, real-time data streaming service whose central building blocks are called ASA Jobs. An ASA job is a continuous real-time data processing unit that receives input data from one or more sources, performs a series of transformations and outputs results to one or more destinations. As the name suggests, it is ideal for streaming scenarios and can be used for data analysis besides data ingestion. However, in the scenario described herein, it simply acts as a helper for the data ingestion process that handles the storing of data in a custom format, while leaving all other aspects of the process to IoT Hub. While using both IoT Hub and ASA might seem redundant at first sight, it is important to note that they are complementary to each other: IoT Hub can't handle the required data formatting and ASA cannot provide the reliability, availability and safety of batch data ingestion from the PLC devices that IoT Hub provides.

(15) Avro is a binary data serialization format that is widely used in big data and data processing applications, since it provides compact, efficient and highly interoperable data serialization capabilities, as well as high-level data schema evolution metadata.

(16) In general, a sensor can provide more than one measurement for more than one feature. However, as explained in Chapter 3, each sensor deployed for this project measures only a single feature, which is why the words "sensor" and "feature" are used in this text interchangeably.

As explained in Section 3.2, each sensor reading comes with the corresponding timestamp, as well as metadata regarding the status of the sensor's system (ON or OFF) and the sensor's name. The sensor's name is a string formatted as

```
{vessel}/{location}/{system}/{feature},
```

where `{vessel}` corresponds to a specific vessel's name or ID, `{location}` corresponds to a monitored system's type (one of four system types discussed in Chapter 3), `{system}` corresponds to the ID of the specific system (for example system No. 5, out of 19 monitored pipe systems) and `{feature}` corresponds to the measured feature (one of the features shown in Tables 3.1 - 3.4). This allows one to define the container where IoT Hub redirects messages as avro files as an ASA job source; then, whenever a new file appears, or an existing file is appended in this source, the job checks each row's metadata and sends (by copying) the data point to a destination container in which data are organized using the `{vessel}/{location}/{system}/{feature}` format. This is done by writing queries in ASA's SQL-like query language.

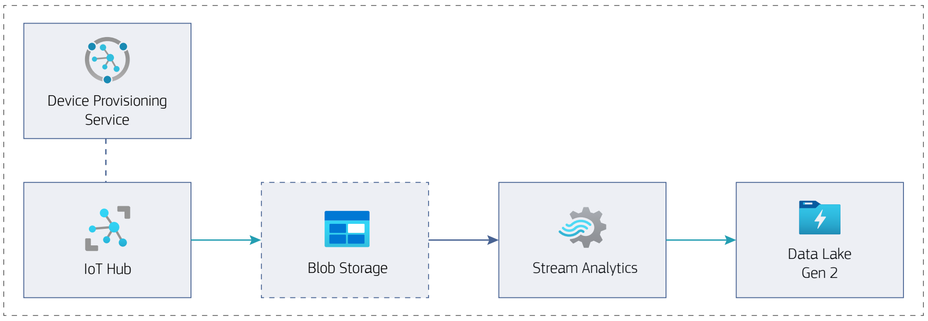


Figure 4.2: Data ingestion process using IoT Hub, DPS, ASA and Storage containers.

Closing the discussion on data ingestion, it is worth mentioning that each IoT Hub Device is in a one-to-one correspondence with a sensor of a monitored shipboard system. This means that for the single vessel that is used as a case study for the present thesis, the total number of required registered devices is given by

$$N_D = \underbrace{28 \cdot 8}_{\text{motor-pump systems}} + \overbrace{1 \cdot 8}^{\text{stern tube}} + \underbrace{19 \cdot 5}_{\text{pipes}} + \overbrace{6 \cdot 9}^{\text{scrubber towers}} = 381. \tag{4.1}$$

This indicates that the total number of devices that need to be registered in IoT Hubs is a number of $\mathcal{O}\left(10^2\right)$, multiplied by the total number of vessels per company and the total number of companies. It becomes evident that a scalable solution for registering large numbers of devices with IoT Hubs is required and this is where the IoT Hub Device Provisioning Service (DPS) enters the stage. The DPS acts as an intermediary between the devices and an IoT Hub and automates the process of registering devices with IoT Hub, ensuring security and consistency and eliminating the need for manual configuration of each device. Fig. 4.2 provides a synopsis of the described ingestion process' flow.

4.3 Delta Lake & Medallion Architecture

As discussed, the destination of the ASA job that transforms and arranges the data in a suitable format is a data storage container with hierarchical structure, which is part of a wider storage service known as Azure Data Lake Storage Gen 2[108] (ADLS2). ADLS2 is a cloud-based data lake that provides a comprehensive solution for managing big data, designed to handle both structured and unstructured formats. In general, data lakes are an evolution of traditional databases which provide a more flexible, scalable and reliable way of managing big data. Unlike databases, which are designed to handle structured data, data lakes allow organizations to store and process both structured and unstructured data in their raw form. They also provide a cost-effective solution for big data management, as they are built on a shared infrastructure and can be scaled as needed.

While data lakes have revolutionized the way organizations store and manage big data, they are not without their challenges: slow and complex data ingestion processes, lack of data governance and data management capabilities, difficulty in ensuring data reliability and consistency, to name but a few. These issues limit the full potential of data lakes and make it arduous for organizations to derive meaningful insights from their data. To address these challenges, Delta Lake[109], a new generation of data lake technology, offers a more sophisticated solution and provides a more streamlined approach to data management and processing.

Azure Delta Lake is an open-source storage format that provides ACID transactions, data versioning and schema enforcement to data lakes, built on top of ADLS2. Through its support for ACID transactions, it ensures that data is always consistent and accurate. This is achieved by implementing a transaction log that keeps track of all changes made to the data, allowing for undo and redo operations. This makes Delta Lake particularly useful in cases where data is being written and read concurrently, as is the case with the previously discussed ingestion process, ensuring that data remains consistent even in the face of errors or failures. As far as versioning is concerned, Delta Lake makes it easier to revert to previous versions of the data if necessary and enables users to track the evolution of the data over time. This feature is particularly useful in cases where data is being updated frequently, as it provides a way to keep track of changes and ensure that the data remains accurate. Finally, with schema enforcement, which helps to ensure that the data remains consistent and conforms to a specific format, data quality issues that can arise from inconsistent data formats are prevented. In addition to these features, Delta Lake also provides a number of performance optimizations that make it faster and more efficient than traditional data lake technologies. For example, Delta Lake supports columnar storage formats, such as Parquet, which enable faster and more efficient processing of big data.

The Delta Lake platform provides a robust and scalable solution for data management in the modern data landscape. However, simply storing data in a Delta Lake does not guarantee its quality or reliability. For the purposes of the work presented herein, the medallion architecture is adopted on top of the Delta Lake platform. This is a three-layer approach to data storage that takes into account the varying quality of the data being stored. The medallion architecture provides a flexible and scalable solution for managing data throughout their lifecycle, from raw and unprocessed data to highly curated and reliable information (see Fig. 4.3).

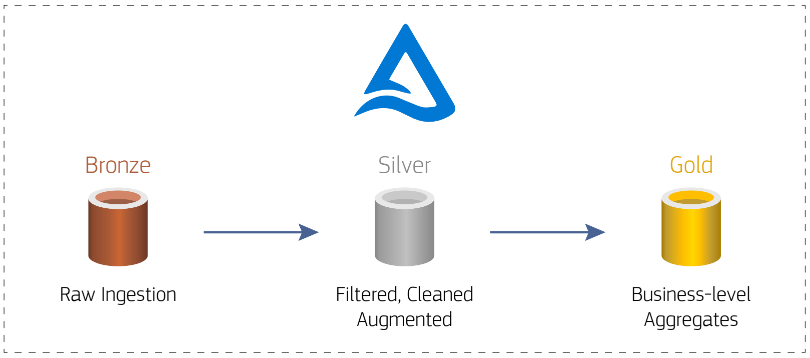


Figure 4.3: The three-layer medallion architecture built on the Delta Lake platform.

The first data layer of this architecture is the bronze tier, which is designed to hold the raw, unprocessed data that is ingested using the process described in the previous section. This layer is used for storing large amounts of data in a cost-effective manner, with no pre-processing and no quality checks. Its primary purpose is to provide a centralized repository for all ingested data, with practically infinite retention period (be reminded that the data ingested into the Blob Storage using IoT Hub remain there only for a limited number of days as a safety measure, so access in historical raw data is only possible through this bronze layer).

The second layer is the silver tier, which corresponds to a curated data layer used for storing data that has been processed and transformed to meet the quality standards required for use in analytics and machine learning applications. Essentially, this layer holds the data that have been transformed from their raw form into a more usable format. For the purposes of the work presented in this thesis, each sensor's newly ingested raw data are aggregated by selecting the median of every three sensor measurements to be used as a single data point. This is how the granularity of the real-time datasets is artificially set equal to 30 seconds, while the granularity of the sensors' measurements is 10 seconds. Additionally, the raw data are also pre-processed for the purposes of machine learning (only min-max normalization is applied so far, which is discussed in the next section). The processed data are then appended in dataframe-like structures supported by the Delta Lake format, with every sensor's data corresponding to a single dataframe.

The third and final layer of the medallion architecture is the gold tier and it is used for storing the output data of machine learning algorithms and in general data that are to be used for reports or to be served in applications and dashboards. This layer provides fast and efficient access to the data stored in it, which are stored in a format that is optimal depending on their use case.

4.4 Computations & Serving

The medallion architecture discussed above describes how the different tiers of data are organized and stored based on their "quality", however it is also important to discuss how each layer is populated and how the data that populate it are curated or generated. Azure Databricks is the service that takes care of most of this orchestration, thanks to its wide array of tools for processing and transforming data, including Spark for big data processing and an integrated Delta Lake platform.

As far as the bronze tier is concerned, since it is populated by ASA using the process described in Section 4.2, Databricks is used only to read the ingested data and process them before saving them to the silver tier layer. Specifically, using the Spark engine, Databricks performs the aggregation required to reduce the granularity of the data (taking the median of every 3 values as a data point). Then, the Spark engine also takes care of additional transformations, namely a min-max normalization of the data. Min-max normalization, also known as feature scaling, performs a linear transformation $x \to \tilde{x}$ on the original data in order to create scaled data in the range (0, 1). The transformation is given by

$$\tilde{x}^{(i,t)} = \frac{x^{(i,t)} - x^{(i)}_{\text{min}}}{x^{(i)}_{\text{max}} - x^{(i)}_{\text{min}}}, \tag{4.2}$$

where the i and t superscripts correspond to the i-th feature and t-th time step, respectively, and $x^{(i)}_{\text{min}}/x^{(i)}_{\text{max}}$ are the i-th feature's minimum and maximum values, respectively. These minimum and maximum values are obtained from the pre-training datasets, as the normalization performed on the real-time data must be identical to the one performed for the pre-training ones. Finally, using the Delta Lake format, Databricks appends the transformed data to existing dataframes, with each dataframe corresponding to an individual sensor (or, equivalently, feature).

When it comes to the gold tier, it stores data that are generated using the deployed machine learning models, which perform inference on the silver tier's pre-processed data. While Databricks provides a comprehensive and integrated platform for machine learning development and deployment, it is not always the best fit for every use case; in some cases, it may be necessary to leverage other technologies. In the present case, the combination of Docker[110], Kubernetes[111] and FastAPI[112] is utilized to carry out the machine learning operations and populate the gold tier.

Docker allows the packaging of the pre-trained machine learning models and all relevant dependencies into a container image, making it easy to distribute and run the models in a consistent, reproducible environment. Kubernetes, which is provided as an Azure service, provisions a platform for deploying and managing such containers in a scalable and reliable manner. This is crucial in the present scenario, where a single container image corresponding to the pre-trained model for a specific shipboard system needs to be distributed to several containers, so that each machine learning model instance can be fine-tuned and used to monitor all different instances of the same shipboard system. This fine-tuning, which is discussed in detail in Chapter 5, along with the inference and possible evaluation of the models is achieved by creating task-specific endpoints, which can be called from other applications (in this case, Databricks). This is where FastAPI is utilized, as it is a web framework that provides a simple and efficient way to serve machine learning models as web services.

All of the steps described in the present section are automated and orchestrated using Databricks Jobs, a feature of Databricks that enables the scheduling, running and monitoring of the execution of long-running, complex, batch-oriented workloads. Jobs can be triggered by time, event, or API and can be scheduled to run at specific intervals, such as daily or weekly. In the case of the data pipeline discussed herein, the first trigger is activated when ASA ingests new data files into the bronze tier. All the other steps, from data processing to machine learning operations, are triggered and executed as a series of orchestrated jobs, with each job being triggered only after the successful completion of the previous one. This not only streamlines the process, but also ensures that each step is executed in the correct order, reducing the risk of errors and increasing overall efficiency.

Before closing the present chapter, it is worth mentioning how the results that are stored in the gold tier are served. The final job in the aforementioned pipeline of Databricks Jobs concerns serving the contents of the gold tier in a PostgreSQL[113] database, also hosted on Azure. PostgreSQL is an open-source relational database management system known for its strong reliability and advanced features, such as data integrity, indexing and query optimization. This database is connected to the back-end of a dashboard web application, which offers visualizations of the real-time data feed (whenever new batches are ingested), as well as the inference results of the machine learning models. One could think of this PostgreSQL database as the database duplicate of the gold tier layer of the ADLS2 and therefore question its necessity in the overall solution architecture. Nonetheless, it is important to note two key points that render both the gold tier and the database important. Firstly, isolating different parts of the whole project is considered a good practice, not only for debugging, but also for security reasons. As a result, using both the gold layer and the SQL database ensures that the web application is kept out of immediate connection to the data pipeline. Secondly, and most importantly, the web application does not offer the visualization of historical data that go arbitrarily back in time, but rather has a threshold of 3 months. This means that the SQL database does not keep all of the contents of the gold tier layer, but rather only the values with timestamps that belong in this time range. This makes queries on the web application significantly faster, as the number of data points that are considered during lookup are orders of magnitude higher in the data lake, compared to the database.

5 Implementation & Results

Having established the theoretical background and the datasets used for training and evaluating all models, and also having presented the data engineering process, this final part of the present thesis discusses the implementation of the anomaly detection models and their results on all three datasets.

5.1 Models Implementation

As far as the implementation of the SR-CNN model is concerned, for the extraction of saliency maps the values $q = 3$ and $z = 21$ are chosen for the hyper-parameters corresponding to the filters $\mathbf{h}_q$ and $\mathbf{h}_z$, which are related to the averaged spectrum and averaged saliency map, respectively. These are also the values suggested by the authors in the original SR-CNN paper[26]. As for the CNN, it is implemented in Python using the PyTorch[114] library. The synthetic dataset generated for the training of the CNN is constructed by scanning the original time-series in a sliding window with size $w = 400$, a choice different compared to the one in the original paper. The scanning step is taken equal to $\beta = 100$, so that the sliding windows are overlapping. As far as the κ-parameter is concerned, it is not considered universal for all instances of the SR-CNN model; depending on the dataset where each model instance is evaluated, κ is chosen so that $\kappa/2w$ is approximately equal to the dataset's contamination factor(17).

The CNN's architecture comprises three 1-dimensional convolutional layers with zero padding, unit kernel(18) and unit stride and output channels of dimension w, $2w$ and $4w$, in this order. The convolutional layers' outputs are then flattened and passed over two fully connected layers with outputs $8w$ and w, in this order. Each layer is followed by a ReLU nonlinearity, except for the final layer which is followed by a logistic sigmoid activation, since the final outputs need to be interpreted as probabilities (or scores). Additionally, batch normalization is used after each convolution and before ReLU, as it helps alleviate the problem of the internal covariate shift[115].

Finally, regarding the network's training, a constant number of 30 epochs is chosen. An early stopping mechanism[116] could be adopted, however experiments showed that imposing L2 regularization

(17) This is because in a window with size w where up to κ points are chosen as anomalies, the mean number of chosen anomalies is $\kappa/2$. For instance, for $w = 400$, in a dataset with a contamination factor of 3%, κ is taken to be approximately equal to 24.

(18) A univariate time-series can either be seen as a T-dimensional sequence of a single channel, or as a 1-dimensional item of T channels. The latter viewpoint is adopted in this case.

is more than sufficient to avoid overfitting. The loss function is obviously binary cross entropy and it is optimized using PyTorch's Stochastic Gradient Descent (SGD) optimizer. The learning rate, $lr\left(e\right)$, is adjustable per epoch e, following the rule

$$lr\left(e\right) = lr_0 \cdot \left(\frac{1}{2}\right)^{\lfloor e/10 \rfloor}, \tag{5.1}$$

where lr_0 is the initial value of the learning rate and $\lfloor a/b \rfloor$ corresponds to integer division of a by b. For all model instances, lr_0 is taken equal to 10^{-3}.

Moving on to the MTAD-GATv2 model for MVAD tasks, it is also implemented in Python using the PyTorch library. A significant portion of the code is based on the implementation by ML4ITS[117], even though their model is not an accurate representation of Microsoft's MTAD-GAT[91] (which is what significantly differentiates it from the work presented here). As in the case of the SR-CNN model, a sliding window approach is followed here as well, with window size $\Delta t = 400$.

For the first part of the model's architecture, the output channel's dimension for the 1-dimensional convolutional layer is set equal to its input channel's dimension (i.e. the number of features, k) and a kernel of size 10 is chosen for the convolution operation. A ReLU nonlinearity is also added as the activation of this single convolutional layer. Regarding the GATv2 layers, their details are discussed thoroughly in Section 2.2.4, however it is worth mentioning that a dropout layer with dropout probability equal to 0.25 is also added in the neural network that calculates the attention vectors, for regularization purposes. As is common practice, the LeakyReLU nonlinearity's slope is taken equal to 0.2. The concatenated outputs of the two GATv2 layers and the 1-dimensional convolutional layer are fed to a single GRU layer with hidden dimension $d = 150$; the code allows for the addition of multiple GRU units (and the introduction of in-between dropout layers for regularization), however experiments showed that this does not lead to noticeable accuracy improvements.

For the second part of the MTAD-GATv2 model's architecture, the MLP corresponding to the forecasting model is chosen to include three hidden layers of sizes d, $2d$ and d, in this order. The input layer and all hidden layers are followed by dropout layers with dropout probability equal to 0.25, which is in turn followed by a ReLU activation. As for the VAE used as the reconstruction model, the prior's, as well as the encoder's and decoder's distributions are taken to be Gaussians with diagonal covariance matrices (see Eqs. (2.29), (2.30) and (2.42) from Chapter 2). In fact, following the original MTAD-GAT paper, the standard Gaussian distribution is chosen for the prior and the decoder. The deep learning models chosen for the encoder's generative task and the decoder's inference task are CNNs. More specifically, the encoder consists of 3 sequential 1-dimensional convolutional layers with ReLU activations, output channel dimensions d, $2d$ and $4d$, in this order, kernels of size 4, strides of 2 and unitary padding. The convolutional layers' outputs are then flattened and duplicated, since the encoder's multivariate Gaussian distribution with a diagonal covariance matrix is parametrized by one L-dimensional vector of means and one L-dimensional vector of variances. Consequently, each duplicate passes through two fully connected layers with hidden dimension d and bottleneck dimension $L = 10$. As for the decoder, its architecture is the inverse[19] of the encoder's architecture, with the exception of its final layer: since the VAE's objective is to reconstruct the original time-series' feature vector, an additional convolutional layer is applied, with output channel dimension equal to k and same kernel, stride and padding sizes.

As far as the MTAD-GATv2 model's training is concerned, the loss function is the sum of the forecasting and reconstruction error given by Eq. (2.47). The loss function chosen for the forecasting model is the MSE, while the loss function corresponding to the VAE's reconstruction error is the opposite of the approximated ELBO of Eq. (2.43) (the ELBO needs to be maximized, so its opposite needs to be minimized). This expression is greatly simplified in the case of Gaussian priors, encoders and decoders, especially given that the decoder's distribution is the standard Gaussian (see Appendix B for detailed calculations). For a given (measured) feature vector $\mathbf{x} = \{x_i\}_{i=1}^{k}$ at time step t, if the forecasting output is denoted by $\hat{\mathbf{x}} = \{\hat{x}_i\}_{i=1}^{k}$ and the reconstruction output is denoted by $\mathbf{r} = \{r_i\}_{i=1}^{k}$, then the overall loss function at time t can be written as

$$\mathcal{L}(t) = \underbrace{\sum_{i=1}^{k} (x_i - \hat{x}_i)^2}_{\mathcal{L}_f(t)} + \underbrace{\frac{1}{2}\sum_{i=1}^{k} (x_i - r_i)^2 + \frac{1}{2}\sum_{i=1}^{L} \left(\sigma_{\phi,i}^2 + \mu_{\phi,i}^2\right) - \frac{1}{2}\sum_{i=1}^{L} \log \sigma_{\phi,i}^2}_{\mathcal{L}_r(t)}, \tag{5.2}$$

where $\mu_{\phi,i}$ and $\sigma_{\phi,i}^2$ are the i-th elements of the L-dimensional mean and variance vectors, respectively. The optimizer used to minimize this loss function is SGD, with an adjustable learning rate that follows Eq. (5.1), with $lr_0 = 10^{-3}$. All model instances are trained using an early stopping mechanism, with patience threshold equal to 14.

A final point of discussion concerns the MTAD-GATv2 model's inference, and specifically the γ hyper-parameter. As discussed in the original MTAD-GAT paper, if γ is set between 0.4 and 1.0, no significant differences in predictive power are observed for the trained models on Microsoft's TSA dataset[91]. Without thorough investigation, but by performing a limited number of experiments, this is also confirmed in the case of all 4 pre-training datasets, thus γ is set equal to 0.8 (as in the original paper) for all MTAD-GATv2 model instances.

5.2 Pre-Training & Evaluation

Following the implementation, the first part of the experimental procedure concerns the pre-training of the MTAD-GATv2 models using the pre-training dataset for each monitored system, as well as their evaluation on a subset of data reserved for this purpose. As far as the SR-CNN model is concerned, since it is trained in an unsupervised manner, the pre-training dataset is used in its entirety for evaluation.

[19] For the convolutional layers of the PyTorch implementation, this means that if the encoder's layers are `Conv1d` objects, then the decoder's layers are `ConvTranspose1d` objects where the `in_channels` of the one corresponds to the `out_channels` of the other and vice-versa.

The formalism used herein is that a dataset's samples classified as non anomalous (labeled with 0) or anomalous (labeled with 1) by an anomaly detection model are split into four categories: 00 (samples that are not anomalous and are classified as such), 01 (samples that are not anomalous but are classified as anomalous), 10 (samples that are anomalous but are classified as non anomalous) and 11 (samples that are anomalous and are classified as such). Based on this, a confusion matrix can be assigned to each model's predictions:

$$\mathrm{CF} = \begin{pmatrix} \mathtt{00} & \mathtt{01} \\ \mathtt{10} & \mathtt{11} \end{pmatrix} \tag{5.3}$$

Given this confusion matrix, a number of different evaluation metrics can be defined:

- Accuracy: the proportion of correctly classified samples among all samples, i.e.
$$\text{Accuracy} = \frac{\mathtt{00} + \mathtt{11}}{\mathtt{00} + \mathtt{01} + \mathtt{10} + \mathtt{11}} \tag{5.4}$$
- Recall: the proportion of correctly predicted anomalies among all actual anomalies, i.e.
$$\text{Recall} = \frac{\mathtt{11}}{\mathtt{10} + \mathtt{11}} \tag{5.5}$$
- Precision: the proportion of correctly predicted anomalies among all predicted anomalies, i.e.
$$\text{Precision} = \frac{\mathtt{11}}{\mathtt{01} + \mathtt{11}} \tag{5.6}$$
- F1-Score: the harmonic mean of Recall and Accuracy, i.e.
$$\text{F1-Score} = 2 \cdot \frac{\mathtt{11}}{\mathtt{01} + \mathtt{10} + 2 \cdot \mathtt{11}} \tag{5.7}$$

Among these metrics, accuracy is the least reliable one, due to how highly imbalanced the utilized datasets are: by definition, anomaly detection concerns rare events, therefore non anomalous data are much more common compared to anomalous events. As a result, a model that classifies all received samples as non anomalous is expected to have satisfactory accuracy, while in practice it is the worst model imaginable for the task[(20)]. For this reason, in what follows, emphasis is given on Recall, Precision and especially F1-Score.

Another metric that is utilized for the evaluation of the MTAD-GATv2 model instances is the Receiver Operating Characteristic (ROC) curve, which depicts a model's predictions' true positive rate (sensitivity) against the corresponding false positive rate (1 - specificity) at different threshold settings (in MTAD-GATv2 the threshold is automatically assigned using the POT algorithm, so to extract the ROC curve this threshold is varied manually). A ROC curve provides a way to evaluate and compare the performance of different model instances by considering the trade-off between sensitivity and specificity. The area under the ROC curve (AUC) is a commonly used metric for evaluating the performance of models such as anomaly detectors. A perfect model's ROC curve has an AUC of 1, while the random classifier's ROC curve has an AUC of 0.5.

[(20)] For instance, in a dataset with a contamination factor of 5%, one such model's accuracy would be 95%, however the model would fail to identify any anomalies.

5.2.1 Evaluation of the SR-CNN Models

As far as the SR-CNN model's instances are concerned, the pre-training datasets are used only for a preliminary evaluation thereof. For this purpose, separate instances of the model are self-trained as explained in Section 2.1.2 using the entirety of the available time-series' timestamps for each feature individually. Figs. 5.1 - 5.4 depict the confusion matrices corresponding to the predictions for each

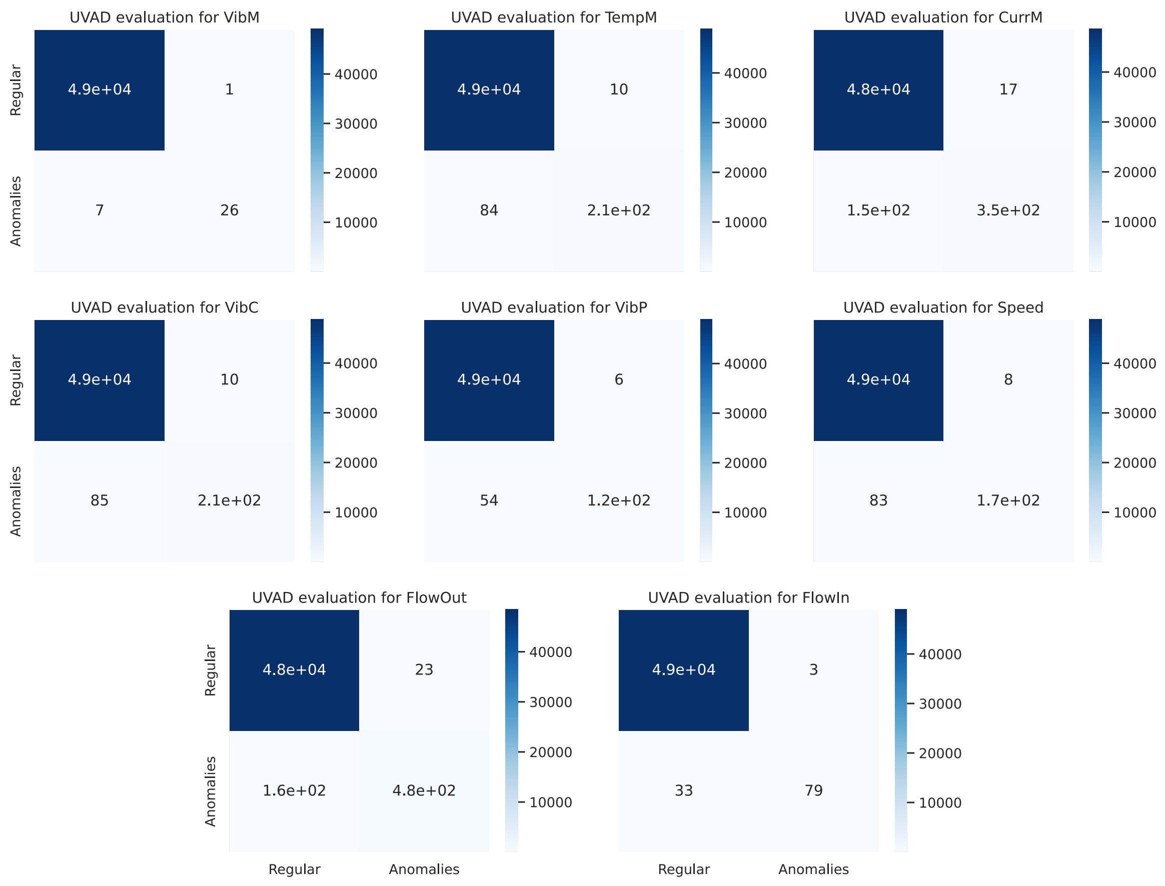


Figure 5.1: Feature-wise confusion matrices for the motor-pump systems with the SR-CNN model.

Feature	Accuracy	Recall	Precision	F1-Score
VibM	99.98%	78.79%	96.30%	86.67%
TempM	99.81%	71.23%	95.41%	81.57%
CurrM	99.66%	70.04%	95.41%	80.78%
VibC	99.81%	71.38%	95.50%	81.70%
VibP	99.88%	69.66%	95.38%	80.52%
Speed	99.81%	66.80%	95.43%	78.59%
FlowOut	99.62%	74.22%	95.38%	83.48%
FlowIn	99.93%	70.54%	96.34%	81.44%

Table 5.1: Feature-wise evaluation metrics for the motor-pump systems with the SR-CNN model.

feature, while Tables 5.1 - 5.4 provide the values for all evaluation metrics[21]. As expected, the accuracy metric provides no valuable information regarding the UVAD models' performance, with all instances achieving higher than 99% results. Additionally, the precision of all instances is higher than

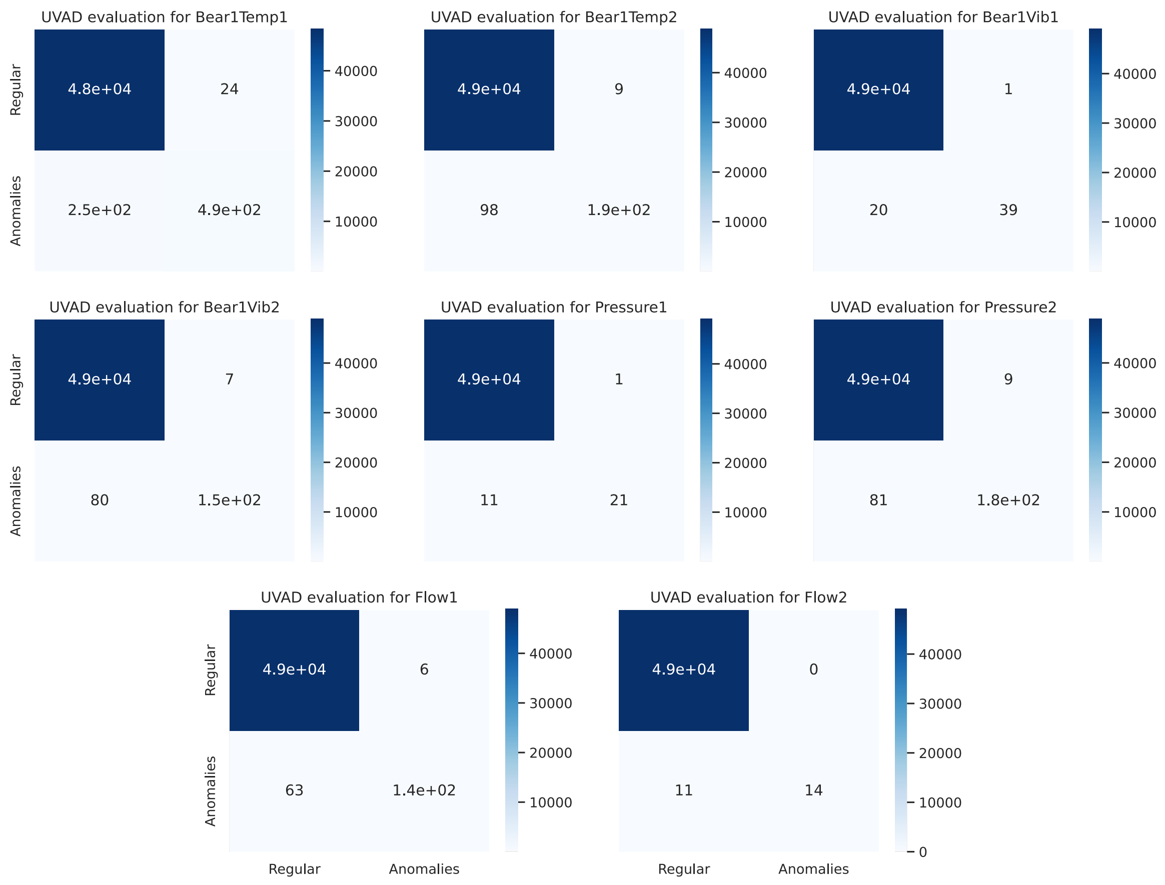


Figure 5.2: Feature-wise confusion matrices for the stern tube with the SR-CNN model.

Feature	Accuracy	Recall	Precision	F1-Score
Bear1Temp1	99.45%	66.58%	95.35%	78.41%
Bear1Temp2	99.78%	65.97%	95.48%	78.03%
Bear1Vib1	99.96%	66.10%	97.50%	78.79%
Bear1Vib2	99.82%	65.67%	95.62%	77.86%
Pressure1	99.98%	65.62%	95.45%	77.78%
Pressure2	99.82%	69.32%	95.31%	80.26%
Flow1	99.86%	68.81%	95.86%	80.12%
Flow2	99.98%	56.00%	100.00%	71.79%

Table 5.2: Feature-wise evaluation metrics for the stern tube with the SR-CNN model.

[21] Note that the inclusion of the evaluation metrics is technically redundant, as the confusion matrices themselves offer all the information required for their extraction.

95%, indicating that the models do not tend to misdiagnose anomalies. This renders the recall and F1-Score as the only suitable metrics for evaluation. More specifically, the relatively low recall scores point to the UVAD models' difficulty in recognizing all of the datasets' anomalies. As for the F1-Score, its mean value for the motor-pump systems' pre-training dataset is 81.84%, its mean value for the stern tube's pre-training dataset is 77.88%, its mean value for the pipes' pre-training dataset is 81.37% and its mean value for the scrubber towers' pre-training dataset is 80.51%.

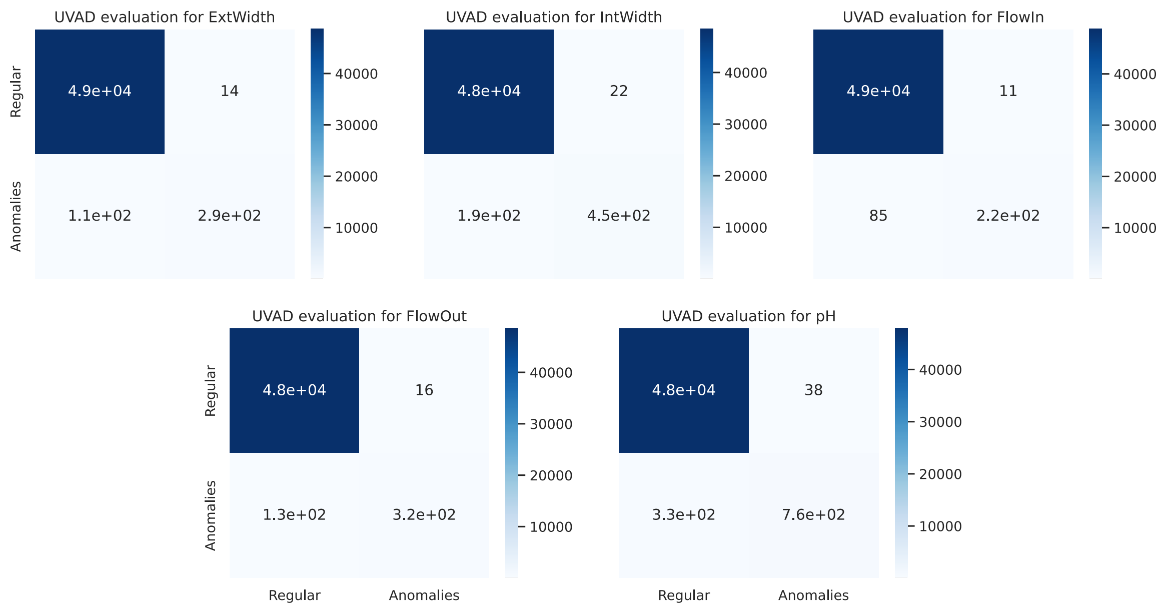


Figure 5.3: Feature-wise confusion matrices for the pipe systems with the SR-CNN model.

Feature	Accuracy	Recall	Precision	F1-Score
`ExtWidth`	99.74%	71.75%	95.35%	81.88%
`IntWidth`	99.58%	70.66%	95.32%	81.16%
`FlowIn`	99.80%	72.31%	95.28%	82.22%
`FlowOut`	99.70%	70.77%	95.27%	81.21%
`pH`	99.24%	69.55%	95.26%	80.40%

Table 5.3: Feature-wise evaluation metrics for the pipe systems with the SR-CNN model.

At first sight, these results might seem unimpressive and can be attributed to the UVAD models' inherent inability in identifying important intra-feature correlations, which are strongly present in this work's datasets. Nevertheless, it must be stressed that, unlike in most other works regarding anomaly detection, the results presented herein are obtained via a one-to-one evaluation of predicted labels, as is common in the evaluation of classification models. In particular, anomaly detection algorithms are usually evaluated using a neighbourhood evaluation scheme, where an anomaly prediction is rendered as correct even when the timestamp on which it is predicted is not the exact timestamp, t, when the anomaly occured, but lies within a neighbourhood δt around t. Another approach is to use a sliding window scheme (different from the one used for training), where the evaluation considers a sequence of consecutive time intervals and calculates the proportion of correctly predicted anomalies within each interval. Here, anomalies are considered correctly predicted only when the prediction is made on the exact timestamp with an anomaly label, which is why the recall score appears so low. This is

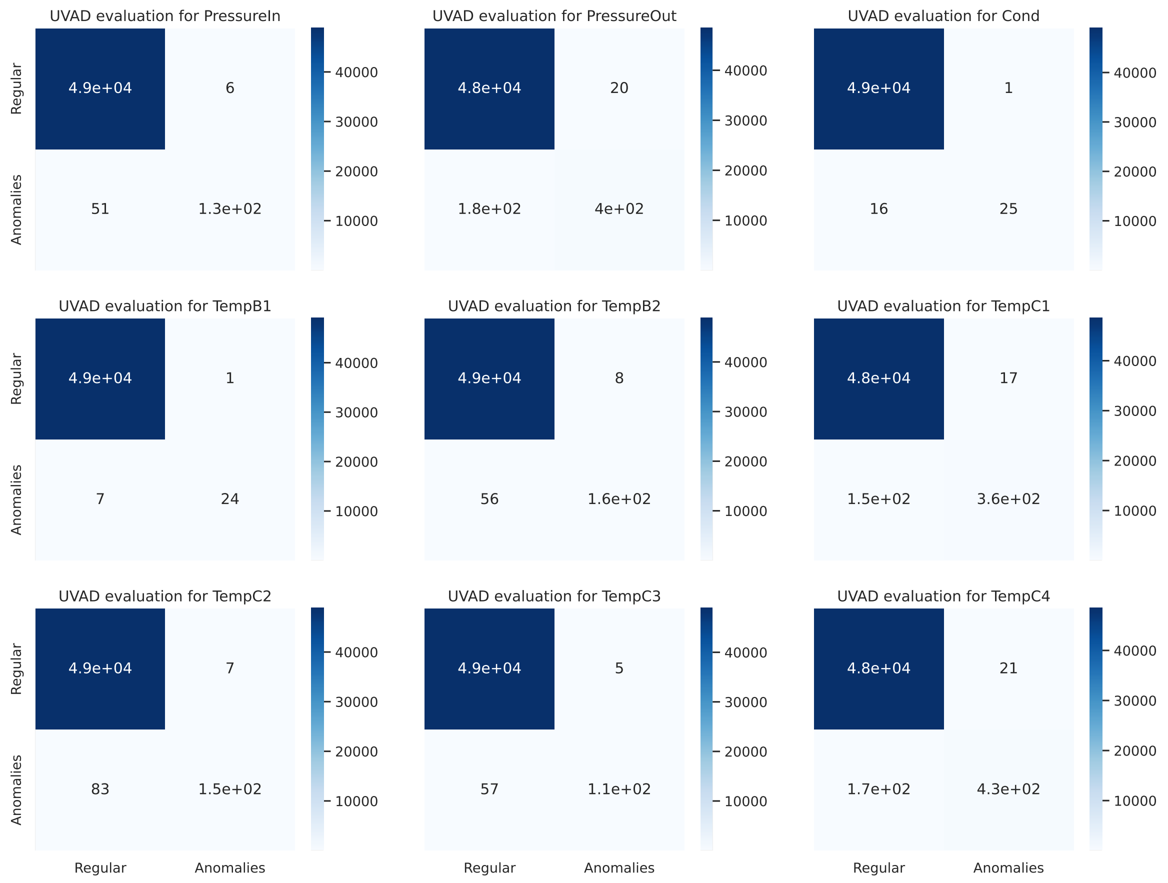


Figure 5.4: Feature-wise confusion matrices for the scrubber towers with the SR-CNN model.

Feature	Accuracy	Recall	Precision	F1-Score
`PressureIn`	99.88%	72.13%	95.65%	82.24%
`PressureOut`	99.60%	69.72%	95.27%	80.52%
`Cond`	99.97%	60.98%	96.15%	74.63%
`TempB1`	99.98%	77.42%	96.00%	85.71%
`TempB2`	99.87%	74.07%	95.24%	83.33%
`TempC1`	99.66%	70.55%	95.45%	81.14%
`TempC2`	99.82%	64.68%	95.60%	77.16%
`TempC3`	99.87%	66.07%	95.69%	78.17%
`TempC4`	99.60%	71.38%	95.38%	81.66%

Table 5.4: Feature-wise evaluation metrics for the scrubber towers with the SR-CNN model.

done for two main reasons; firstly, it is important to evaluate the models as if there is no degree of uncertainty regarding the anomalies' timestamps, so that a lower threshold of their performance can be known before their deployment - in other words, "it's better to be safe than sorry". Secondly, the datasets include a series of point anomalies (mainly extrema), therefore it is important that the models recognize them as such exactly when they occur.

5.2.2 Pre-Training of the MTAD-GATv2 Models

Moving on to the MTAD-GATv2 model's instances, which is the main reason the pre-training datasets were developed, the results for their training and evaluation can be seen in what follows: Figs. 5.5, 5.7, 5.9 and 5.11 depict the loss functions during the models' training, Figs. 5.6, 5.8, 5.10 and 5.12 show the results' confusion matrices and Tables 5.5 - 5.8 present the corresponding evaluation metrics.

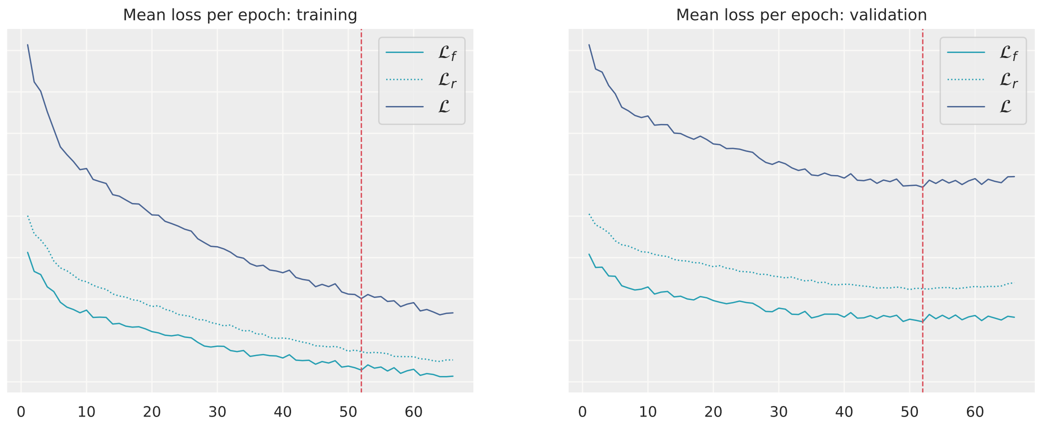


Figure 5.5: Loss functions for the training and validation of the MTAD-GATv2 model for the motor-pump systems.

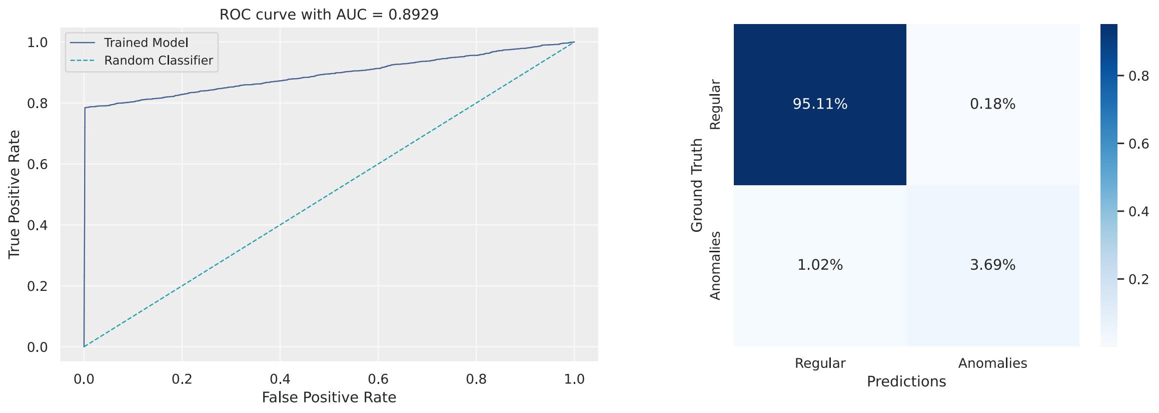


Figure 5.6: ROC curve & confusion matrix for the motor-pump systems' MTAD-GATv2 model.

Accuracy	**Recall**	**Precision**	**F1-Score**
98.81%	78.45%	95.41%	86.10%

Table 5.5: Evaluation metrics for the MTAD-GATv2 model for the motor-pump systems.

Since each MTAD-GATv2 model instance is trained by minimizing the joint loss function of Eq. (5.2), which corresponds to a forecasting loss function, $\mathcal{L}_f$, and a reconstruction loss function, $\mathcal{L}_r$, the figures portray both individual loss functions as well as their sum, as far as the models' training (left) and validation (right) are concerned. The vertical dashed red line signals the epoch where the early stopping mechanism is activated and training is terminated, due to a constant increase in validation losses. The reason why some graphs show $\mathcal{L}_f$ higher than $\mathcal{L}_r$ (or vice-versa), with this relevant positioning also changing in some cases as the number of epochs increases, is simply because sometimes the losses corresponding to the forecasting's optimization are higher compared to the ones that correspond to the reconstruction's optimization (or vice-versa).

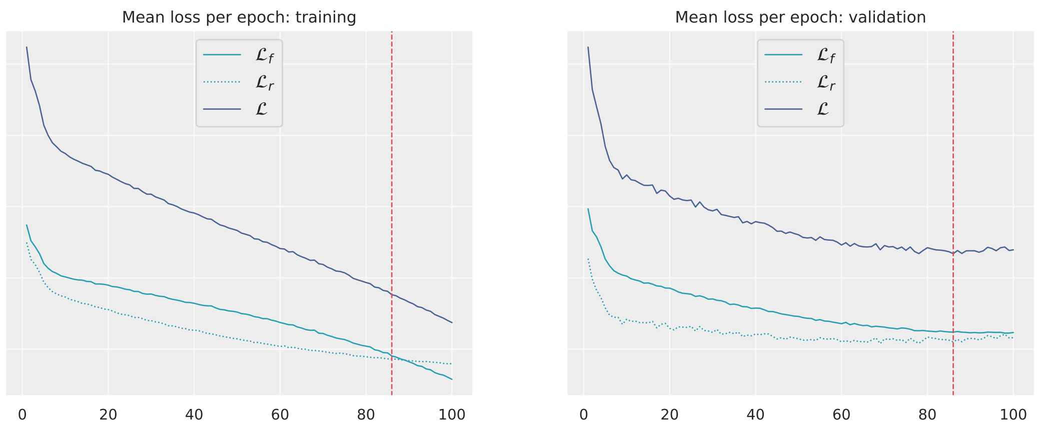


Figure 5.7: Loss functions for the training and validation of the MTAD-GATv2 model for the stern tube.

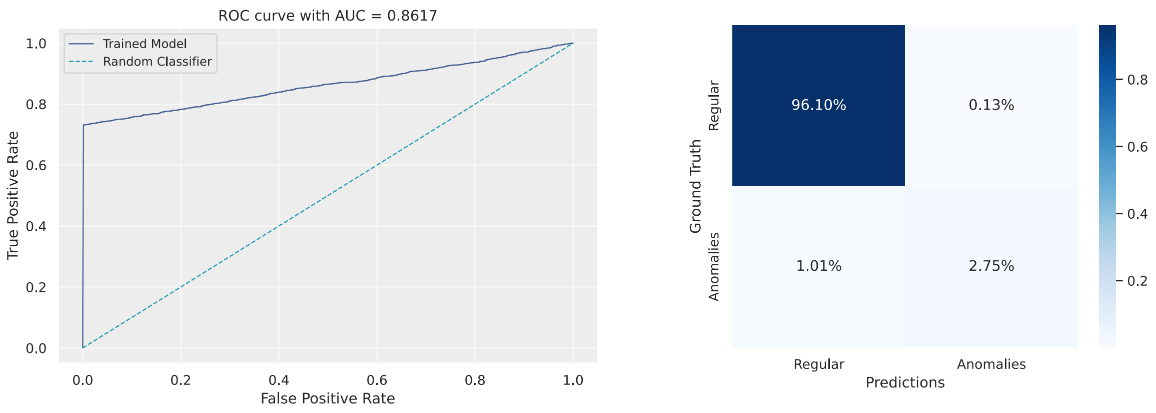


Figure 5.8: ROC curve & confusion matrix for the stern tube's MTAD-GATv2 model.

Accuracy	**Recall**	**Precision**	**F1-Score**
98.86%	73.18%	95.33%	82.80%

Table 5.6: Evaluation metrics for the MTAD-GATv2 model for the stern tube.

Unlike in the case of UVAD models, the ROC curves obtained from the MVAD models' results are also depicted in this section. The AUC is higher than 86.17% in all cases, which might not seem optimal at first sight[(22)], however one must keep in mind the one-to-one evaluation process previously discussed, as well as the fact that the pre-training datasets are custom datasets developed for the very specific requirements of the present work and cannot be easily compared to commonly available datasets used for anomaly detection benchmarking. As far as the shapes of the curves are concerned, the fact that they are steep at the beginning and then start moving horizontally signifies that the positive (anomalies) and negative (regular data) classes are well separated for a given range of thresholds. This happens because the corresponding samples are well separated in the feature space and the model is able to exploit this separation to make accurate predictions.

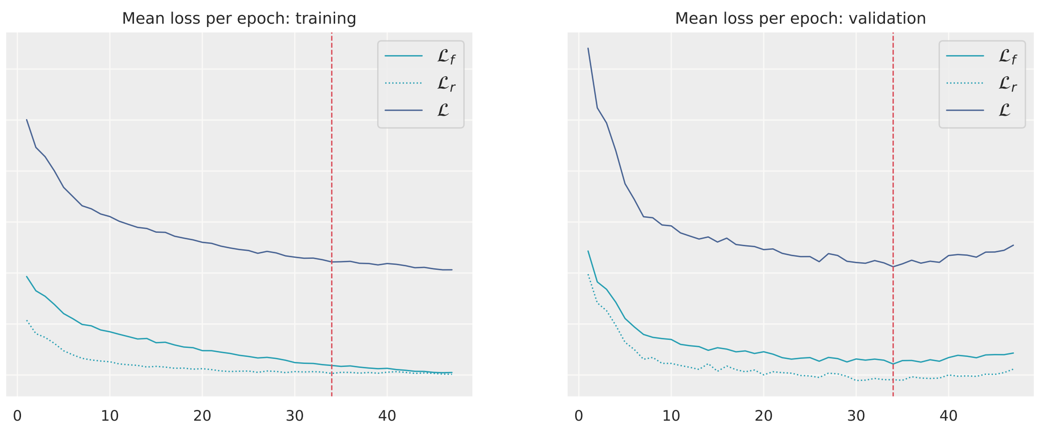
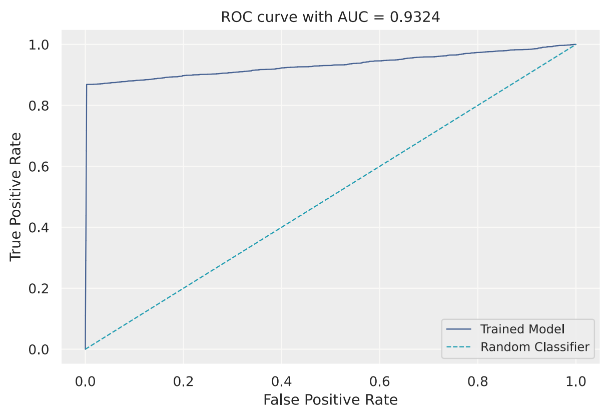


Figure 5.9: Loss functions for the training and validation of the MTAD-GATv2 model for the pipe systems.



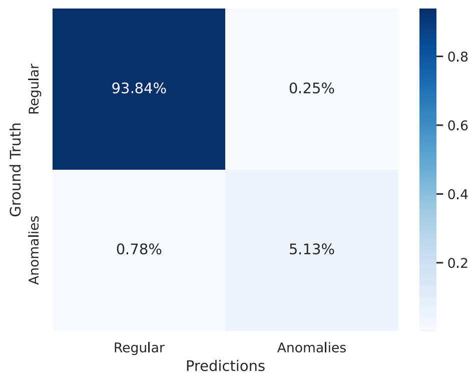


Figure 5.10: ROC curve & confusion matrix for the pipe systems' MTAD-GATv2 model.

Accuracy	**Recall**	**Precision**	**F1-Score**
98.98%	86.86%	95.41%	90.94%

Table 5.7: Evaluation metrics for the MTAD-GATv2 model for the pipe systems.

[(22)] In the original MTAD-GAT paper it is stated that the AUC of ROC curves is not chosen as an evaluation metric, as all models yield AUCs which are higher than 97%. Nonetheless, it is also stated that the used datasets concern only continuous (and not point) anomalies, where the correct detection of a single anomaly renders the whole segment of anomalies as correctly detected.

As regards the other evaluation metrics, it is worth noting that the confusion matrices are given here normalized, since the exact support of each area can be inferred from the existing knowledge of the total number of samples and contamination factors (in the case of UVAD models, the exact anomaly distribution across features is not given beforehand, with the exception of the graphs of Appendix C, which is why each individual support is depicted as an absolute number). The recall and F1-Scores achieved by the MVAD model instances are significantly better compared to the UVAD ones, perhaps with the exception of the stern tube's dataset. Obviously, this can be attributed to the fact that the architecture of the MVAD-GATv2 model is such that it allows the inclusion and proper weighing of intra-feature correlations using the feature-oriented GATv2 layers and their dynamic attention mechanisms.

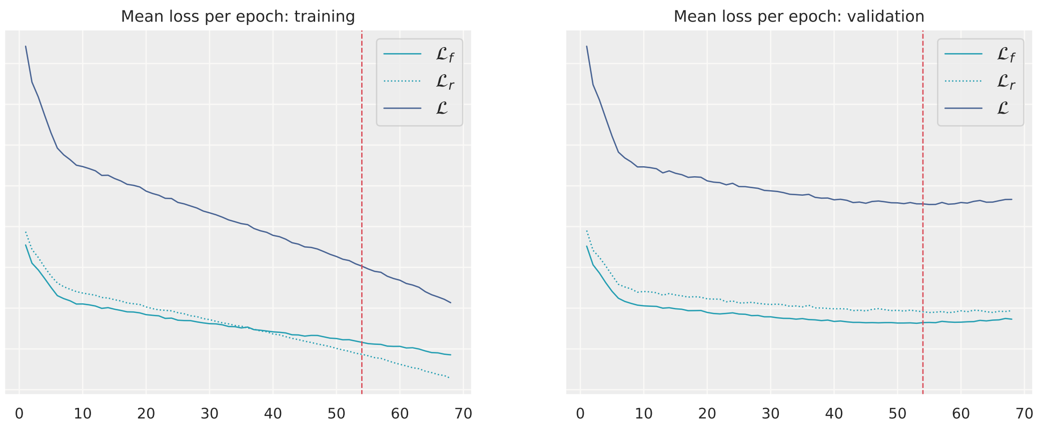


Figure 5.11: Loss functions for the training and validation of the MTAD-GATv2 model for the scrubber towers.

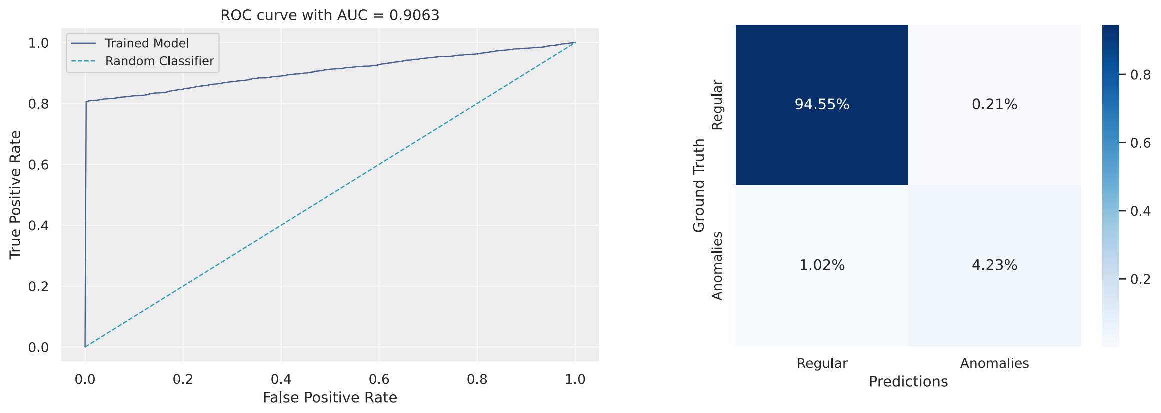


Figure 5.12: ROC curve & confusion matrix for the scrubber towers' MTAD-GATv2 model.

Accuracy	Recall	Precision	F1-Score
98.77%	80.63%	95.26%	87.34%

Table 5.8: Evaluation metrics for the MTAD-GATv2 model for the scrubber towers.

5.3 Expressivity Evaluation

With the pre-trained MVAD models readily available, the next part of the experimental process concerns their evaluation on the synthetic time-series datasets. Again, when it comes to UVAD models, their results do not provide information about their expressivity, as they are self-trained anew on the new time-series. Instead, they simply indicate how good the SR-CNN model is in identifying different types of anomalies in periodic data. The evaluation results for each SR-CNN instance can be seen in the confusion matrices of Fig. 5.13 for each univariate dataset, as well as in Table 5.9.

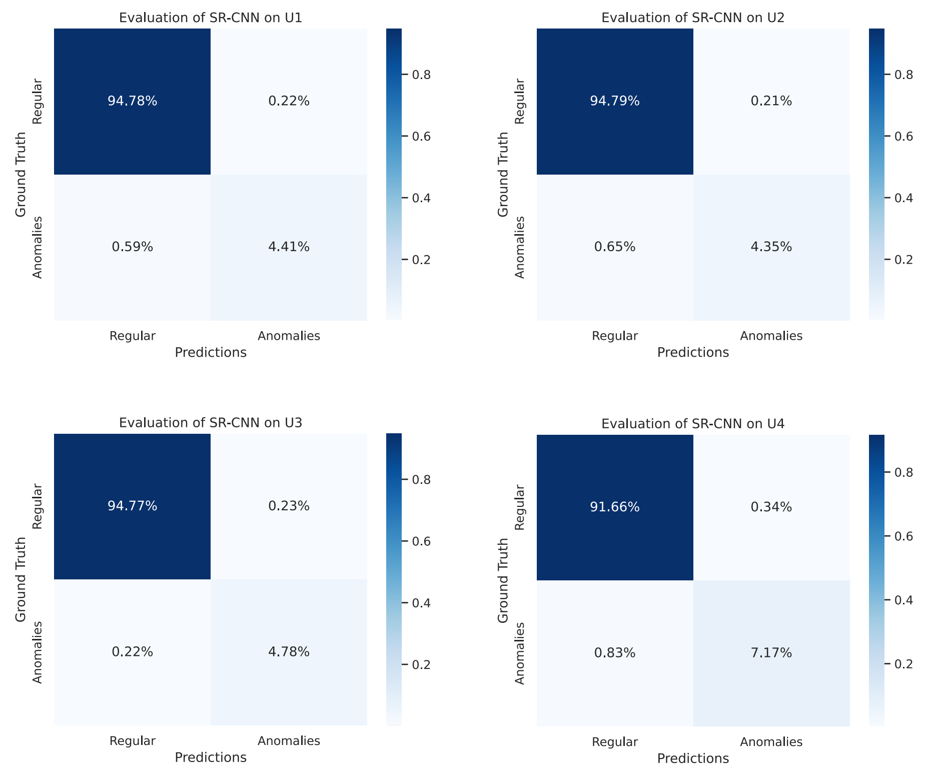


Figure 5.13: Confusion matrices for the SR-CNN model on the univariate datasets.

Dataset	Accuracy	Recall	Precision	F1-Score
U1	99.19%	88.20%	95.25%	91.59%
U2	99.14%	87.00%	95.39%	91.00%
U3	99.55%	95.60%	95.41%	95.50%
U4	98.83%	89.62%	95.47%	92.46%

Table 5.9: Evaluation metrics for the SR-CNN model on the univariate datasets.

The results are considerably better compared to the results for the pre-training datasets. This can be attributed to the fact that the synthetically generated time-series are based on periodic functions with relatively small variance, so learning this periodic behaviour and then classifying divergences from this periodicity as anomalies (especially when they correspond to well-defined types of anomalies, as is the case for the time-series in these datasets) is quite simple for a CNN model to achieve.

As for the MVAD models, their results can be seen in Fig. 5.14 and Table 5.10.

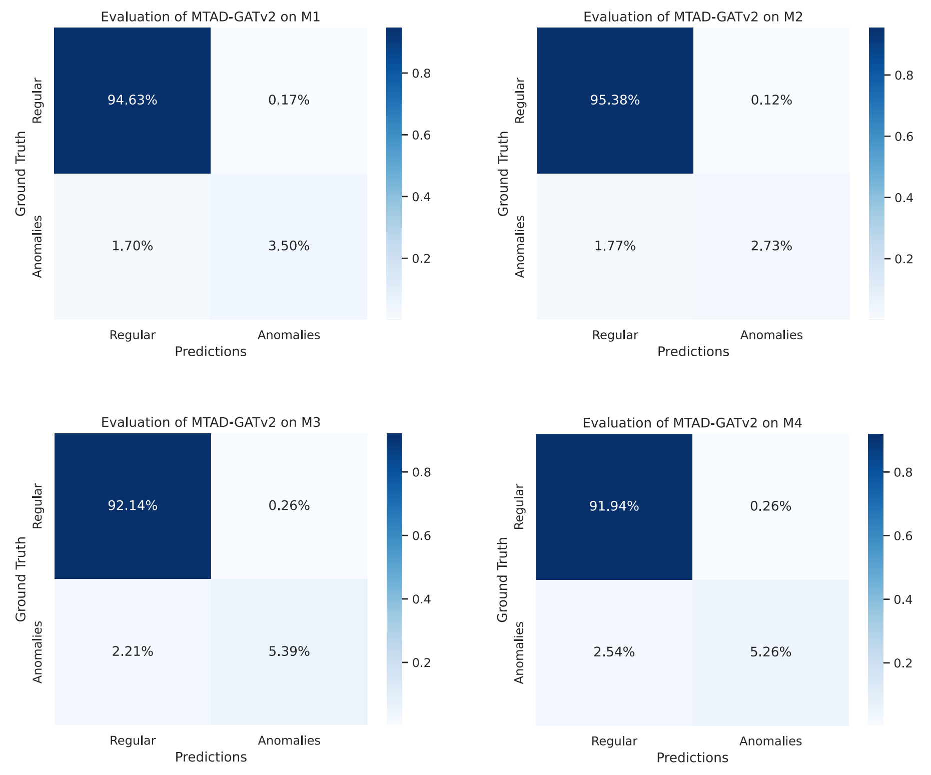


Figure 5.14: Confusion matrices for the MTAD-GATv2 model on the multivariate datasets.

Dataset	**Accuracy**	**Recall**	**Precision**	**F1-Score**
M1	98.13%	67.31%	95.37%	78.92%
M2	98.11%	60.67%	95.79%	74.29%
M3	97.53%	70.92%	95.40%	81.36%
M4	97.20%	67.44%	95.29%	78.98%

Table 5.10: Evaluation metrics for the MTAD-GATv2 model on the multivariate datasets.

As expected, the results for the MTAD-GATv2 model instances are not as promising as the ones for the SR-CNN. The first reason concerns the fact that the anomalies present in these datasets are "mathematical", in the sense that many of them can't be expected to appear in real-world problems (for example, a uniform, constant shift for all values in a given time window does not have physical significance). The other reason is that the distributions from which the regular time-series data are generated in these datasets (periodic functions) are completely different compared to the distributions that generate the real-world sensor data. Besides, these are the reasons why these datasets were generated as an expressivity challenge. All things considered, a mean recall of 66.49% and an average F1-Score of 78.39% are satisfactory, especially given how the evaluation is performed (the one-to-one evaluation scheme described in the previous section).

5.4 Fine-Tuning and Case Studies

The final and perhaps most interesting part of this chapter concerns the findings using the real-world data transmitted from operational vessels equipped with sensors. Before investigating two case studies, it is worth mentioning that the SR-CNN model is not utilized for inference, as it is costly and inferior to the MTAD-GATv2 model. Additionally, the MTAD-GATv2 pre-trained model instances are not directly deployed for inference after their training. Instead, a fine-tuning procedure is first followed, essentially equivalent to transfer learning.

First, approximately 10000 values (about 4 days of data feed) are extracted from the time-series data transmitted from each system's sensors. Then, the data are cleaned from possible anomalies (which are extremely rare in such short windows of operation), by applying the SR technique presented in Section 2.1.1. The threshold, τ, required for the rule of Eq. (2.7) is defined by an iterative POT process similar to the one applied for the automatic inference threshold calculation of the MTAD-GATv2 model. This effectively creates a set of mini time-series containing non anomalous data, on which the pre-trained models are fine-tuned for a total of 15 epochs.

As discussed, while the pre-training datasets are a good way to train models in a supervised manner, the conditions on which their data are collected do not accurately represent real-world vessel conditions and operation. As a result, this fine-tuning is essential, so that the models can capture additional information regarding the distributions from which the real-time data are generated. There are several different ways to perform this fine-tuning process as far as the models' learnable parameters are concerned[118–120]. The approach followed here concerns freezing all learned parameters except for some of the final layers of the forecasting and reconstruction models and retraining them with a learning rate of $5 \cdot 10^{-5}$.

With the fine-tuned models deployed on Kubernetes as explained in Chapter 4 and performing inference on received data, after 2 months (by the time of writing the present thesis) of data transmission, several alerts of detected anomalies have been published by the models. The two most significant among them, which were verified by engineers who performed follow-up checks on the corresponding systems, are discussed in what follows.

5.4.1 Case #1: Cracks on Scrubber Towers

The first verified anomaly detection event concerns a scrubber tower system. As explained in Chapter 3, the most common problem with scrubber towers are cracks and damages that occur due to thermal shocks, which are in turn caused by sudden changes in temperature. Fig. 5.15 depicts three segments of the pre-processed time-series obtained from the `TempB1` feature's sensors (the situation is similar for the other temperature-related features, which is why depicting only one suffices). The vertical dashed lines indicate the beginning and the end of the scrubber tower's heating process, which is a crucial part of its operation, and the segments have been positioned in a way such that the beginning of the heating process is common in all three. In the first segment, the heating process is done smoothly, over a certain period of time (smooth heating). In the second segment the heating process is done over a smaller time-window and less smoothly (the curve is almost linear, hence the title linear heating). In the third segment, the heating process is finished at almost half the time compared to the first and in a very sharp manner (sharp heating). The MTAD-GATv2 model correctly identifies the behaviour shown in the second and third segment as a series of anomalies: more than 3/5 of the second segment's area within borders is recognized as anomalous, while the entirety of the area within borders of the third segments is recognized as anomalous.

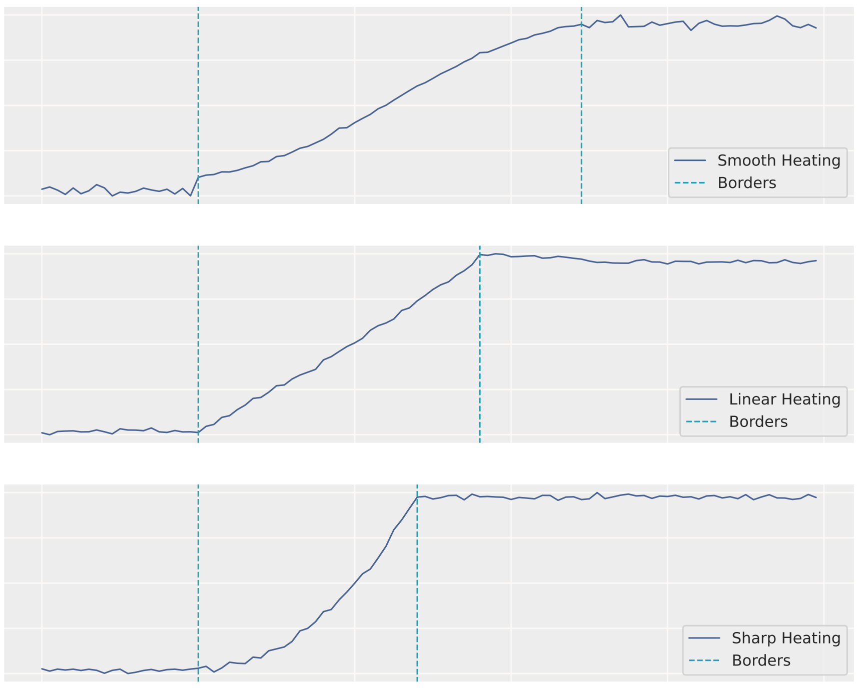


Figure 5.15: Three different segments portraying the depicting a scrubber tower's heating process over different time windows, with different behaviours.

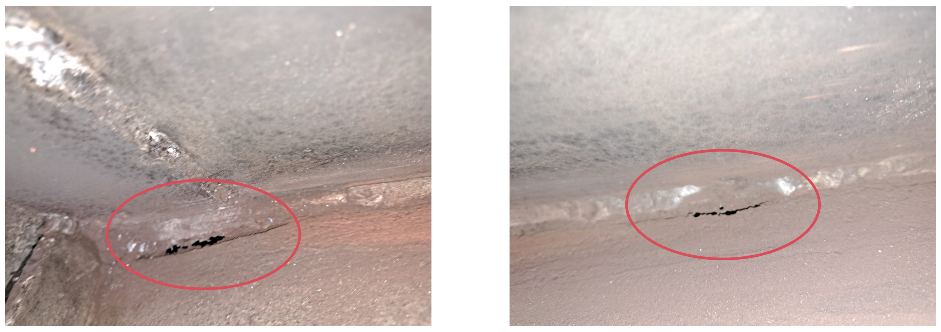


Figure 5.16: Visual inspection of cracks on the studied scrubber tower.

It is important to note that the analysis of these segments is made possible thanks to the ON/OFF metadata discussed in Section 4.2. The scrubber towers are not always in operating mode, but are activated (ON state) and de-activated (OFF state) depending on which engine is used (the studied vessel is a cruise liner with six diesel engines) and on whether they are needed or not. However, the sensors attached on them receive and transmit measurements for all the monitored features even when the scrubber towers are not in operation. It is evident that the data gathered while a scrubber tower is in its ON state come from a different distribution compared to the ones gathered while the scrubber tower is in its OFF state. Therefore, filtering the data on this key is essential for the MTAD-GATv2 model to be able to recognize anomalous events during the scrubber tower's operation[23].

Interestingly, about two weeks after the MTAD-GATv2 model first reported the anomalous behaviour of this scrubber tower (which was most likely due to bad operation practices), an engineer performed an inspection on the tower. Among the findings of the inspection were two small cracks on the tower (see Fig. 5.16), which were most likely the result of thermal shocks caused by the non-smooth heating of the scrubber towers during operation.

5.4.2 Case #2: Shaft Misalignment in Motor-Pump Systems

The second case study corresponds to a monitored motor-pump system and is a great example of why MVAD algorithms are necessary to detect anomalies in scenarios such as the studied one. Fig. 5.17 portrays segments of the time-series for the `VibM`, `TempM` and `CurrM` features extracted from a motor-pump system's sensors. While the values have been removed from the axes for privacy reasons, note that the three time-series share the same time axis, i.e. vertical slices thereof correspond to the system's feature vector at a given timestamp.

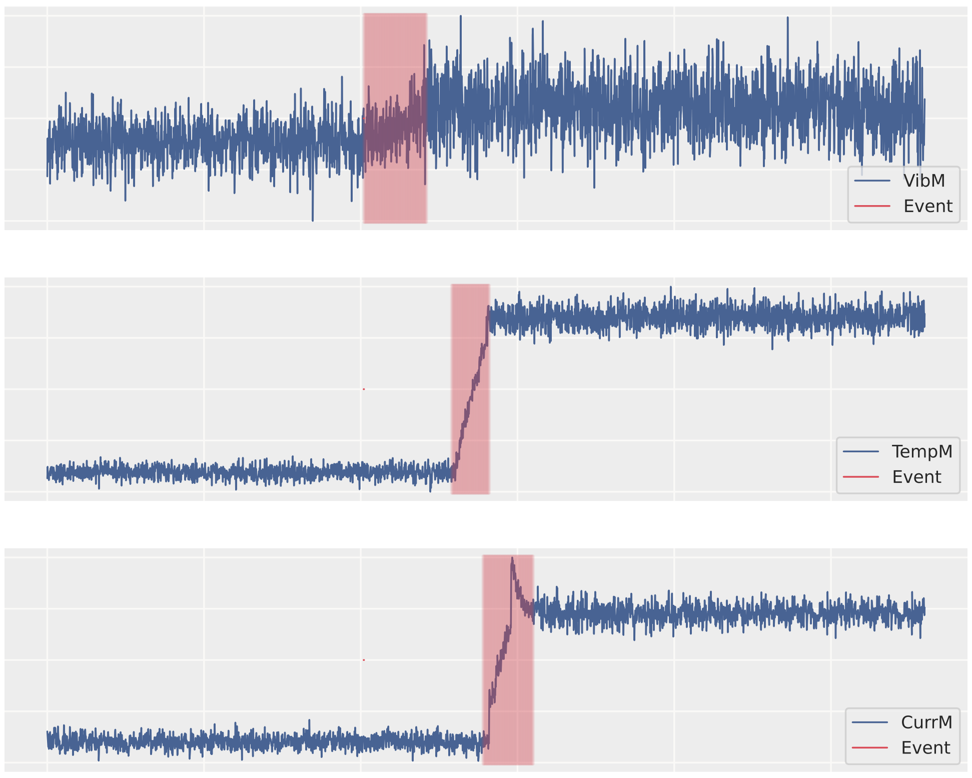


Figure 5.17: Multivariate anomaly detection of an event that propagates across features.

First, the time-series segment corresponding to the `VibM` feature shows a very slight shift which is almost inconceivable by a visual inspection and is followed by values with larger variance compared to the previous ones. A few timestamps later, a sharp increase in temperature can be seen; while the

[23] Without this filtering and due to the fact that the alternation between ON and OFF states is not periodic, it is very likely that the algorithm would identify the scrubber tower's operational states as continuous anomalies.

absolute value of the temperature does not change significantly, a phase-transition-like behaviour is clear. Before this transition is complete, another transition seems to appear for the current, although in a different way (i.e. with a different functional dependence). Upon inspection, it was found that the motor's shaft was slightly misaligned, which is what caused this series of highly correlated events. Note that in Fig. 5.17 the events have been highlighted after the fact, as an interpretation of what happened (some sort of labeling). The MTAD-GATv2 model identified the event as an anomaly before it propagated to the temperature and current features. Of course the anomaly score assigned to the event while it was only present in the `VibM` feature was smaller compared to the anomaly score it assigned to the event once it became visible in the other two features (almost equal to maximum).

6 Conclusion

In this thesis, an approach for detecting anomalies in shipboard systems through the application of machine learning was presented. Specifically, UVAD and MVAD algorithms were deployed with the purpose of monitoring the health of vessel systems and identifying potential issues before they became critical. The Spectral Residual Convolutional Neural Network (SR-CNN) model was used for univariate anomaly detection, while an advanced version of Microsoft's Multivariate Time-Series Anomaly Detection via Graph Attention Network (MTAD-GAT) model was used for multivariate anomaly detection. The most significant improvement on Microsoft's original model concerned the addition of GATv2 layers, which compute dynamic instead of static attention (MTAD-GATv2).

Three separate dataset types were developed to train the MVAD models, evaluate their performance and compare it to the performance of the self-trained SR-CNN instances. The first among them contained datasets with artificially induced anomalies for pre-training the MVAD models which would be deployed to monitor four different types of shipboard systems: motor-pump systems, stern tubes, pipes and scrubber towers. The second dataset type consisted of synthetic time-series datasets which contained 8 different types of anomalies injected on 4 different types of base periodic functions, with the aim of evaluating the expressivity of the trained MVAD models using unseen types of data and anomalies. The third and final type of dataset comprised the time-series extracted from the feed of the sensors installed on operational vessels.

Additionally, a comprehensive analysis of the data engineering process was conducted, thoroughly explaining the development of a complete data pipeline on the cloud using Microsoft's Azure Ecosystem. The first part of this pipeline concerned the batch ingestion of the transmitted sensor data into the cloud through IoT Hubs and their organization into containers and directories in a query-optimized way using Stream Analytics. Then, the storing of the data using the Delta Lake format was discussed, including an introduction of the medallion architecture, where data is logically organized in multiple layers, with each layer representing a different step in the data processing pipeline. Finally, the role of computation services such as Databricks - both in data engineering and in machine learning - was investigated, before describing the pipeline's exit point where results are served on a PostgreSQL database at the back-end of a dashboard web application.

The results from training and evaluating the models on the first type of datasets demonstrated the promising nature of the proposed approach. Using a one-to-one evaluation scheme, similar to a classification problem, the MVAD models seemed to perform quite well for all used metrics: accuracy, recall, precision, F1-Score and AUC of ROC curves. On the other hand, the UVAD instances noticeably outperformed the pre-trained MVAD models on the second type of datasets, since the latter were not trained on data from similar distributions or types of anomalies, while the former were self-trained on each synthetic time-series separately. The most important result, however, was the MVAD models' achievement in identifying two cases of anomalous behavior in the time-series of a vessel's live data feed: cracks on a scrubber tower system caused by sharp increases in temperature and the misalignment of a motor's shaft in a motor-pump system. After the detection of these anomalies, they were validated by inspections performed by engineers onboard.

6.1 Outlook

This work represents a significant step forward in the digitalization and automation of the marine industry, through the utilization of machine learning algorithms for anomaly detection in vessel systems. Nonetheless, there are still challenges to be addressed and opportunities for further development. As far as the studied models are concerned, a possible avenue for future work is replacing the VAE model in the MTAD-GATv2 architecture with another, perhaps less sophisticated model (such as a GRU-based reconstruction model) in favor of computational efficiency, as long as it does not come at the expense of predictive accuracy. Besides, this was the reason a MLP was used as the forecasting model instead of more complex architectures such as LSTMs or Conv-LSTMs, which were tested but did not yield noticeably better results. In this case, it is also worth exploring the use of GATs instead of GAT layers (i.e. collections of GAT layers instead of single ones).

As far as the datasets are concerned, a significant improvement would be to use the datasets of the real-time data feed collected from sensors on operational vessels for the pre-training of the MTAD-GATv2 model instances. Of course, this requires the labeling of millions of data points, which is a highly parallelizable, yet time-consuming task. For this reason, a direction worth exploring is the automatic labeling of the datasets by using either deterministic algorithms or machine learning[121]. Another possible step could be towards designing richer datasets for the evaluation of the models' expressivity, for example using Generative Adversarial Networks (GANs), such as the TGAN-AD[122], which is an anomaly detector itself, however its architecture can be used for generative tasks.

Last but not least, when it comes to the Data Engineering process, the data pipeline that handles the ingestion, storing, processing and serving of the data should be enriched with additional practices concerning the machine learning operations: scheduled model re-training, continuous re-evaluation on new data as they become available, hyper-parameter re-tuning, to name but a few. The most well known framework suitable for such operations is MLflow[123], which is an open source platform that helps manage the machine learning lifecycle. MLflow is ideal for the work presented here, since it also offers the possibility to create endpoints for the machine learning models and is compatible with Databricks workspaces, thus rendering the use of FastAPI and Kubernetes Clusters redundant.

Appendix

A. Different Types of Attention

As mentioned in Section 2.2.1.2, Brody et al.[65] define two different types of attention. If the attention mechanism always weighs one key at least as much as any other key regardless of the query, then this attention mechanism is static. More formally:

Definition A.1: *A family of scoring functions $\mathcal{F} \subseteq (\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R})$ computes static scoring for a given set of key vectors $\mathbb{K} = \{\mathbf{k}_1, \dots, \mathbf{k}_\ell\} \subset \mathbb{R}^d$ and query vectors $\mathbb{Q} = \{\mathbf{q}_1, \dots, \mathbf{q}_n\} \subset \mathbb{R}^d$, if for every $f \in \mathcal{F}$ there exists a "highest scoring" key $j_f \in [\ell]$ such that for every query $i \in [n]$ and key $j \in [\ell]$ it holds that $f\left(\mathbf{q}_i, \mathbf{k}_{j_f}\right) \geq f\left(\mathbf{q}_i, \mathbf{k}_j\right)$. A family of attention functions computes static attention given $\mathbb{K}$ and $\mathbb{Q}$, if its scoring function computes static scoring, possibly followed by monotonic normalization, such as softmax.*

In the above definition, ℓ and n are the cardinalities of $\mathbb{K}$ and $\mathbb{Q}$, respectively, d is the dimension of the key/query vectors and $[s] = \{1, \dots, s\} \subset \mathbb{N}$. Following this definition, one may prove the following theorem:

Theorem A.1: *A GAT layer computes static attention for any set of node representation vectors $\mathbb{K} = \mathbb{Q} = \{\mathbf{v}_1, \dots, \mathbf{v}_k\}$.*

Recall that a GAT layer is described by Eqs. (2.11) - (2.13). To prove Theorem A.1, consider a graph modeled by a GAT layer with some learned **W** and **a** layers and having node vectors $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$. The **a** vector can be written as the concatenation $\mathbf{a} = [\mathbf{a}_l \oplus \mathbf{a}_r] \in \mathbb{R}^{2T'}$, with $\mathbf{a}_l, \mathbf{a}_r \in \mathbb{R}^{T'}$. Then, Eq. (2.13) assumes the form

$$e_{ij} = A_{ij} \operatorname{LeakyReLU}\left(\mathbf{a}_l \cdot \mathbf{W} \cdot \mathbf{v}_i + \mathbf{a}_r \cdot \mathbf{W} \cdot \mathbf{v}_j\right). \tag{A.1}$$

Since $[k]$ is finite, there exists a node $j_f \in [k]$ such that $\mathbf{a}_r \cdot \mathbf{W} \cdot \mathbf{v}_{j_f}$ is maximal among all nodes $j \in [k]$. Due to the monotonicity of LeakyReLU and softmax, for every query node $i \in [k]$, the node j_f also leads the maximal value of its attention distribution $\{\alpha_{ij} \mid j \in [k]\}$. It follows directly from Definition A.1 that the GAT computes only static attention. Note that in the case of multi-head attention, Theorem A.1 holds for each head separately: every attention head $m \in [M]$ has a specific node that maximizes $\{\mathbf{a}_r \cdot \mathbf{W}^m \cdot \mathbf{v}_j \mid j \in [k]\}$ and the output is the concatenation/average of M static attention heads.

Due to the limitations of static attention discussed in Section 2.2.1.2, Brody et al. suggest a more expressive form of attention, which they call dynamic attention:

Definition A.2: *A family of scoring functions $\mathcal{F} \subseteq \left(\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}\right)$ computes dynamic scoring for a given set of key vectors $\mathbb{K} = \{\mathbf{k}_1, \dots, \mathbf{k}_\ell\} \subset \mathbb{R}^d$ and query vectors $\mathbb{Q} = \{\mathbf{q}_1, \dots, \mathbf{q}_n\} \subset \mathbb{R}^d$, if for any mapping $\phi : [n] \to [\ell]$ there exists a $f \in \mathcal{F}$ such that $f\left(\mathbf{q}_i, \mathbf{k}_{\phi(i)}\right) > f\left(\mathbf{q}_i, \mathbf{k}_j\right)$, for any query $i \in [n]$ and any key $j \in [\ell]$ with $j \neq \phi(i)$. A family of attention functions computes dynamic attention given $\mathbb{K}$ and $\mathbb{Q}$, if its scoring function computes dynamic scoring, possibly followed by monotonic normalization, such as softmax.*

This definition indicates that, unlike static attention, dynamic attention can select every key $\phi(i)$ using the query i by making $f\left(\mathbf{q}_i, \mathbf{k}_{\phi(i)}\right)$ maximal in $\left\{f\left(\mathbf{q}_i, \mathbf{k}_j\right) \mid j \in [\ell]\right\}$. Based on this definition, the following theorem regarding GATv2 can be proven:

Theorem A.2: *A GATv2 layer computes dynamic attention for any set of node representation vectors $\mathbb{K} = \mathbb{Q} = \{\mathbf{v}_1, \dots, \mathbf{v}_k\}$.*

Recall that a GATv2 layer is described by Eqs. (2.19), (2.20), however for the following proof instead of Eq. (2.19) the equivalent Eq. (2.18) will be referenced. To prove Theorem A.2, consider a graph modeled by a GATv2 layer, having node vectors $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$. Additionally, let $\phi : [k] \to [k]$ be any arbitrary function that maps $[k] \to [k]$ and define $g : \mathbb{R}^{2T} \to \mathbb{R}$ as

$$g(\mathbf{x}) = \begin{cases} 1, & \text{if } \exists\, i : \mathbf{x} = \left[\mathbf{v}_i \oplus \mathbf{v}_{\phi(i)}\right] \\ 0, & \text{else} \end{cases}. \tag{A.2}$$

Next, define a continuous function $g_c : \mathbb{R}^{2T} \to \mathbb{R}$ such that it is equal to g only for the k^2 inputs $\left[\mathbf{v}_i \oplus \mathbf{v}_j\right]$, with $i, j \in [k]$, i.e.

$$g_c\left(\left[\mathbf{v}_i \oplus \mathbf{v}_j\right]\right) = g\left(\left[\mathbf{v}_i \oplus \mathbf{v}_j\right]\right), \quad \forall i, j \in [k]. \tag{A.3}$$

There is an infinite number of continuous functions that satisfy Eq. (A.3), since k^2 is finite. Now, by design, for every node $i, j \in [k]$ with $j \neq \phi(i)$,

$$1 = g_c\left(\left[\mathbf{v}_i \oplus \mathbf{v}_{\phi(i)}\right]\right) > g_c\left(\left[\mathbf{v}_i \oplus \mathbf{v}_j\right]\right) = 0 \tag{A.4}$$

holds. Consequently, by defining the scoring function

$$e\left(\mathbf{v}_i, \mathbf{v}_j\,;\, \mathbf{W}, \mathbf{a}\right) = A_{ij}\, \mathbf{a} \cdot \operatorname{LeakyReLU}\left(\mathbf{W} \cdot \left[\mathbf{v}_i \oplus \mathbf{v}_j\right]\right), \tag{A.5}$$

the universal approximation theorem[66] states that e can approximate g_c for any compact subset of $\mathbb{R}^{2T}$. This part clarifies the need to define g_c: the universal approximation theorem concerns only continuous functions and g is not. Nonetheless, since the theorem ensures that e can approximate g_c, then e also approximates g in the k^2 chosen points as a special case. Therefore, for any sufficiently small $\epsilon > 0$, there exist parameters $\mathbf{W}$ and $\mathbf{a}$ such that for all nodes $i, j \in [k]$ with $j \neq \phi(i)$,

$$e\left(\mathbf{v}_i, \mathbf{v}_{\phi(i)}\,;\, \mathbf{W}, \mathbf{a}\right) > 1 - \epsilon > 0 + \epsilon > e\left(\mathbf{v}_i, \mathbf{v}_j\,;\, \mathbf{W}, \mathbf{a}\right) \tag{A.6}$$

holds, thus proving that e computes dynamic scoring. Finally, due to the increasing monotonicity of softmax, this result implies that that α is an attention function which by Definition A.2 computes dynamic attention, thus concluding the proof of Theorem A.2. For completeness, it is stated that in the case of multi-head attention Theorem A.2 holds for each head separately (every attention head $m \in [M]$ computes dynamic attention).

B. VAE ELBO Calculation

In the present Appendix the approximated ELBO of Eq. (2.44) is calculated analytically for the Gaussian distributions of Eqs. (2.29), (2.30) and (2.42). As far as the reconstruction accuracy is concerned, assuming $M = 1$ based on the reasoning presented in Section 2.2.3.2, it is equal to

$$\sum_{n=1}^{N} \log p_\theta\left(\mathbf{x}_n|\mathbf{z}_n\right) = \sum_{n=1}^{N} \log \mathcal{N}\left(\mathbf{x}_n; \boldsymbol{\mu}_\theta\left(\mathbf{z}_n\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{z}_n\right)\right)$$
$$= -\frac{Nd}{2}\log\left(2\pi\right) - N\sum_{i=1}^{d}\log\sigma_\theta^{(i)}\left(\mathbf{z}_n\right) - \frac{1}{2}\sum_{n=1}^{N}\sum_{i=1}^{d}\frac{\left[x_n^{(i)} - \mu_\theta^{(i)}\left(\mathbf{z}_n\right)\right]^2}{\sigma_\theta^{(i)\,2}\left(\mathbf{z}_n\right)}, \tag{B.1}$$

where

$$\boldsymbol{\Sigma}_\theta\left(\mathbf{z}_n\right) = \text{diag}\left(\left\{\sigma_\theta^{(i)\,2}\left(\mathbf{z}_n\right)\right\}_{i=1}^{d}\right). \tag{B.2}$$

Excluding additive constants, Eq. (B.1) is essentially the squared Mahalanobis distance between the input and the decoder's output, summed over all input data points. Dividing this expression by N and requiring that the covariance matrix of the Gaussian distribution be the identity matrix, transforms it into half the mean-squared error (MSE) between the input dataset and the decoder's output.

Moving on to the regularization term, the KL divergence can be written as

$$-D_{\text{KL}}\left(q_\phi\left(\mathbf{z}_n|\mathbf{x}_n\right) \parallel p\left(\mathbf{z}_n\right)\right) = \int q_\phi\left(\mathbf{z}_n|\mathbf{x}_n\right)\left[\log p\left(\mathbf{z}_n\right) - \log q_\phi\left(\mathbf{z}_n|\mathbf{x}_n\right)\right] d\mathbf{z}_n$$
$$= \underbrace{\int q_\phi\left(\mathbf{z}_n|\mathbf{x}_n\right)\log p\left(\mathbf{z}_n\right)\, d\mathbf{z}_n}_{T_1} - \underbrace{\int q_\phi\left(\mathbf{z}_n|\mathbf{x}_n\right)\log q_\phi\left(\mathbf{z}_n|\mathbf{x}_n\right)\, d\mathbf{z}_n}_{T_2}, \tag{B.3}$$

where

$$T_1 = -\frac{L}{2}\log\left(2\pi\right) - \frac{1}{2}\int \mathbf{z}_n^2\, \mathcal{N}\left(\mathbf{z}_n; \boldsymbol{\mu}_\phi\left(\mathbf{x}_n\right), \boldsymbol{\Sigma}_\phi\left(\mathbf{x}_n\right)\right) d\mathbf{z}_n = -\frac{L}{2}\log\left(2\pi\right) -$$
$$-\frac{1}{2}\int\left[\left(\mathbf{z}_n - \boldsymbol{\mu}_\phi\left(\mathbf{x}_n\right)\right)^2 - \boldsymbol{\mu}_\phi^2\left(\mathbf{x}_n\right) + 2\mathbf{z}_n\cdot\boldsymbol{\mu}_\phi\left(\mathbf{x}_n\right)\right]\mathcal{N}\left(\mathbf{z}_n; \boldsymbol{\mu}_\phi\left(\mathbf{x}_n\right), \boldsymbol{\Sigma}_\phi\left(\mathbf{x}_n\right)\right) d\mathbf{z}_n$$
$$= -\frac{L}{2}\log\left(2\pi\right) - \frac{1}{2}\sum_{i=1}^{L}\left[\sigma_\phi^{(i)\,2}\left(\mathbf{x}_n\right) - \mu_\phi^{(i)\,2}\left(\mathbf{x}_n\right) + 2\mu_\phi^{(i)\,2}\left(\mathbf{x}_n\right)\right]$$
$$= -\frac{L}{2}\log\left(2\pi\right) - \frac{1}{2}\sum_{i=1}^{L}\left[\sigma_\phi^{(i)\,2}\left(\mathbf{x}_n\right) + \mu_\phi^{(i)\,2}\left(\mathbf{x}_n\right)\right] \tag{B.4}$$

and

$$\begin{aligned}
T_2 =& -\frac{L}{2}\log\left(2\pi\right) - \sum_{i=1}^{L}\log\sigma_{\phi}^{(i)}\left(\mathbf{x}_n\right) - \\
&-\frac{1}{2}\int\sum_{i=1}^{L}\frac{\left[z^{(i)}-\mu_{\phi}^{(i)}\left(\mathbf{x}_n\right)\right]^2}{\sigma_{\phi}^{(i)\,2}\left(\mathbf{x}_n\right)}\mathcal{N}\left(\mathbf{z}_n;\boldsymbol{\mu}_{\phi}\left(\mathbf{x}_n\right),\boldsymbol{\Sigma}_{\phi}\left(\mathbf{x}_n\right)\right)d\mathbf{z}_n \\
=& -\frac{L}{2}\log\left(2\pi\right) - \sum_{i=1}^{L}\log\sigma_{\phi}^{(i)}\left(\mathbf{x}_n\right) - \frac{L}{2}, \qquad \text{(B.5)}
\end{aligned}$$

with

$$\boldsymbol{\Sigma}_{\phi}\left(\mathbf{x}_n\right) = \operatorname{diag}\left(\left\{\sigma_{\phi}^{(i)\,2}\left(\mathbf{x}_n\right)\right\}_{i=1}^{d}\right). \qquad \text{(B.6)}$$

As a result, the regularization term becomes

$$\begin{aligned}
-\sum_{n=1}^{N}D_{\text{KL}}\Big(q_{\phi}\left(\mathbf{z}_n|\mathbf{x}_n\right)\parallel p\left(\mathbf{z}_n\right)\Big) =& \frac{NL}{2} + \sum_{n=1}^{N}\sum_{i=1}^{L}\log\sigma_{\phi}^{(i)}\left(\mathbf{x}_n\right) - \\
&-\frac{1}{2}\sum_{n=1}^{N}\sum_{i=1}^{L}\left[\sigma_{\phi}^{(i)\,2}\left(\mathbf{x}_n\right)+\mu_{\phi}^{(i)\,2}\left(\mathbf{x}_n\right)\right]. \qquad \text{(B.7)}
\end{aligned}$$

Finally, dividing by N to average over all dataset vectors and ignoring constant terms, the averaged approximated ELBO becomes

$$\begin{aligned}
\bar{\mathcal{L}}\left(\theta,\phi;\mathbf{X}\right) =& -\sum_{i=1}^{d}\log\sigma_{\theta}^{(i)}\left(\mathbf{z}_n\right) - \frac{1}{2N}\sum_{n=1}^{N}\sum_{i=1}^{d}\frac{\left[x_n^{(i)}-\mu_{\theta}^{(i)}\left(\mathbf{z}_n\right)\right]^2}{\sigma_{\theta}^{(i)\,2}\left(\mathbf{z}_n\right)} - \\
&-\frac{1}{2N}\sum_{n=1}^{N}\sum_{i=1}^{L}\left[\sigma_{\phi}^{(i)\,2}\left(\mathbf{x}_n\right)+\mu_{\phi}^{(i)\,2}\left(\mathbf{x}_n\right)\right] + \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{L}\log\sigma_{\phi}^{(i)}\left(\mathbf{x}_n\right). \qquad \text{(B.8)}
\end{aligned}$$

C. Anomaly Distribution for Dataset Type #1

This Appendix contains the ground truth labels for each feature of the first type of datasets, i.e. how a multivariate time-series' anomalies are distributed across features.

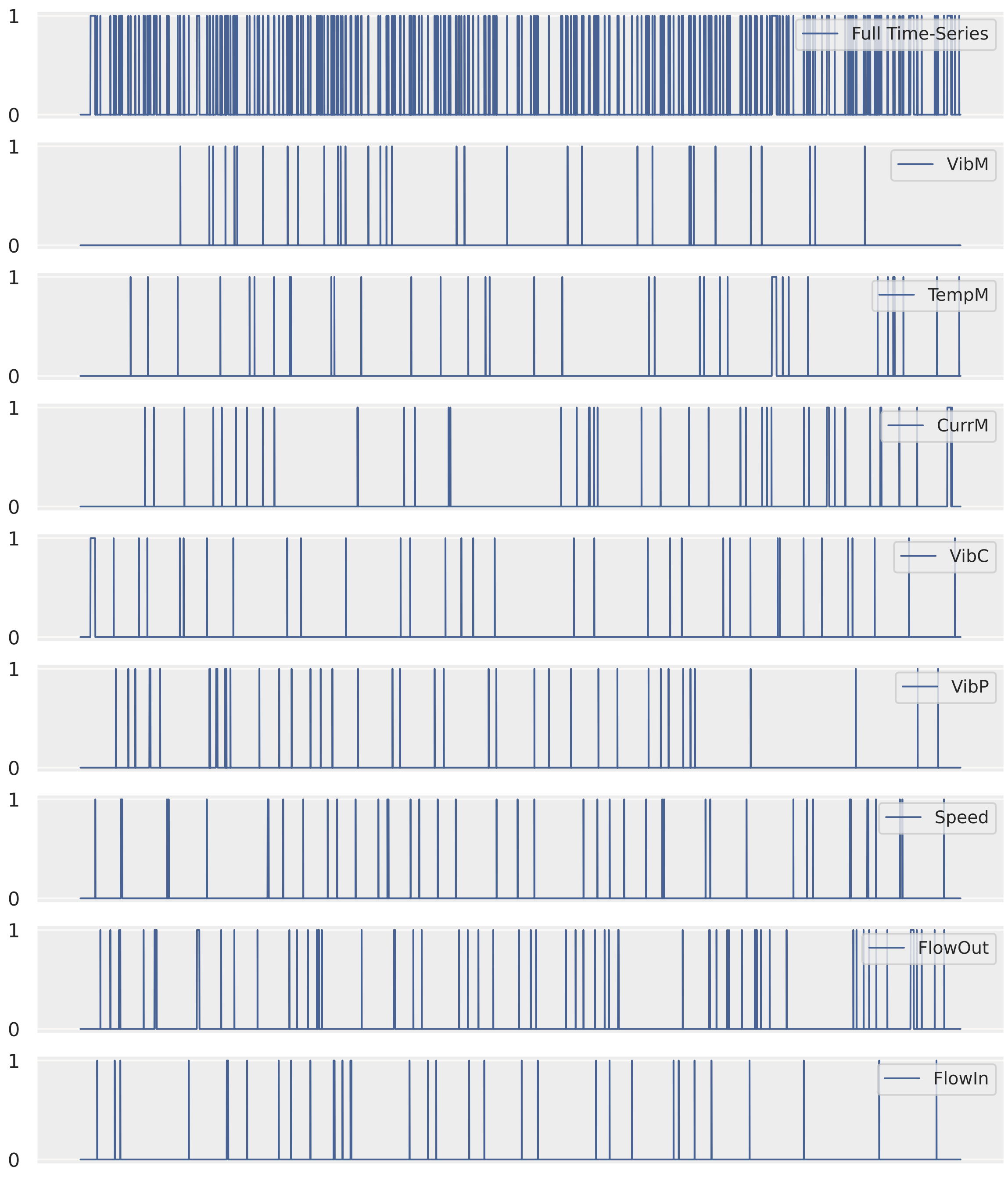


Figure C.1: Features' anomaly distribution for the pre-training dataset of the motor-pump systems.

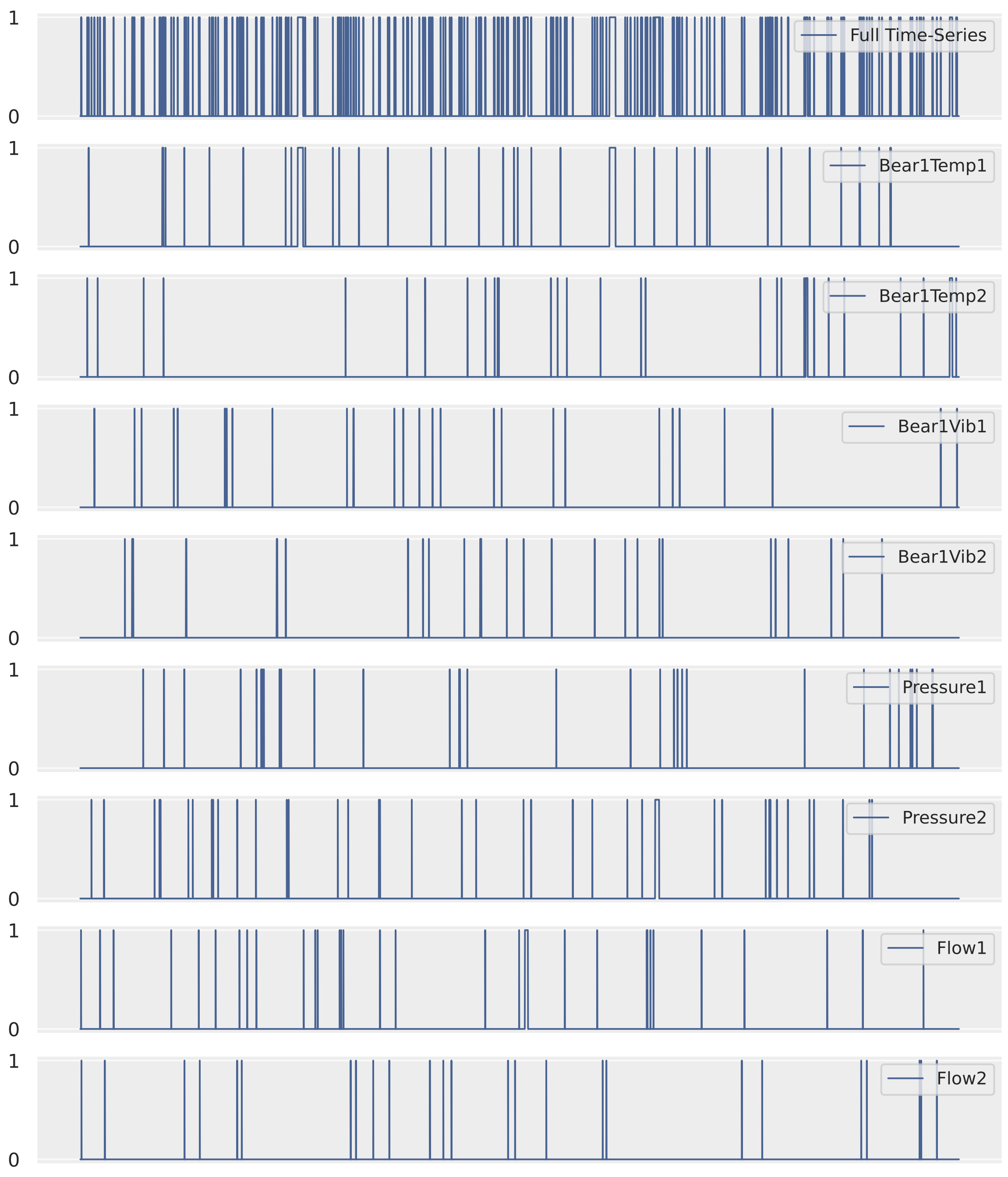


Figure C.2: Features' anomaly distribution for the pre-training dataset of the stern tube.

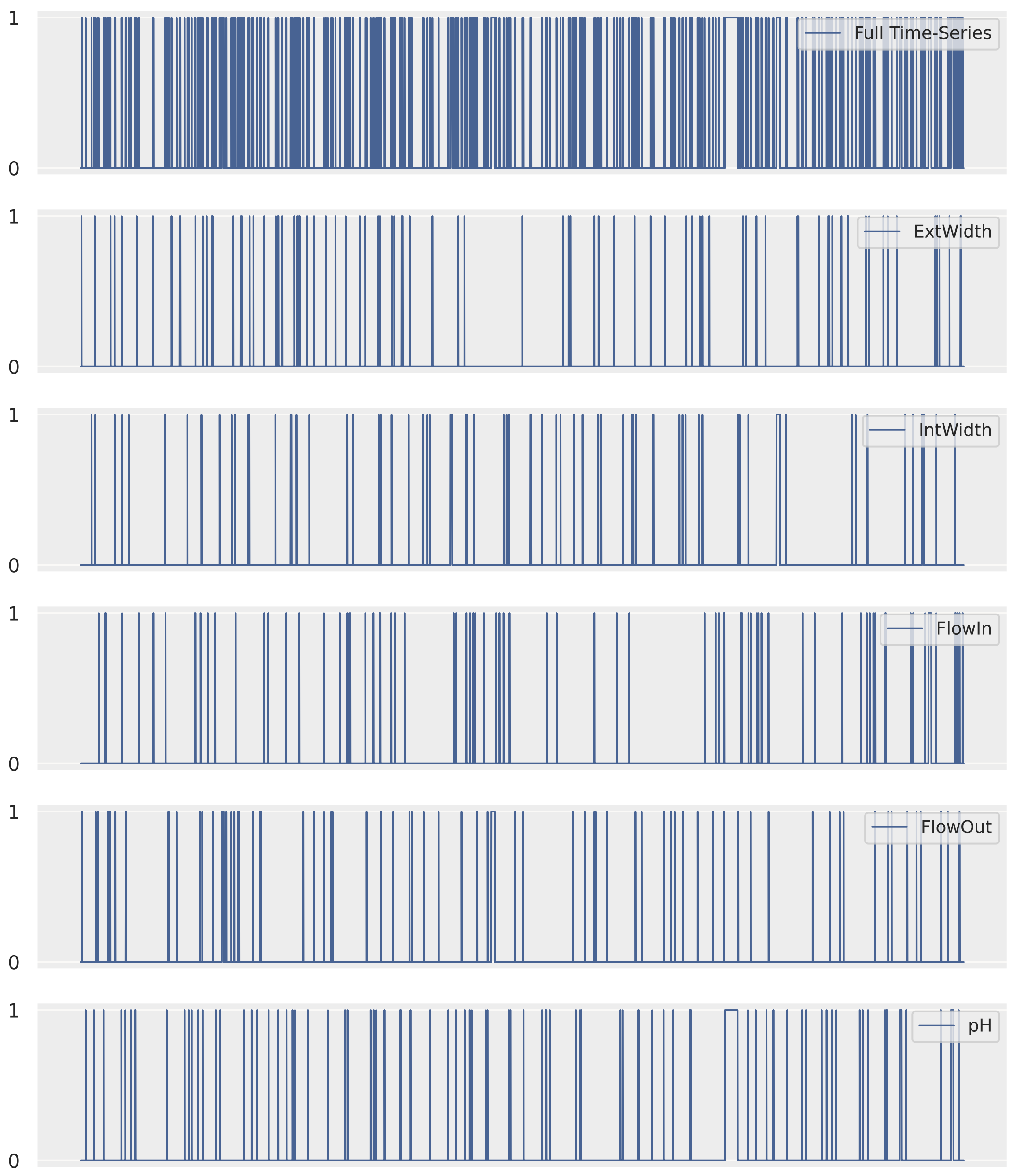


Figure C.3: Features' anomaly distribution for the pre-training dataset of the pipes.

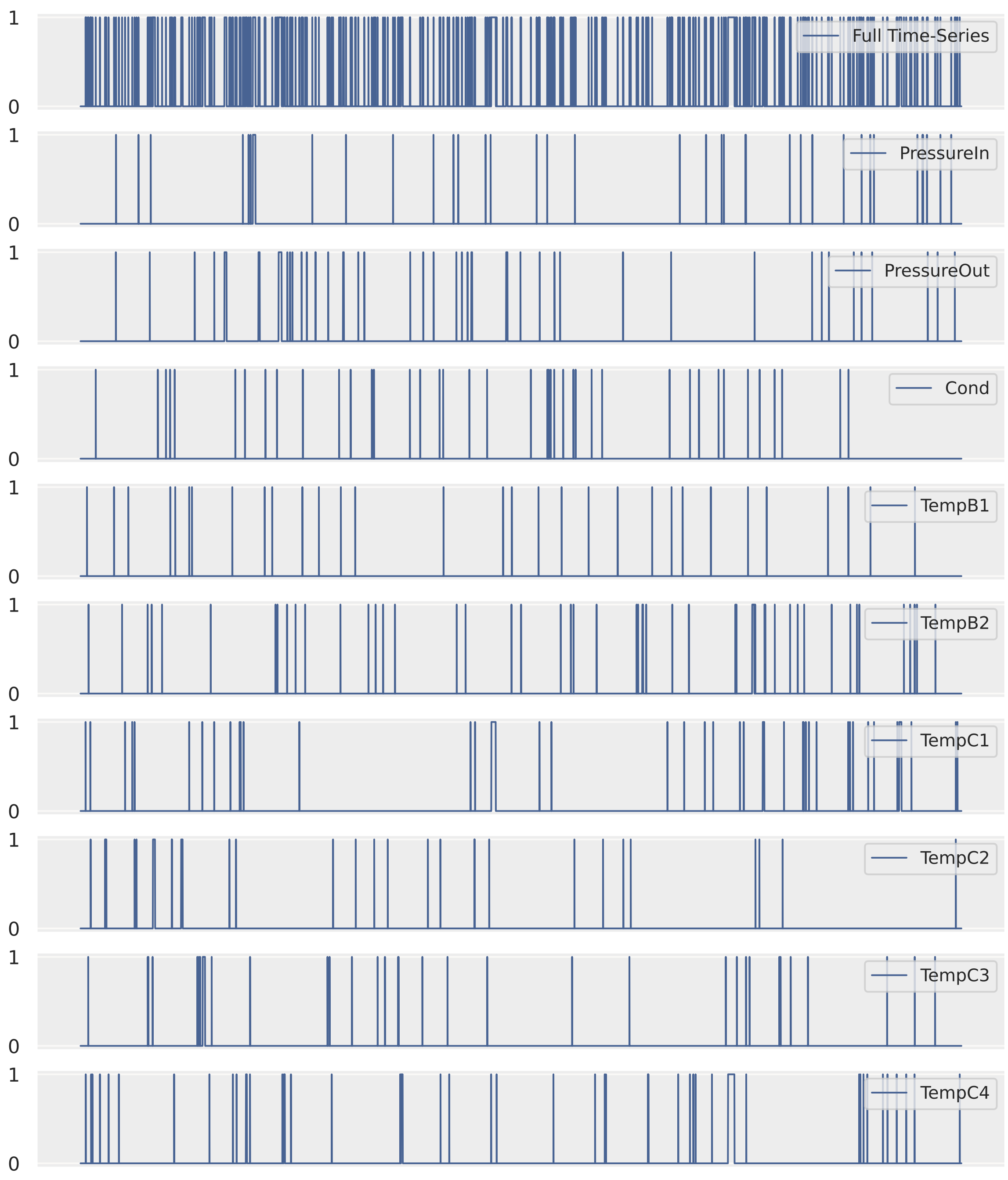


Figure C.4: Features' anomaly distribution for the pre-training dataset of the scrubber towers.

References

[1] A. Nawaz, S. Ahmed, H. A. Khattak, V. Akre, A. Rajan, and Z. A. Khan. Latest Advances in Interent Of Things and Big Data with Requirments and Taxonomy. *Seventh International Conference on Information Technology Trends*, pages 13–19, 2020. DOI: 10.1109/ITT51279.2020.9320892.

[2] A. Artemenko. Keynote: Advances and Challenges of Industrial IoT. *IEEE International Conference on Pervasive Computing and Communications Workshops and Other Affiliated Events*, pages 526–526, 2021. DOI: 10.1109/PerComWorkshops51409.2021.9431146.

[3] E. Tijan, M. Jović, S. Aksentijević, and A. Pucihar. Digital Transformation in the Maritime Transport Sector. *Technological Forecasting and Social Change*, 170:120879–120894, 2021. DOI: 10.1016/j.techfore.2021.120879.

[4] I. Lazakis, K. Dikis, A. L. Michala, and G. Theotokatos. Advanced Ship Systems Condition Monitoring for Enhanced Inspection, Maintenance and Decision Making in Ship Operations. *Transportation Research Procedia*, 14:1679–1688, 2016. DOI: 10.1016/j.trpro.2016.05.133.

[5] M. Constapel, P. Koch, and H.-C. Burmeister. On the Implementation of a Rule-Based System to Perform Assessment of COLREGs Onboard Maritime Autonomous Surface Ships. *Journal of Physics: Conference Series*, 2311, 2022. DOI: 10.1088/1742-6596/2311/1/012033.

[6] J. Han, Xu Li, and T. Tang. Energy Management Using a Rule-Based Control Strategy of Marine Current Power System with Energy Storage System. *Journal of Marine Science and Engineering*, 9, 2021. DOI: 10.3390/jmse9060669.

[7] A. B. Nassif, M. A. Talib, Q. Nasir, and F. M. Dakalbab. Machine Learning for Anomaly Detection: A Systematic Review. *IEEE Access*, 9:78658–78700, 2021. DOI: 10.1109/ACCESS.2021.3083060.

[8] A. Brandsæter, G. Manno, E. Vanem, and I. K. Glad. An Application of Sensor-Based Anomaly Detection in the Maritime Industry. *IEEE International Conference on Prognostics and Health Management*, pages 1–8, 2016. DOI: 10.1109/ICPHM.2016.7811910.

[9] A. Brandsæter, E. Vanem, and I. K. Glad. Cluster Based Anomaly Detection with Applications in the Maritime Industry. *International Conference on Sensing, Diagnostics, Prognostics, and Control*, pages 328–333, 2017. DOI: 10.1109/SDPC.2017.69.

[10] A. L. Ellefsen, P. Han, Xu Cheng, F. T. Holmeset, V. Æsøy, and H. Zhang. Online Fault Detection in Autonomous Ferries: Using Fault-Type Independent Spectral Anomaly Detection. *IEEE Transactions on Instrumentation and Measurement*, 69:8216–8225, 2020. DOI: 10.1109/TIM.2020.2994012.

[11] D. Kim, S. Lee, and J. Lee. An Ensemble-Based Approach to Anomaly Detection in Marine Engine Sensor Streams for Efficient Condition Monitoring and Analysis. *Sensors*, 20, 2020. DOI: 10.3390/s20247285.

[12] D. Kim, G. Antariksa, M. P. Handayani, S. Lee, and J. Lee. Explainable Anomaly Detection Framework for Maritime Main Engine Sensor Data. *Sensors*, 21, 2021. DOI: 10.3390/s21155200.

[13] C. Qu, Z. Zhou, Z. Liu, and S. Jia. Predictive Anomaly Detection for Marine Diesel Engine Based on Echo State Network and Autoencoder. *Energy Reports*, 8:998–1003, 2022. DOI: 10.1016/j.egyr.2022.01.225.

[14] C. Velasco-Gallego and I. Lazakis. RADIS: A Real-Time Anomaly Detection Intelligent System for Fault Diagnosis of Marine Machinery. *Expert Systems with Applications*, 204, 2022. DOI: 10.1016/j.eswa.2022.117634.

[15] P. Baraldi, F. Di Maio, P. Turati, and E. Zio. Robust signal reconstruction for condition monitoring of industrial components via a modified Auto Associative Kernel Regression method. *Systems and Signal Processing*, 60-61:29–44, 2015.

[16] S. Cheng and M. Pecht. Using cross-validation for model parameter selection of sequential probability ratio test. *Expert Systems with Applications*, 39:8467–8473, 2012. DOI: 10.1016/j.eswa.2012.01.172.

[17] M. M. Breunig, H. P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying density-based local outliers. *ACM SIGMOD Record*, 29:93–104, 2000. DOI: 10.1145/335191.335388.

[18] Y. Zhao, Z. Nasrullah, M. K. Hryniewicki, and Z. Li. LSCP: Locally selective combination in parallel outlier ensembles. *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 585–593, 2019. DOI: 10.1137/1.9781611975673.66.

[19] S. M. Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4768–4777, 2017.

[20] L. Basora, X. Olive, and T. Dubot. Recent Advances in Anomaly Detection Methods Applied to Aviation. *Aerospace*, 6, 2019. DOI: 10.3390/aerospace6110117.

[21] C. Derse, M. el Baghdadi, O. Hegazy, U. Sensoz, H. Nur Gezer, and M. Nil. An Anomaly Detection Study on Automotive Sensor Data Time Series for Vehicle Applications. *Sixteenth International Conference on Ecological Vehicles and Renewable Energies*, pages 1–5, 2021. DOI: 10.1109/EVER52347.2021.9456629.

[22] M. Ibrahim, A. Alsheikh, F. M. Awaysheh, and M. D. Alshehri. Machine Learning Schemes for Anomaly Detection in Solar Power Plants. *Energies*, 15, 2022. DOI: 10.3390/en15031082.

[23] J. W. Tukey. Comparing Individual Means in the Analysis of Variance. *Biometrics*, 5:99–114, 1949. DOI: 10.2307/3001913.

[24] F. E. Grubbs. Sample Criteria for Testing Outlying Observations. *The Annals of Mathematical Statistics*, 21:27–58, 1950. DOI: 10.1214/aoms/1177729885.

[25] P. Whittle. Hypothesis Testing in Time Series Analysis. *Journal of the Royal Statistical Society, Series A*, 114:579, 1951. DOI: 10.2307/2981095.

[26] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang. Time-Series Anomaly Detection Service at Microsoft. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3009–3017, 2019. DOI: 10.1145/3292500.3330680.

[27] A. Saju and H. N. Suresh. Object Detection in Real Time Images Using Saliency Mapping Technique. *2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies*, 1:221–226, 2019. DOI: 10.1109/ICICICT46008.2019.8993345.

[28] P. Mukherjee, B. Lall, and A. Shah. Saliency Map Based Improved Segmentation. *IEEE International Conference on Image Processing*, pages 1290–1294, 2015. DOI: 10.1109/ICIP.2015.7351008.

[29] X. Hou and Z. Liqing. Saliency Detection: A Spectral Residual Approach. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007. DOI: 10.1109/CVPR.2007.383267.

[30] G. R. Moreno, M. Niranjan, and A. Prugel-Bennett. Saliency Map on Cnns for Protein Secondary Structure Prediction. *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1249–1253, 2019. DOI: 10.1109/ICASSP.2019.8683603.

[31] Y. Iwashima, J. Wang, and Y. Yashima. Full Reference Image Quality Assessment by CNN Feature Maps and Visual Saliency. *IEEE 8th Global Conference on Consumer Electronics*, pages 203–207, 2019. DOI: 10.1109/GCCE46687.2019.9015318.

[32] W. Qinglan. Saliency Maps-Based Convolutional Neural Networks for Facial Expression Recognition. *IEEE Access*, 9:76224–76234, 2021. DOI: 10.1109/ACCESS.2021.3082694.

[33] P. W. Anderson. More Is Different: Broken Symmetry and the Nature of the Hierarchical Structure of Science. *Science*, 177:393–396, 1972. DOI: 10.1126/science.177.4047.393.

[34] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff. LSTM-Based Encoder-Decoder for Multi-Sensor Anomaly Detection. *ICML Anomaly Detection Workshop*, 2016. DOI: 10.48550/arXiv.1607.00148.

[35] Q. Ding, S. Liu, B. Zhou, H. Shen, and X. Cheng. Multi-Scale Anomaly Detection for Big Time Series of Industrial Sensors. *arXiv pre-print*, 2022. DOI: 10.48550/arXiv.2204.08159.

[36] T.-A. Pham, J.-H. Lee, and Park C.-S. MST-VAE: Multi-Scale Temporal Variational Autoencoder for Anomaly Detection in Multivariate Time Series. *Applied Sciences*, 12:10078, 2022. DOI: 10.3390/app121910078.

[37] A. Sperduti and A. Starita. Supervised Neural Networks for the Classification of Structures. *IEEE Transactions on Neural Networks*, 8:714–735, 1997. DOI: 10.1109/72.572108.

[38] M. Gori, G. Monfardini, and F. Scarselli. A New Model for Learning in Graph Domains. *IEEE International Joint Conference on Neural Networks*, 2:729–734, 2005. DOI: 10.1109/IJCNN.2005.1555942.

[39] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and Monfardini G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20:61–80, 2009. DOI: 10.1109/TNN.2008.2005605.

[40] C. Gallicchio and M. Alessio. Graph Echo State Networks. *International Joint Conference on Neural Networks*, pages 1–8, 2010. DOI: 10.1109/IJCNN.2010.5596796.

[41] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32:4–24, 2021. DOI: 10.1109/TNNLS.2020.2978386.

[42] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel. Gated Graph Sequence Neural Networks. In *4th International Conference on Learning Representations*, 2016. DOI: 10.48550/arXiv.1511.05493.

[43] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How Powerful are Graph Neural Networks? In *7th International Conference on Learning Representations*, 2019. DOI: 10.48550/arXiv.1810.00826.

[44] T. Guo, J. Dong, H. Li, and Y. Gao. Simple Convolutional Neural Network on Image Classification. *IEEE 2nd International Conference on Big Data Analysis*, pages 721–724, 2017. DOI: 10.1109/ICBDA.2017.8078730.

[45] S. P. Singh, A. Kumar, H. Darbari, L. Singh, A. Rastogi, and S. Jain. Machine Translation Using Deep Learning: An Overview. *International Conference on Computer, Communications and Electronics*, pages 162–167, 2017. DOI: 10.1109/COMPTELIX.2017.8003957.

[46] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral Networks and Locally Connected Networks on Graphs. In *2nd International Conference on Learning Representations*, 2014. DOI: 10.48550/arXiv.1312.6203.

[47] M. Henaff, J. Bruna, and LeCun Y. Deep convolutional networks on graph-structured data. *CoRR*, 2015. DOI: 10.48550/arXiv.1506.0516.

[48] T. N. Kipf and M. Welling. Semi-Supervised classification with graph convolutional networks. *CoRR*, 2016. DOI: 10.48550/arXiv.1609.02907.

[49] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 3844–3852, 2016.

[50] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein. CayleyNets: Graph Convolutional Neural Networks With Complex Rational Spectral Filters. *IEEE Transactions on Signal Processing*, 67: 97–109, 2019. DOI: 10.1109/TSP.2018.2879624.

[51] S. Wan, C. Gong, P. Zhong, Bo Du, L. Zhang, and J. Yang. Multiscale Dynamic Graph Convolutional Network for Hyperspectral Image Classification. *IEEE Transactions on Geoscience and Remote Sensing*, 58:3162–3177, 2020. DOI: 10.1109/TGRS.2019.2949180.

[52] S. Asif and S. Sumitra. Spectral Graph Convolutional Neural Networks in the Context of Regularization Theory. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–12, 2022. DOI: 10.1109/TNNLS.2022.3177742.

[53] A. Micheli. Neural Network for Graphs: A Contextual Constructive Approach. *IEEE Transactions on Neural Networks*, 20:498–511, 2009. DOI: 10.1109/TNN.2008.2010350.

[54] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

[55] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive Representation Learning on Large Graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[56] D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations*, 2015.

[57] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[58] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. *CoRR*, 2017. DOI: 10.48550/arXiv.1710.10903.

[59] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research*, 15:1929–1958, 2014.

[60] Z. Sun, A. Harit, J. Yu, A. I. Cristea, and N. Al Moubayed. A Generative Bayesian Graph Attention Network for Semi-Supervised Classification on Scarce Data. *International Joint Conference on Neural Networks*, pages 1–7, 2021. DOI: 10.1109/IJCNN52387.2021.9533981.

[61] B. Gaudel, D. Guan, W. Yuan, D. Shrestha, B. Chen, and Y. Tu. A Generative Bayesian Graph Attention Network for Semi-Supervised Classification on Scarce Data. *Big Data*, pages 137–147, 2021. DOI: 10.1007/978-981-16-0705-9_10.

[62] Bo Jiang, D. Lin, J. Tang, and B. Luo. Data Representation and Learning With Graph Diffusion-Embedding Networks. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10406–10415, 2019. DOI: 10.1109/CVPR.2019.01066.

[63] W. Jingyi and Z. Deng. A Deep Graph Wavelet Convolutional Neural Network for Semi-Supervised Node Classification. *International Joint Conference on Neural Networks*, pages 1–8, 2021. DOI: 10.1109/IJCNN52387.2021.9533634.

[64] Y. Zhu, J. Wang, J. Zhang, and K. Zhang. Node Embedding and Classification with Adaptive Structural Fingerprint. *Neurocomputing*, 502:196–208, 2022. DOI: 10.1016/j.neucom.2022.05.073.

[65] S. Brody, U. Alon, and E. Yahav. How Attentive are Graph Attention Networks? In *10th International Conference on Learning Representations*, 2022. DOI: 10.48550/arXiv.2105.14491.

[66] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2:303–314, 1989. DOI: 10.1007/BF02551274.

[67] D. E. Rumelhart and J. L. McClelland. Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, chapter 8, pages 318–362. MIT Press, 1987.

[68] M. I. Jordan. Serial Order: A Parallel Distributed Processing Approach. *Advances in Psychology*, 121:471–495, 1997. DOI: 10.1016/S0166-4115(97)80111-2.

[69] Y. Bengio, P. Simard, and P. Frasconi. Learning Long-Term Dependencies with Gradient Descent Is Difficult. *IEEE Transactions on Neural Networks*, 5:157–166, 1994. DOI: 10.1109/72.279181.

[70] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *NIPS Workshop on Deep Learning*, 2014. DOI: 10.48550/arXiv.1412.3555.

[71] J. Martens and I. Sutskever. Learning Recurrent Neural Networks with Hessian-Free Optimization. *Proceedings of the 28th International Conference on Machine Learning*, pages 1033–1040, 2011.

[72] A. Graves. Generating Sequences With Recurrent Neural Networks. *arXiv pre-print*, 2014. DOI: 10.48550/arXiv.1308.0850.

[73] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, 2014. DOI: 10.3115/v1/W14-4012.

[74] L. Yao and G. Yazhuo. An Improved LSTM Structure for Natural Language Processing. *IEEE International Conference of Safety Produce Informatization*, pages 565–569, 2018. DOI: 10.1109/IICSPI.2018.8690387.

[75] N. Habbat, H. Anoun, and L. Hassouni. Combination of GRU and CNN Deep Learning Models for Sentiment Analysis on French Customer Reviews Using XLNet Model. *IEEE Engineering Management Review*, pages 1–9, 2022. DOI: 10.1109/EMR.2022.3208818.

[76] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio. Light Gated Recurrent Units for Speech Recognition. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2:92–102, 2018. DOI: 10.1109/TETCI.2017.2762739.

[77] G. Tiwari, A. Sharma, A. Sahotra, and R. Kapoor. English-Hindi Neural Machine Translation-LSTM Seq2Seq and ConvS2S. *International Conference on Communication and Signal Processing*, pages 871–875, 2020. DOI: 10.1109/ICCSP48568.2020.9182117.

[78] J. Singh, S. Sharma, and J. Briskilal. Natural Language Processing Based Machine Translation for Hindi-English Using GRU and Attention. *International Conference on Applied Artificial Intelligence and Computing*, pages 965–969, 2022. DOI: 10.1109/ICAAIC53929.2022.9793214.

[79] R. Jozefowicz, W. Zaremba, and I. Sutskever. An Empirical Exploration of Recurrent Network Architectures. *Proceedings of the 32nd International Conference on Machine Learning*, pages 2342–2350, 2015.

[80] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *arXiv pre-print*, 2013. DOI: 10.48550/arXiv.1312.6114.

[81] D. P. Kingma and M. Welling. An Introduction to Variational Autoencoders. *Foundations and Trends in Machine Learning*, 12:307–392, 2019. DOI: 10.1561/2200000056.

[82] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313:504–507, 2006. DOI: 10.1126/science.1127647.

[83] Q. Fournier and D. Aloise. Empirical Comparison between Autoencoders and Traditional Dimensionality Reduction Methods. *IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering*, pages 211–214, 2019. DOI: 10.1109/AIKE.2019.00044.

[84] Q. Li and C. Yang. Learning to Compress Using Deep AutoEncoder. *57th Annual Allerton Conference on Communication, Control, and Computing*, pages 930–936, 2019. DOI: 10.1109/ALLERTON.2019.8919866.

[85] S. Ghaffarzadegan, H. Bořil, and J. H. L. Hansen. Generative Modeling of Pseudo-Whisper for Robust Whispered Speech Recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24:1705–1720, 2016. DOI: 10.1109/TASLP.2016.2580944.

[86] L. Girin, S. Leglaive, X. Bie, J. Diard, T. Hueber, and X. Alameda-Pineda. Dynamical Variational Autoencoders: A Comprehensive Review. *Foundations and Trends in Machine Learning*, 15:1–175, 2021. DOI: 10.1561/2200000089.

[87] A. Asperti, D. Evangelista, and E. Loli Piccolomini. A Survey on Variational Autoencoders from a Green AI Perspective. *SN Computer Science*, 2, 2021. DOI: 10.1007/s42979-021-00702-9.

[88] L. Girin, F. Roche, T. Hueber, and S. Leglaive. Notes on the use of variational autoencoders for speech and audio spectrogram modeling. *22nd International Conference on Digital Audio Effects*, pages 1–8, 2019.

[89] R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. *Learning in Graphical Models*, pages 355–368, 1998. DOI: 10.1007/978-94-011-5014-9_12.

[90] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37:183–233, 1999. DOI: 10.1023/A:1007665907178.

[91] H. Zhao, Y. Wang, J. Duan, C. Huang, D. Cao, Y. Tong, B. Xu, J. Bai, J. Tong, and Qi Zhang. Multivariate Time-Series Anomaly Detection via Graph Attention Network. *IEEE International Conference on Data Mining*, pages 841–850, 2020. DOI: 10.1109/ICDM50108.2020.00093.

[92] A. Siffer, P.-A. Fouque, A. Termier, and C. Largouet. Anomaly Detection in Streams with Extreme Value Theory. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1067–1075, 2017. DOI: 10.1145/3097983.3098144.

[93] A. Siffer. OmniAnomaly. `https://github.com/NetManAIOps/OmniAnomaly`, 2016.

[94] N. P. DeGuglielmom, S. M. S. Basnet, and D. E. Dow. Introduce Ladder Logic and Programmable Logic Controller (PLC). *Annual Conference Northeast Section*, pages 1–5, 2020. DOI: 10.1109/ASEENE51624.2020.9292646.

[95] D.-A. Lee, J. Yoo, and J.-S. Lee. Guidelines for the Use of Function Block Diagram in Reactor Protection Systems. *21st Asia-Pacific Software Engineering Conference*, 1:135–142, 2014. DOI: 10.1109/APSEC.2014.29.

[96] M. Kocánek and P. Balda. General Sequential Function Charts Editor. *12th International Carpathian Control Conference*, pages 191–194, 2011. DOI: 10.1109/CarpathianCC.2011.5945845.

[97] Azure IoT Hub. `https://learn.microsoft.com/en-us/azure/iot-hub/`, 2016. Accessed: 2023-01-16.

[98] P. Wenig, S. Schmidl, and T. Papenbrock. TimeEval: A Benchmarking Toolkit for Time Series Anomaly Detection Algorithms. *Proceedings of the VLDB Endowment*, 15(12):3678–3681, 2022. DOI: 10.14778/3554821.3554873.

[99] Microsoft Azure. `https://azure.microsoft.com/`, 2008. Accessed: 2023-02-16.

[100] Databricks. `https://www.databricks.com/`, 2013. Accessed: 2023-02-16.

[101] Azure Databricks. `https://learn.microsoft.com/en-us/azure/databricks/`, 2018. Accessed: 2023-02-16.

[102] Azure Monitor. `https://learn.microsoft.com/en-us/azure/azure-monitor/`, 2018. Accessed: 2023-02-16.

[103] Azure AD. `https://learn.microsoft.com/en-us/azure/active-directory/`, 2008. Accessed: 2023-02-16.

[104] Azure Key Vault. `https://learn.microsoft.com/en-us/azure/key-vault/`, 2015. Accessed: 2023-02-16.

[105] Azure Stream Analytics. `https://learn.microsoft.com/en-us/azure/stream-analytics/`, 2015. Accessed: 2023-02-16.

[106] Azure IoT Hub Device Provisioning Service. `https://learn.microsoft.com/en-us/azure/iot-dps/`, 2017. Accessed: 2023-02-16.

[107] Azure Blob Storage. `https://learn.microsoft.com/en-us/azure/storage/blobs/`, 2008. Accessed: 2023-02-16.

[108] Azure Data Lake Storage Gen 2. `https://learn.microsoft.com/en-us/azure/storage/blobs/data-lake-storage-introduction`, 2018. Accessed: 2023-02-16.

[109] Delta Lake. `https://docs.delta.io/latest/index.html`, 2017. Accessed: 2023-02-16.

[110] Docker. `https://www.docker.com/`, 2013. Accessed: 2023-02-16.

[111] Azure Kubernetes Service. https://learn.microsoft.com/en-us/azure/aks/, 2018. Accessed: 2023-02-16.

[112] FastAPI. https://fastapi.tiangolo.com/, 2018. Accessed: 2023-02-16.

[113] PostgreSQL. https://www.postgresql.org/, 1986. Accessed: 2023-02-16.

[114] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, Lu Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. https://dl.acm.org/doi/10.5555/3454287.3455008, 2019.

[115] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 448–456, 2015. DOI: 10.48550/arXiv.1502.03167.

[116] G. Raskutti, M. J. Wainwright, and B. Yu. Early Stopping for Non-Parametric Regression: An Optimal Data-Dependent Stopping Rule. *49th Annual Allerton Conference on Communication, Control, and Computing*, pages 1318–1325, 2011. DOI: 10.1109/Allerton.2011.6120320.

[117] W. Kvaale and A. Øvrebø Harstad. ML4ITS - Machine Learning for Irregular Time Series. https://github.com/ML4ITS/mtad-gat-pytorch, 2021.

[118] M. A. Wani and S. Afzal. A New Framework for Fine Tuning of Deep Networks. *16th IEEE International Conference on Machine Learning and Applications*, pages 359–363, 2017. DOI: 10.1109/ICMLA.2017.0-135.

[119] A. Ramdan, A. Heryana, A. Arisal, R. B. S. Kusumo, and H. F. Pardede. Transfer Learning and Fine-Tuning for Deep Learning-Based Tea Diseases Detection on Small Datasets. *International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications*, pages 206–211, 2020. DOI: 10.1109/ICRAMET51080.2020.9298575.

[120] Z. Cai and C. Peng. A Study on Training Fine-Tuning of Convolutional Neural Networks. *13th International Conference on Knowledge and Smart Technology*, pages 84–89, 2021. DOI: 10.1109/KST51265.2021.9415793.

[121] T. Fredriksson, D. I. Mattos, J. Bosch, and H. H. Olsson. An Empirical Evaluation of Algorithms for Data Labeling. *IEEE 45th Annual Computers, Software, and Applications Conference*, pages 201–209, 2021. DOI: 10.1109/COMPSAC51774.2021.00038.

[122] L. Xu, K. Xu, Y. Qin, Y. Li, X. Huang, Z. Lin, N. Ye, and X. Ji. TGAN-AD: Transformer-Based GAN for Anomaly Detection of Time Series Data. *Applied Sciences*, 12, 2022. DOI: 10.3390/app12168085.

[123] A. Chen, A. Chow, A. Davidson, A. DCunha, A. Ghodsi, S. A. Hong, A. Konwinski, C. Mewald, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, A. Singh, F. Xie, M. Zaharia, R. Zang, J. Zheng, and C. Zumar. Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle. *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*, pages 1–4, 2020. DOI: 10.1145/3399579.3399867.