Deep learning-based multi-sensor fusion for industrial process monitoring

Deep learning-based multi-sensor fusion for industrial process monitoring

A dissertation submitted to

University of Sheffield

Department of Automatic Control and Systems Engineering

in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

2nd June 2023

The University of Sheffield

Department of Automatic Control and Systems Engineering

Amy Johnson Building

Portobello Street

Sheffield, S1 3JD

"There is only one true heroism in the world: to see the world as it is, and to love it"

Roman Rolland

Acknowledgement

First and foremost, I would like to express my deepest gratitude to my supervisor, Professor Ashutosh Tiwari. Your guidance, patience, and unwavering belief in my abilities have been invaluable throughout this journey. Each discussion with you has always been a treasure trove of knowledge, opening my eyes to new perspectives and methodologies, and offering insights and constructive criticisms that pushed me to a level I did not think possible. You inspired me to remain resilient in the face of adversity, a lesson I will carry with me beyond this academic milestone. I am fortunate to have had you as my guide in this monumental phase of my academic life.

To my dearest father, Zhang Jianzhong, and mother, Chen Jv, who have provided me with continuous love, encouragement, and support throughout my entire life, including the years spent on this PhD journey. Your unconditional love and belief in me provided the strength I needed during my toughest times. Thank you for the countless sacrifices you made and for always putting my needs ahead of yours. I hope that this achievement serves not only as a testament to my efforts but more importantly, to your successful parenting. The journey to this PhD was long and winding, and I could not have navigated it without you.

Great thanks to Dr Michael Farnsworth, Dr Divya Tiwari, and Dr Boyang Song, who are ready to help all the time, providing me with meaningful guidance and support. Big thanks to all the nice colleagues of Professor Ashutosh Tiwari's group. It is my great pleasure to be a member of you.

Lastly, I would like to express my sincere appreciation to my friends and everyone I met in the UK. Thank you for your company and it is a pleasure to have the time with you.

Declaration

I declare that the work presented in this thesis is my own. All material in this thesis which is not of my own work, has been properly accredited and referenced.

Sheffield, 2 Jun 2023

Abstract

Modern industrial processes rapidly evolve due to advanced technologies. However, long-term operation often leads to faults or anomalies originating from ageing and stochastic factors, which can cost manufacturers up to 70% of their production expenditures on maintenance. Effective process monitoring can be crucial in preventing accidents, ensuring productivity, and preserving product quality. A considerable body of research has illustrated that multi-sensor fusion can be important for monitoring complex processes characterised by non-linear, dynamic, and multi-modal properties.

Compared to traditional sensor fusion, deep learning-based fusion has the advantages of being hypothesis-independent, allowing automatic feature extraction, having a high upper complexity limit of the object being modelled, and not relying on deep domain knowledge, making it an emerging direction. However, its inherent properties, such as data hunger, black box, and high computational complexity, limit its development in industrial data processing. Therefore, this thesis focuses on the research of the following three aspects. Firstly, what deep learning structure can be suitable for sensor fusion. Secondly, how to alleviate the data hunger and black box nature of deep learning considering the complexities associated with industrial data acquisition and the need for interpretability. Thirdly, how to optimise the computational consumption of deep learning to save limited industrial computational resources.

This thesis first identifies the advantages of the attention mechanism for sensor fusion through a literature review and then proposed a novel dual-channel attention model for sensor anomaly detection. This model was validated on a public dataset and the results show that it outperforms the existing methods in the literature working on this dataset in a number of metrics. However, the models based on attention mechanisms often require a large amount of data to be used to their advantage, which can be difficult for industrial applications. Therefore, a novel transfer learning approach was proposed to reduce the required amount of data. This research first embeds sensor data into an embedding space and then transfers a pre-trained natural language model to process the sensor data, thus allowing even limited sensor data to benefit from the learning capabilities of a large-depth model. The method was validated on three public datasets and the results show that the method can achieve high performance with a smaller amount of data. In addition, this research also shows that attention mechanisms can be potentially useful for improving the interpretability of deep learning. Finally, as the high computational load of the self-attention mechanism was observed, a deep learning architecture specifically for processing multi-sensor data was proposed for optimising the computational resource occupation based on the Fast Fourier Transform and the self-attention mechanism. The results show a significant improvement in both memory usage and inference speed.

Contents

List of Figures

List of Tables

Glossary

AI	Artificial Intelligence
AM	Additive Manufacturing
ANN	Artificial Neural Network
BN	Batch Normalisation
BPA	Basic Probability Assignment
BSM	Basic Safety Messages
CAVs	Connected and Automated Vehicles
CCA	Canonical Correlation Analysis
CL	Center Line
CNN	Convolutional Neural Network
CPS	Cyber Physical System
CPS	Cyber-Physical Systems

CUSUM	Cumulative Sum
D-S	Dempster-Shafer
DAI-DAO	Data in-Data out
DAI-FEO	Data in-Feature out
DAM	Dual-channel Attention Mechanism
DAS	Data Acquisition System
DEI-DEO	Decision in-Decision out
DFT	Discrete Fourier Transform
EWMA	Exponentially Weighted Moving Average
FEI-DEO	Feature in-Decision out
FEI-FEO	Feature in-Feature out
FFT	Fast Fourier Transform
FL	Fuzzy Logic
GNB	Gaussian Naive Bayes
GPS	Global Positioning System
GPT	Pre-trained Transformer
GPU	Graphic Processing Unit
GRU	Gated Recurrent Unit
HI	Health Index
ICA	Independent Component Analysis

IIoT	Industrial Internet of Thing
IPC	Industrial Personal Computers
JDL	Joint Directors of Laboratory
KNN	K-Nearest Neighbor
LCL	Lower Control Limit
LDA	Linear Discriminant Analysis
LIDAR	Light Detection and Ranging
LN	Layer Normalisation
LSTM	Long Short-Term Memory
MSPM	Multivariate Statistical Process Monitoring
NCA	Neighborhood Component Analysis
NLP	Natural Language Processing
PCA	Principal Component Analysis
PLS	Partial Least Squares Regression
PLS	Partial Least Squares
RDE	Research Data Exchange
ReLU	Rectified Linear Unit
ResNets	Residual Nets
RNN	Recurrent Neural Network
RUL	Remaining Useful Life

SGD	Stochastic Gradient Descent
SNR	Signal-to-Noise Ratio
SOTA	State of The Art
SPC	Statistical Process Control
SPMD	Safety Pilot Model Deployment
SVD	Singular Value Decomposition
TPU	Tensor Processing Unit
TSA	Time Series Analysis
TSC	Time Series Classification
UCL	Upper Control Limit
ViT	Vision Transformer

1

Introduction

1.1 Motivation: The Importance of Industrial Process Monitoring

Modern industrial processes are evolving in a complex and extensive manner due to the rapid development of advanced technologies and the rising demand for high-quality, high-performance, and complex products on the global market. However, regardless of how reliable the design of an industrial process is, faults or anomalies are inevitable over long periods of operation due to the ageing of the system and a variety of unpredictable random factors [1], especially for complex industrial processes. The diagnosis and maintenance of industrial processes can be labour-intensive and time-consuming, and they usually cause interruptions in the production process, which add significantly to the cost and lower efficiency. According to [2], manufacturing companies spend between 15% and 70% of their whole production expenses on maintenance.

Moreover, failure to detect anomalies not only affects productivity and production quality but may also lead to serious accidents over time. For example, the explosion of the Mina Al-Ahmadi oil refinery caused by a fault in a condensate line caused five deaths and the factory was destroyed by this accident [3]. In 2010, the Deepwater Horizon disaster released 5.3×10^{11}g of oil and 1.7×10^{11}g of natural gas into the ocean because of the failure of a critical valve, making it the largest accidental release [4]. This accident not only caused huge economic damages and loss of life but also had a long-term impact on the marine environment [4]. Therefore, effective and reliable process monitoring technologies that can detect and track the arising defects or anomalies have attracted significant interest from both researchers and the industry.

1.2 The Role of Sensor Fusion in Process Monitoring

In modern industrial processes, the underlying non-linear, dynamic, multi-modal, space-time complex and high-dimension natures are usually observed [5], which renders them complex systems. For example, in the machining process, the product quality is the result of a combination of various physical processes and material properties, such as plastic deformation, chip formation, structural homogeneity and rigidity of material etc., making it a highly nonlinear and dynamic process [6]. Numerous studies have been conducted to figure out the correlation between single sensor information (such as vibration and current) and product quality or tool conditions [7] [8] [9], but gradually researchers realised that the process monitoring based on a single sensor or process parameter can be far from sufficient to handle its complexity. Hence, the research focus of this domain has been moved to the multi-sensor system that monitors different physical quantities at different locations [10] [11] [12], as combining information from multiple sensors usually provides a more comprehensive recognition, and led to better performance. The aforementioned phenomena are usually seen in industrial process monitoring scenarios. Consequently, sensor fusion is gradually growing in significance in this research area.

Sensor fusion refers to the integration and analysis of data and information from multiple sensors by combining redundant or complementary spatial or temporal information to provide a more realistic and reliable representation of a given system or object. Compared with the use of a single sensor, a multi-sensor system has the following advantages [13]:

- Representation: In contrast to every single source, a multi-sensor system usually has a higher level of abstraction, enabling a better semantic. This facilitates the distillation of useful information from a large amount of data obtained from industrial processes.
- Certainty: The confidence and signal-to-noise ratio of data can be improved by combining the redundant information from multiple sensors that monitor the same object or process, thus increasing the accuracy and reducing the impact of single sensor failure and uncertainty on the system. This feature is critical for dealing with complex industrial environments such as dust contamination, changing workloads, tough working conditions, and long-term working in industrial scenarios.
- Completeness: If new knowledge of the object being monitored can be introduced by different sensors, a more comprehensive view can be provided by organising these sensors into a consistent space, which is preferable for handling the inherent complexity of industrial processes.

Therefore, sensor fusion technology is receiving increasing attention in modern industry.

1.3 Research Challenges of Sensor Fusion in Process Monitoring

Sensor fusion faces many challenges when applied to industrial process monitoring, as in other applications. The challenges are summarised in table 1.1 [14–17]. For any fusion algorithm, it is very difficult to solve all the problems mentioned in this table simultaneously. When designing a sensor fusion system for process monitoring, the most critical challenges should be considered and identified, depending on the characteristics and requirements of the specific tasks. The common challenges in the view of industrial process monitoring are listed below.

Sensor anomalies. Sensors are an important foundation for monitoring systems, as any monitoring and analysis algorithm relies heavily on sensor readings to obtain information about the conditions of the system. The distortion of sensor data will lead to a reduction in the amount of useful information, and even introduce false information, which may lead to system degradation and even catastrophic failure. However, the sensor readings can be anomalies due to many reasons. For example, the harsh industrial environment (e.g., high workloads, dust contamination, strong vibration and long operating hours) and the malicious attack [18].

Data correlation. This issue is very common for a process monitoring system due to some process parameters of industrial processes can be highly correlated with each other and system designers usually employ redundant sensors mounted on distributed locations to monitor the same parameter. In addition, the same noise source may bias different sensors simultaneously, making them numerically correlated. This kind of dependency should be addressed properly, otherwise, the fusion algorithm might be biased (over or under confidence) on the one hand [15], and on the other hand, the computational resources may be wasted.

Multiple modalities. In order to achieve a comprehensive perception of industrial processes, it is important to obtain data from different modalities which might be significantly different in terms of characteristics, sampling frequency, data structure, etc., such as vibration, vision, and pressure. How to fuse the different modalities and extract knowledge from them can be an essential and challenging problem.

Table 1.1.: Challenges of Sensor Fusion

Categories	Challenges	Descriptions
Data -related	Data imperfection	Distortion of data in the representation of real objects.
	Data confliction	Opposite representation when representing the same object.
	Data correlation	Dependencies between different sensors.
	Data association	How to link data to different targets based on specific tasks.
	Data alignment	How to align the different types of data to a unified framework.
Sensor -related	Time scales	Sensors may vary in synchronisation, response time, and sampling rates.
	Sensor uncertainty	Sensor readings may be uncertain due to noise and ambiguity.
	Sensor anomalies	Outliers may be observed or the sensors may be in a faulty state.
	Accuracy	The accuracy of sensors has to be sufficient to fulfil the requirements of tasks.
Algorithm -related	Fusion of hard and soft data	The algorithms for the fusion of sensor data and human-generated data.
	Multiple modalities	Different modalities such as vision and vibration may be monitored on the same object.
	Granularity	A huge difference can be observed in the level of details from different modalities (such as vibration and temperature).
	Architecture design	Centralised or distributed fusion architecture.
	Real-time performance	The trade-off between accuracy and inference speed of fusion algorithm.

Granularity. This term represents the difference in the level of detail from different modalities. For example, the vibration data with a very high sampling rate (around tens of kilohertz) can be much denser compared with other spares data, such as rotation speed and temperature. However, it may be only a few specific patterns inside of the vibration signal that really contributes to the information increment. The difference between the density of information and data should also be addressed by the sensor fusion algorithms [19].

Real-time performance. It is another important challenge in process monitoring. If some faults in industrial processes are not spotted and addressed immediately, they might quickly develop into failures. Considering the large and complex input space of industrial monitoring systems, achieving the desired real-time performance is not an easy task. Additionally, computational resources in industrial scenarios are usually limited, which places higher demands on fusion algorithms.

1.4 Research Scope of This Dissertation

Industrial process monitoring can be divided into model-based, knowledge-based, and data-based process monitoring [20], which will be further discussed in the following chapter. In this dissertation, the research focus will be data-based process monitoring and deep-learning-based sensor fusion algorithms will be developed for the following reasons:

- Firstly, in modern industries, the availability of process data has been enhanced significantly by the advanced infrastructure for communication and computing [21], such as the Industrial Internet of Things (IIoT), cloud/edge computing, and high-performance graphic/tensor processing units (GPU/TPU). This provides a solid foundation for the data-driven approach.

- Secondly, the data-based method does not require a deep understanding of the physical details of the process. This feature is preferred by modern industries because of the increasing complexity of industrial processes and the difficulties of modelling these processes physically.

1.5 Aims and Objectives

The overarching aim of this research is to adapt existing deep learning architecture and develop novel deep learning architecture for the sensor fusion domain for industrial process monitoring scenarios. This will be achieved through the following list of objectives:

a) Objective 1: To develop a deep-learning-based reliable sensor anomaly detection algorithm to discover the anomaly values in a multi-sensor data stream.

b) Objective 2: To develop a deep-learning-based fusion algorithm for combining significantly different modalities for industrial process monitoring.

c) Objective 3: To develop a transfer-learning-based method to mitigate the data-hungry and black-box nature of deep learning for sensor fusion in industrial process monitoring.

d) Objective 4: To reduce the computational resource consumption of the developed algorithms, benefiting industrial usage.

1.6 Research Methodology

The overall research methodology of this thesis can be illustrated in Figure 1.1. This dissertation contributes to the industrial in-process monitoring domain in three aspects, namely data reliability (Chapter 3), algorithm effectiveness (Chapter 4), and computational efficiency (Chapter 5), as these can be three important indicators for evaluating the performance of a monitoring system.

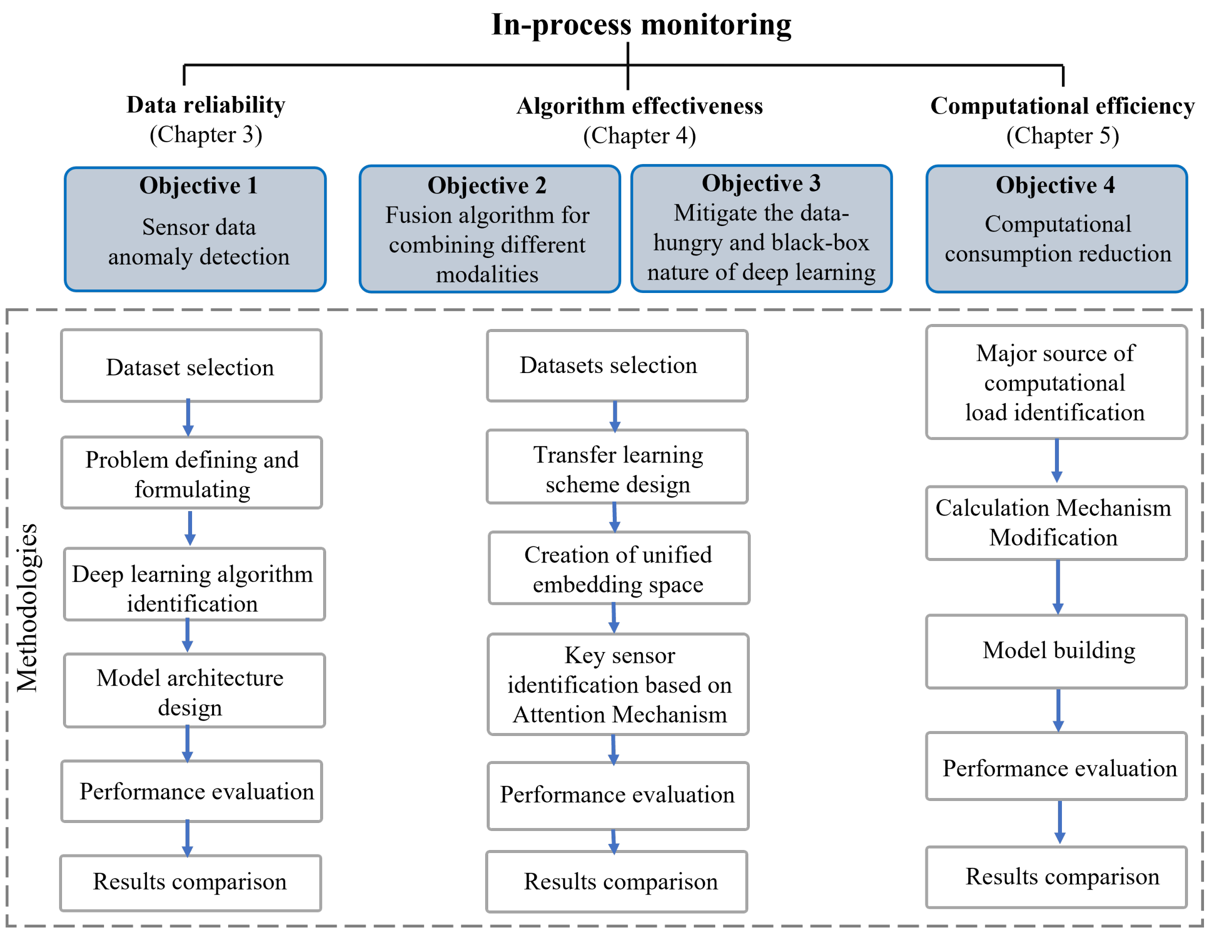

In-process monitoring
Data reliability
(Chapter 3)
Algorithm effectiveness
(Chapter 4)
Computational efficiency
(Chapter 5)
Objective 1
Sensor data anomaly detection
Objective 2
Fusion algorithm for combining different modalities
Objective 3
Mitigate the data-hungry and black-box nature of deep learning
Objective 4
Computational consumption reduction
Methodologies
Dataset selection
Problem defining and formulating
Deep learning algorithm identification
Model architecture design
Performance evaluation
Results comparison
Datasets selection
Transfer learning scheme design
Creation of unified embedding space
Key sensor identification based on Attention Mechanism
Performance evaluation
Results comparison
Major source of computational load identification
Calculation Mechanism Modification
Model building
Performance evaluation
Results comparison

Figure 1.1.: Research methodology

Objective 1: As shown in Figure 1.1, for achieving Objective 1 Sensor data anomaly detection, the first step was selecting a sensor anomaly detection dataset, followed by problem-defining and -formulating. Then, among the prevalent deep learning algorithms, e.g. Convolutional Neural Networks (CNN), Artificial Neural Networks (ANN), Recurrent Neural Networks (RNN), etc., which deep learning algorithm can be appropriate should be identified, before designing the deep learning model architecture. Finally, the proposed model was evaluated and compared with the other methods in the literature to verify its effectiveness.

The multiple time series data of Connected and Automated Vehicles (CAVs) provided by the Safety Pilot Model Deployment Data repository [22] were chosen for this research. This is because this dataset was composed of multi-sensor time series and obtained under real CAV working conditions, and its representativeness was widely recognised. In addition, two high-quality publications in recent years have used this dataset, providing good baselines for model performance comparisons. The sensor anomaly detection task was formulated as a multi-time series classification problem for a non-stationary process here, due to the driving behaviour measured by multiple sensors was not controlled in this dataset and the developed model was expected to provide an output on whether the current sensor data present anomalies. The problem definition provides important guidance for the design of the model and the details will be presented in Chapter 3. In terms of deep learning algorithms, the self-attention mechanism of the Transformer was identified as a major building block for the overall architecture via a literature review, since it has advantages for spatiotemporal feature modelling, which can be essential for anomaly detection in multiple sensor streams. The evaluation of the proposed model was kept the same as the publications working on the same dataset to keep comparability.

Objective 2 and 3: Establishing a unified feature representation for different modalities, as well as mitigating the data-hunger and black-box nature of deep learning, are important aspects of improving the effectiveness of data-driven industrial process monitoring algorithms. Due to the end-to-end model design, these two objectives can be achieved simultaneously. As shown in Figure 1.1, the first step for achieving the two objectives was selecting datasets of industrial processes containing different modalities, followed by a transfer learning scheme design used for reducing the required training data volume. The identification of the proper deep learning model was also conducted in this step. Next, an embedding method was developed to combine different modalities and how to identify the key sensors was investigated to improve the interpretability. Finally, the proposed model was evaluated and compared with the other methods in the literature to test its performance and verify its effectiveness.

The main dataset used in this research was Condition Monitoring of a Hydraulic

System [23]. This is because this dataset contains 17 different sensors measuring different modalities with different sampling rates, which can be suitable for testing the proposed method. Two additional datasets, the bearing and gearbox datasets were also applied here to verify the generalisability. These three datasets were widely used in the literature, providing a variety of baselines to compare with. In this research, the self-attention mechanism was used since it was identified as an effective algorithm in the literature review and previous research. In terms of transfer learning scheme design, the pre-trained natural language model, Generative Pre-trained Transformer 2 (GPT-2), was used under the assumption that multi-sensor problems can be similar to natural language processing. GPT-2 is an open-source model that can be fully run locally. The latest GPT-3/4 can be more powerful, but they are held on a third-party server which might be unacceptable by industrial applications due to data security considerations. The reason for not using the prevailing transfer learning scheme that transfers the model from a similar process is that similar industrial process data may be equally difficult to obtain in industrial applications.

As for the creation of a unified embedding space, the data-driven method was developed here as the efforts of artificial featuring engineering were expected to be kept minimum. The key sensor identification was based on the self-attention mechanism since it can be regarded as the basis for the decisions of the Transformer. The evaluation of the proposed model was kept the same as the publications working on the same datasets.

Objective 4: To reduce the computational consumption of the Transformer. As shown in Figure 1.1, The first step was to identify the major source of high computational load. Then, the calculation mechanism can be modified and adapted based on the characteristics of multiple sensors' data processing. Finally, a deep learning architecture can be designed based on the modified calculation mechanism, before evaluating and comparing its performance.

As this work can be an attempt to improve the Transformer architecture for sensor fusion used in previous research, the Hydraulic dataset was reused in this research. The Fast Fourier Transform (FFT) based autocorrelation calculation was employed here to improve the computational efficiency due to it can be efficient for identifying the temporal and spatial redundancy. The overall model architecture was kept the same as the original Transformer to keep comparability. Finally, the original Transformer and the proposed model were tested on the same GPU with full GPU capacity to evaluate their computational efficiencies.

1.7 Thesis Outline and Research Contributions

This thesis starts with investigating and verifying the potential of the Transformer architecture, a deep learning architecture based on the Attention Mechanism, on sensor anomaly detection. Then it focuses on adapting and improving this architecture to sensor fusion tasks in process monitoring, with respect to training data requirements, interpretability, and computational efficiency. The thesis is composed of 6 chapters. A brief description of these chapters and the corresponding contributions are given as follows:

Chapter 1: The current chapter first introduces the motivation for this research and then discussed the importance and challenges of sensor fusion technology in industrial process monitoring. Next, the research scope of this dissertation was given, followed by aims and objectives. Finally, the key contribution of this dissertation was summarised and the publications were listed.

Chapter 2: This chapter systematically reviewed the application of multi-sensing fusion algorithms in industrial process monitoring, including an overview of industrial process monitoring and sensor fusion technologies, as well as some commonly used conventional sensor fusion algorithms and artificial intelligence-based methods. The challenges of deep learning for the industry are also highlighted in this chapter.

Chapter 3: This chapter proposes a novel Dual-channel Attention-based Convolutional Neural Network for sensor anomaly detection in multivariate time series. The main contributions include:

a) A novel self-attention-based deep neural network block, Dual-channel Attention Mechanism (DAM), was proposed. This block can incorporate sensor-wise and time wise attention, making the extraction of spatiotemporal features integrated into the learning process, and eliminating the artificial signal processing stage for extracting the spatiotemporal features before designing anomaly detection algorithms.

b) This method achieved SOTA performance on a public dataset of the Connected and Automated Vehicles in a number of metrics, especially its sensitivity and performance on small anomaly detection.

Chapter 4 This chapter develops a novel Transformer-based deep transfer learning solution that generalises the feature representation from a data-rich modality to address the challenges in sensor fusion, thus benefiting from the learning ability of deep models and reducing the reliance on expensive industrial data collection. The main contributions include:

a) This methodology can establish a unified feature representation and association relationship for the multiple sensor data at significantly different sampling rates from different modalities.

b) The problem of poor interpretability when using Deep Learning in sensor fusion tasks is alleviated based on the Attention Mechanism. By visualising the attention weight matrix, the judgment basis of the model can be inspected, thus assisting in the identification of critical sensors.

Chapter 5 This chapter proposes a modified Transformer architecture, Sensorformer, for sensor fusion based on the Fast Fourier Transform and the self-attention mechanism to improve computational efficiency. The main contributions include:

a) This architecture can merge the correlated channels progressively throughout the training process. Instead of the raw data level, the merging of correlated data occurs at the feature learning level, providing a new approach to addressing sensor correlation in Deep Learning.

b) The proposed method can reduce memory usage to around one-fifth of its original size while maintaining similar inference accuracy. The inference time can also be 2 times faster. This allows the Transformer to be used for industrial multi-sensor data processing tasks in a more resource-efficient and faster manner.

Chapter 6: This chapter summarises all the chapters, including the methodologies and the corresponding results. Based on the research presents in this thesis, some future works worth investigating are presented subsequently.

1.8 Publications

- Zhang, Z., Farnsworth, M., Tiwari, D., Jewell, G. and Tiwari, A., 2022, September. Sensorformer: A Memory-efficient Transformer for Industrial Sensor Fusion. In 2022 27th International Conference on Automation and Computing (ICAC) (pp. 1-6). IEEE.
  DOI: 10.1109/ICAC55051.2022.9911157.

- Zhang, Z., Farnsworth, M., Song, B., Tiwari, D. and Tiwari, A., 2022. Deep Transfer Learning With Self-Attention for Industry Sensor Fusion Tasks. IEEE Sensors Journal, 22(15), pp.15235-15247.
  DOI: 10.1109/JSEN.2022.3186505

- Farnsworth, M., Tiwari, D., Zhang, Z., Jewell, G.W. and Tiwari, A., 2022. Augmented classification for electrical coil winding defects. The International Journal of Advanced Manufacturing Technology, 119(11), pp.6949-6965. DOI: 10.1007/s00170-022-08671-w

- Tiwari, D., Farnsworth, M., Zhang, Z., Jewell, G.W. and Tiwari, A., 2021. In-process monitoring in electrical machine manufacturing: a review of the state of the art and future directions. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 235(13), pp.2035-2051. DOI: 10.1177/09544054211016675
- Zhang, Z., Yao, Y., Hutabarat, W., Farnsworth, M., Tiwari, D. and Tiwari, A., Self-attention-based Sensor Anomaly Detection in Automated Vehicles. Submitted to IEEE Transactions on Intelligent Transportation Systems.

2 Literature Review

2.1 Overview of Industrial Process Monitoring

Industrial process monitoring leverages human labour, sensors, and various data acquisition techniques to gather comprehensive insights from industrial operations. This concept employs model-based, knowledge-based, and data-driven approaches to analyse the collected data, aiming to infer the working conditions and performance of the system. Within this framework, sensor anomaly detection plays a supporting role by identifying deviations in sensor data. Although not the primary focus, this capability is essential for ensuring the accuracy and reliability of the monitoring process. It aids in the timely identification of potential issues, contributing to the overall effectiveness of industrial process monitoring by allowing for prompt corrective actions when necessary.

2.1.1 Model-based Process Monitoring

In model-based process monitoring, the first step is acquiring in-depth knowledge of the target process, which enables researchers to build a mathematical model to describe the process behaviour based on its physical and mathematical characteristics. The overall workflow can be illustrated in Figure 2.1. Given a system input, the mathematical model will calculate the behaviour of the system, allowing for comparison with the real behaviour. The difference between real and calculated behaviour can be then identified by residual generation, and the system conditions can be finally determined by evaluating the generated residuals. Namely, when the normal condition of this system is being observed, the real behaviour should be the same as the calculated behaviour, making the residuals zero. A non-zero value in the residuals is often an indication of faults in the system. It is important to mention here that since noise and uncertainty are generally present in real systems, the thresholds of residual values are usually applied [24].

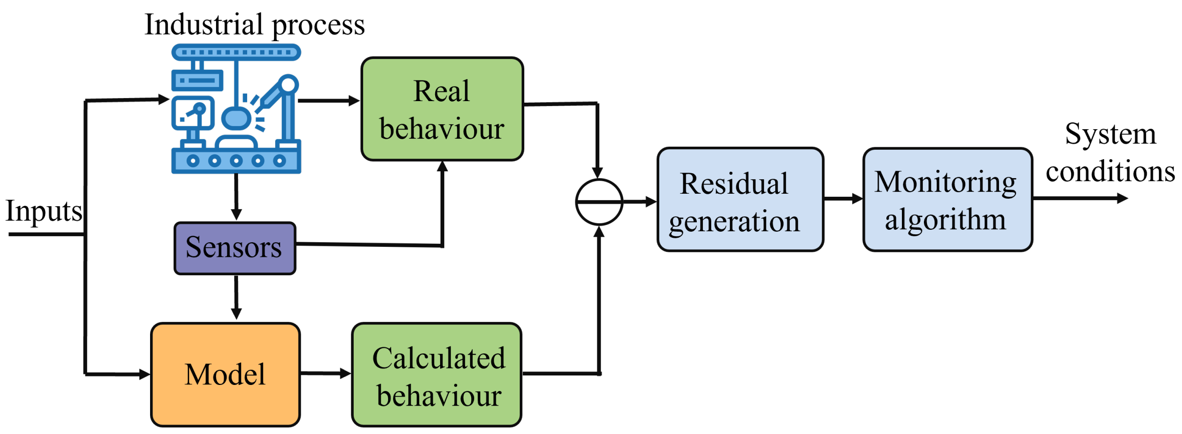


Figure 2.1.: Model-based Process monitoring

It is clear that interpretability is the strength of this approach since the physical processes of processes are mathematically modelled. Over the last few decades, there has been a large number of mathematical model-based process monitoring [25–27]. However, for many modern industrial processes, developing a model-based monitoring system can be challenging due to the complexity of the processes. Understanding sufficient physical details can be time-consuming and difficult, which limits the application of the model-based methods [28].

2.1.2 Knowledge-based Process Monitoring

When sufficient physical details of the process are difficult to understand due to the non-linear, dynamic or other complex factors, knowledge-based process monitoring could be a possible alternative [28]. The conceptual workflow of this method can be found in Figure 2.2. In this type of method, firstly, a dataset that contains both historical normal and faulty conditions need to be created. Then, the knowledge of this system can be extracted by evaluating the correlation between the features of process parameters or sensor data and system characteristics. Finally, the data from the real process will be analyzed based on the extracted knowledge to identify system conditions. It can be found that feature extraction can be critical for this type of method, as the representativeness of these features will directly affect the performance of the monitoring system. The commonly used features include statistical features such as mean, variation, Kurtosis, and Skewness [29], time-frequency domain features [30], and sometimes task-oriented features [31]. Expert systems [32], fuzzy logic [33], and Bayesian networks [34] are examples of commonly used knowledge-based model monitoring techniques [28].

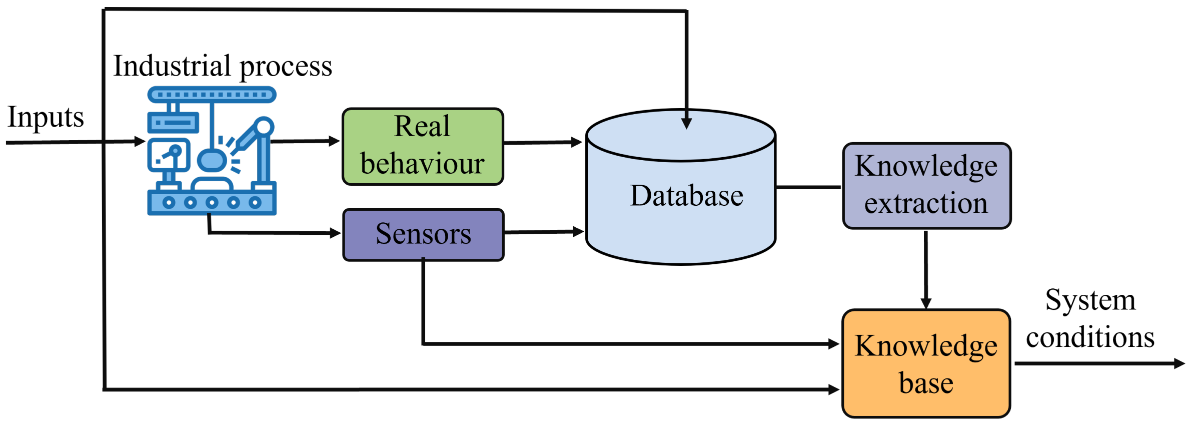


Figure 2.2.: Data-based Process monitoring

Compared with the model-based method, although the knowledge-based approach does not rely on the physical details of a system, it may not be an easy task for this approach to address a large and complex input space, such as the process monitoring system with a large number of sensors measuring different modalities with different sampling rates. This is because extracting knowledge from a high-dimension and complex space can be challenging.

2.1.3 Data-based Process Monitoring

Data-based, also called data-driven process monitoring is an approach that relies entirely on the historical data of a system [35], hence it offers an alternative when none of the methods mentioned earlier can handle the complexity of the monitoring task. As shown in Figure 2.3, the data-driven process monitoring can be regarded as an extension of the knowledge-based approach, with the difference that the knowledge extraction part is replaced by a data-driven model. The advantage of this method is that the mapping relationship between input (sensor data) and output (system conditions) space can be found directly with the help of the data-driven model, thus avoiding artificial knowledge extraction and rule building.

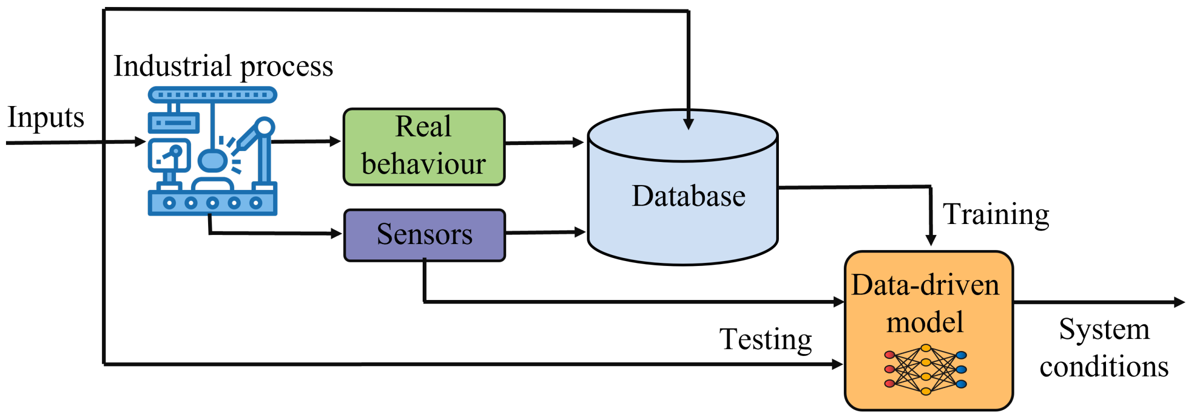


Figure 2.3.: Data-based Process monitoring

Multivariate Statistical Process Monitoring (MSPM) is a set of commonly used data-based methods and techniques for monitoring and controlling processes by collecting and analysing multiple process variables. These methods are particularly suited to production environments where process variables have complex interdependencies and interactions. The goal of MSPM is to ensure process stability and maximise product quality while reducing waste and improving efficiency. By monitoring and analysing multiple variables in the industrial processes, MSPM can identify production deviations, trends, and potential quality issues based on the statistical hehaviour of multivariate process parameters that are historically recorded [36].

The following are some examples of common MSPM methods:

- Principal Component Analysis (PCA): By identifying the dimensions that contain the largest variation (principal components) in the data, PCA reduces the dimensionality of the data while retaining most of the information. This helps to identify and monitor key variables in the production process [36].
- Partial Least Squares Regression (PLS): PLS takes into account the correlation between variables. It constructs a linear regression model to predict one or more response variables. This is applicable when there is a high degree of correlation between the process parameters [36].
- Statistical Process Control (SPC) charts: SPC charts are a tool for monitoring the stability of an industrial process. It can identify abnormal changes in the process by tracking the control limits of process variables or product quality parameter [37].
- Multivariate control charts: such as T2 control charts and Squared Prediction Error charts, specifically designed to monitor multiple variables of interest. This is a frequent approach in the early literature [38].

Compared to model-based or knowledge-based approaches, data-based approaches focus more on the use of real-world data to reveal the behaviour and performance of processes. It supports continuous improvement. This is because, by continuously analysing newly collected data, the data-based feedback loop ensures that the process monitoring model can be continuously updated and improved. However, the lack of interpretability is a significant drawback of this type of approach, as the characteristics and working behaviour of the process are modelled as a black box, resulting in the generalisation capabilities of this method being often questioned. In addition, the data-hungry nature is another disadvantage of this approach when applied to industrial scenarios, as the complexity of the model rises with the complexity of the process being monitored, and complex models tend to place higher demands on the amount of historical data.

2.1.4 Sensor Anomaly Detection

Anomaly detection has been extensively researched in the literature and commonly used methods include, but are not limited to, statistical methods, signal processing, time series analysis (TSA), and data-driven methods, especially deep learning. In a statistical approach, sensor data can be modelled by a statistical model and readings that deviate from the model prediction can be considered anomalous according to a pre-defined threshold [39]. The statistical models for this type of method can either be built from domain knowledge or statistically derived from the data collected, such as the Gaussian model [40], histogram-based model [41], Hypothesis testing [42], etc. Generally speaking, most of them can be interpretable and computationally efficient. However, this type of method requires that the sensor data can be characterised by a specific distribution, which may not be the case for many applications, especially for data of high dimensionality [43]. Signal processing-based anomaly detection, such as the Fourier transform and the wavelet transform is another commonly used method, which can be used for a variety of sensor data, such as acoustic, vision, physical parameters date etc. In [44], a wavelet-based method was proposed to detect anomalies in the presence of noise, and they achieved remarkable performance with respect to detection rate and computational efficiency. As signal processing methods attempt to form unique descriptions of different signals, they often have an advantage when dealing with unseen anomalies. However, similar to statistical methods, their performance may heavily depend on making assumptions (e.g. quasi-stationarity) to processes such as noise distribution, which may limit their applications [43]. In terms of time-series analysis-based anomaly detection, it can also be a prevailing Methodology [45], such as Kalman filtering [46], autoregressive moving average [47], symbolic TSA [48], etc. However, according to [43], this type of method may suffer from degrading performance when the anomaly causes dramatic changes to the original sensor readings.

Deep learning as a promising technology for sensor anomaly detection has received more attention recently, particularly in scenarios where there are large amounts of data with complex patterns and high dimensionality [49]. This is because deep learning methods typically can learn feature representations of the data that contain complex patterns and dependencies automatically. Compared with conventional methods, deep learning-based methods normally do not require any assumptions on the distribution of input data or noise, making it applicable in most areas with a certain amount of data. CNN can be one of the most commonly used methods and it has been shown to be effective in sensor anomaly detection [50]. It can be especially useful for handling multiple sensor inputs, as the multiple sensors data can be reorganised into one 2-d matrix or different channels, which does not increase computational load significantly as the other method when addressing multivariate inputs. For example, in [51], Chen et al. proposed a CNN-based anomaly detection algorithm for multiple sensor streams. However, CNN can be limited in its ability to model global features due to its reliance on a large number of local kernels to extract features, resulting in difficulty in capturing long-range dependencies, such as inter-sensor dependencies and time dependencies. The above shortcomings of CNNs can be well compensated by the LSTM, which models time dependence through a recurrent structure. For example in [52], LSTM and CNN were combined to detect sensor anomalies for Connected and Automated Vehicles (CAVs). They achieved a state-of-the-art performance on a public CAV dataset. However, since LSTM can be an RNN structure, despite the advantage over CNNs of introducing time-dependent modelling, the forgetting problem may be still non-negligible, resulting in small anomalies in the time series being difficult to detect.

2.2 Definition and Architectures of Sensor Fusion

Industry 4.0 or smart manufacturing introduced a clear trend of industrial technology development, which is powered by advanced communication technology and advanced data analytical methods. In such a scenario, the variety and amount of sensor data coming from the production process, products, and production machinery may grow exponentially [53]. This often includes state, process, vision, vibration pressure data, etc. Hence, there is a challenge on how to harness sensor data collected from different modalities to extract beneficial information, that can be used to improve the analysis performance such as useful remaining life (RUL) estimation [54], faults inspection, and diagnosis. Therefore, sensor fusion methods also referred to as data fusion, can play an important role in solving this challenge.

2.2.1 Definition of Sensor Fusion

Humans and other animals normally employ a variety of senses, including sight, touch, smell, hearing, and taste, to perceive the external environment in order to gain a thorough awareness, assisting them in responding to the environment. Sensor fusion is the engineering version of the same logic. This research domain focuses on the methods for combining and deriving the information from multiple sensors to form a unified representation and thus improve the quality and comprehensiveness of information compared with using a single sensor alone. The formal definition of sensor fusion was initially given by the Joint Directors of Laboratories (JDL): a multilevel, multifaceted process dealing with the automatic detection, association, correlation, estimation and combination of data from single and multiple sources to achieve refined position and identity estimates, and complete and timely assessments of situations and threats and their significance [55]. It can be found that the original concept of sensor fusion was specifically aimed at target tracking and state estimation [56], but gradually, as the benefits of multi-sensor fusion were widely witnessed, it had been generalised to many other applications, such as autonomous vehicles [57], human activity recognition [58], robotics [59], and industrial process monitoring [60] etc.

2.2.2 Architecture of Sensor Fusion

Sensor fusion is an application-level term that involves many different disciplines. Any attempt to obtain high-level information from multiple sensor information sources can be classified as a sensor fusion technique. It is therefore difficult to establish a unified sensor fusion classification method. In the literature, researchers usually classify sensor fusion architectures according to 3 different criteria:

- The relationship between the sensors being fused.
- The relationship between the inputs and outputs.
- The location where the fusion takes place.

Classification based on the relationship between sensors

These classification criteria were proposed by Durrant-Whyte [61]. As shown in Figure 2.4, based on the relationships of multiple sensors, sensor fusion can be divided into 3 different types:

- Competitive fusion: The competitive fusion configuration employs multiple sensors to measure the same modality independently to improve the fault tolerance and reliability as illustrated by sensors S_1 and S_2 in Figure 2.4. In some harsh working environments, sensors may experience failure, which can be unacceptable by some safety-critical systems, hence this kind of configuration is usually applied to such systems to monitor critical parameters [62], such as hydraulic systems in aircraft [63].

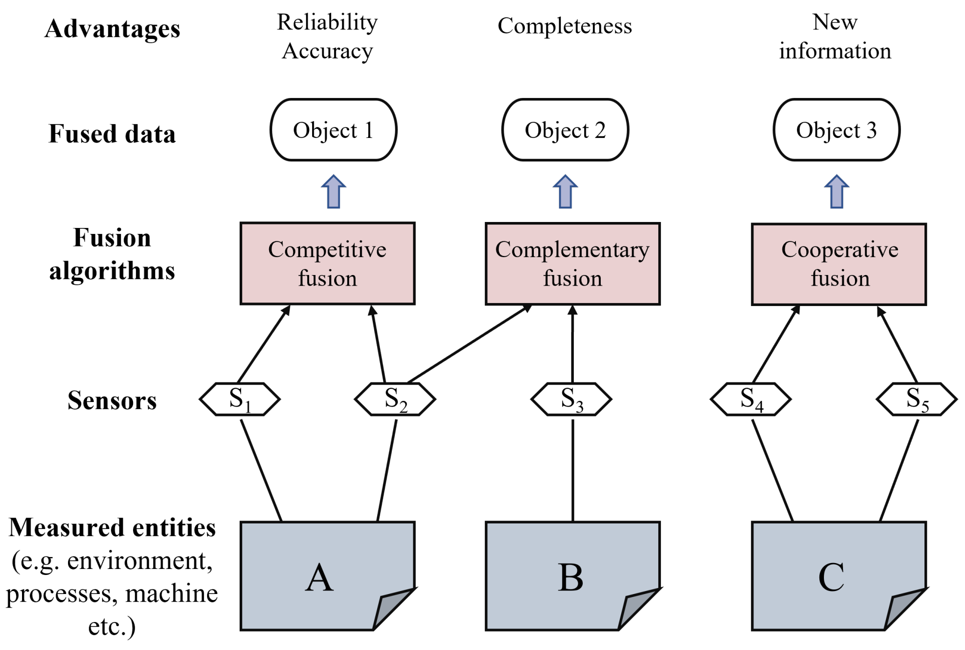


Figure 2.4.: Classification based on the relationship between sensors

- Complementary fusion: this configuration combines the sensors that measure the same entities on different aspects to provide a more comprehensive view of the phenomenon being observed, which can be illustrated by sensor S_2 and S_3 in Figure 2.4. For instance, in the field of autonomous driving, although the cameras can easily determine the identity of the surrounding objects, it can be challenging to precisely measure the distance between the car and the other objects. The situation is the opposite for light detection and ranging sensors (LIDAR). For LIDAR, while the distance information can be precise, the inference of identity information can be challenging due to the lack of visual features and low resolution [64]. Hence, the fusion of cameras and LIDAR can enhance the understanding of the surrounding environment significantly [65], and it can be classified as a complementary fusion configuration.

- Cooperative fusion. This type of sensor fusion can be expressed by sensor S_4 and S_5 in Figure 2.4. In this configuration, different sensors work cooperatively to infer further information that can not be captured by

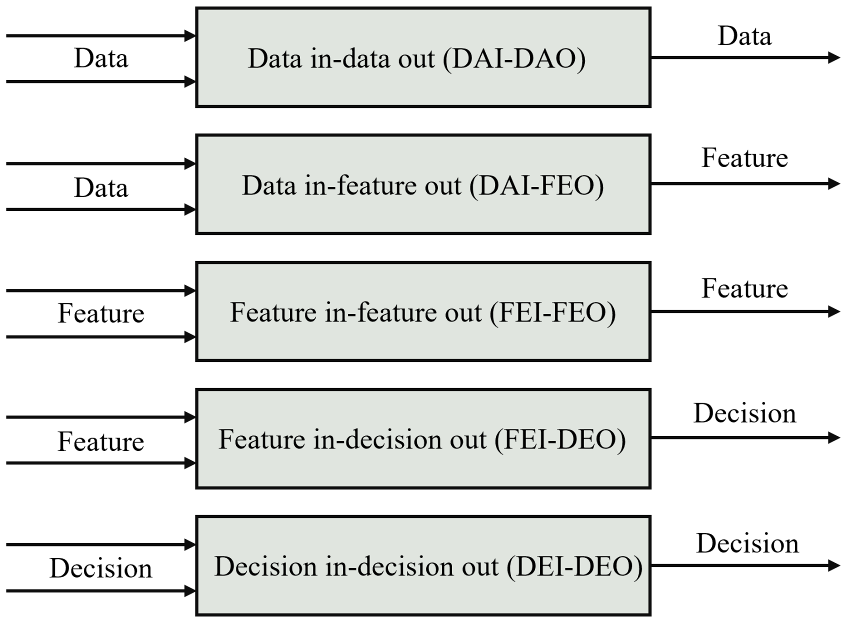

Data
Data in-data out (DAI-DAO)
Data
Data
Data in-feature out (DAI-FEO)
Feature
Feature
Feature in-feature out (FEI-FEO)
Feature
Feature
Feature in-decision out (FEI-DEO)
Decision
Decision
Decision in-decision out (DEI-DEO)
Decision

Figure 2.5.: Classification based on the relationship between the inputs and outputs

each of the single sensors. A typical example is binocular vision, which uses the triangular formed by two cameras with different views and the targets being observed to infer the depth of information.

Classification based on the relationship between the inputs and outputs

In addition to Durrant-Whyte's classification based on sensor interrelationships, Dasarathy offered an alternative classification method based on the relationship between the inputs and outputs of sensor fusion algorithms [66] as shown in Figure 2.5.

- Data In–Data Out (DAI-DAO) Fusion: This is the type of fusion where both the input and output are raw data and no feature extraction or pattern recognition will be performed, such as pixel matching in binocular vision and hyperspectral imaging techniques. Temporal and spatial data registration is critical to this type of fusion, and it has significant requirements for data compatibility in terms of format, sampling rate, and dimensionality.

- Data In–Feature Out (DAI-FEO) Fusion: The features of the object or environment being observed are derived from multiple sources. This type of fusion can be usually found in the domain of landscape monitoring [67], for example in [68], different features indicating the air quality of a city were generated from a variety of sources, such as sensors data from air quality monitor stations, meteorological data, taxi trajectories etc.

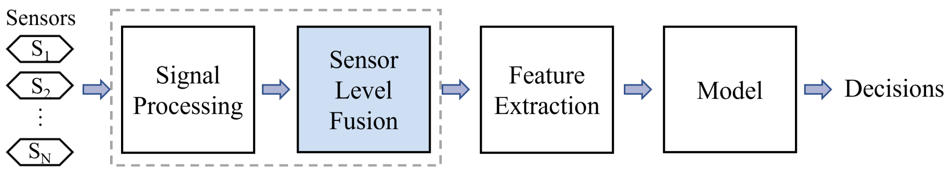
- Feature In–Feature Out (FEI-FEO) Fusion: A new feature space is generated from the features extracted from each of the single sensors. This type of fusion is mainly used for improving the representativity or reducing the complexity of the original feature space to lower the difficulty of extracting the high-level information from the features [69].

- Feature In–Decision Out (FEI-DEO) Fusion: In this type of fusion, feature extraction is usually performed first, and then a classifier or regressor is used to make the decision. It could be one of the most commonly used architectures of sensor fusion in process monitoring, and a large amount of literature can be found in this domain falling into this category [23] [70] [71].

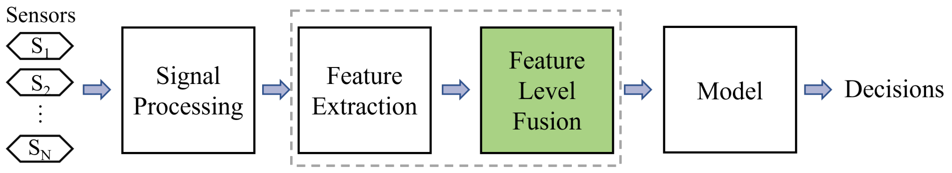
- Decision In–Decision Out (DEI-DEO) Fusion: This type of fusion focuses on how to combine the decisions from every single source to provide a final decision. When the above-mentioned architectures are not practical due to excessive differences in data characteristics, DEI-DEO fusion is at least feasible [72] [66].

Classification based on the location where the fusion takes place

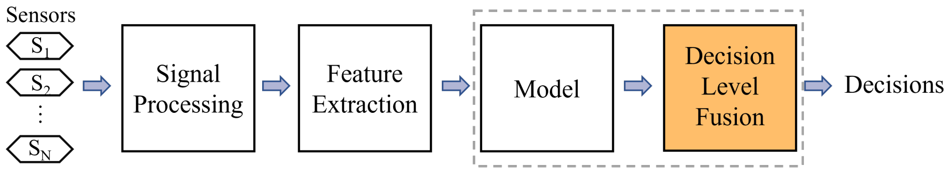
In the literature, Dasarathy's classification is sometimes simplified to sensor level fusion (DAI-DAO, DAI-FEO), feature level fusion (FEI-FEO, FEI-DEO), and decision level fusion (DEI-DEO) depending on where the fusion occurs as shown in Figure



(a) Sensor level fusion



(b) Feature level fusion



(c) Decision level fusion

Figure 2.6.: Different fusion levels

2.6 [73]. From sensor-level fusion to decision-level fusion, the level of abstraction increases with the level of fusion. It has to be mentioned that while lower levels of fusion have the richest details of information, they are usually corrupted by noise and large amounts of meaningless information. Higher levels of fusion after data processing and feature extraction may reduce the noise but may suffer from information loss [66]. Therefore, in practice, it is a trade-off that should be carefully considered when designing a sensor fusion architecture. In some studies, different levels of fusion sometimes co-exist to achieve better performance, such as in [74], which can be referred to as hybrid fusion. An example of this type of architecture can be found in Figure 2.7 where different levels of fusion collaborate with each other to make final decisions.

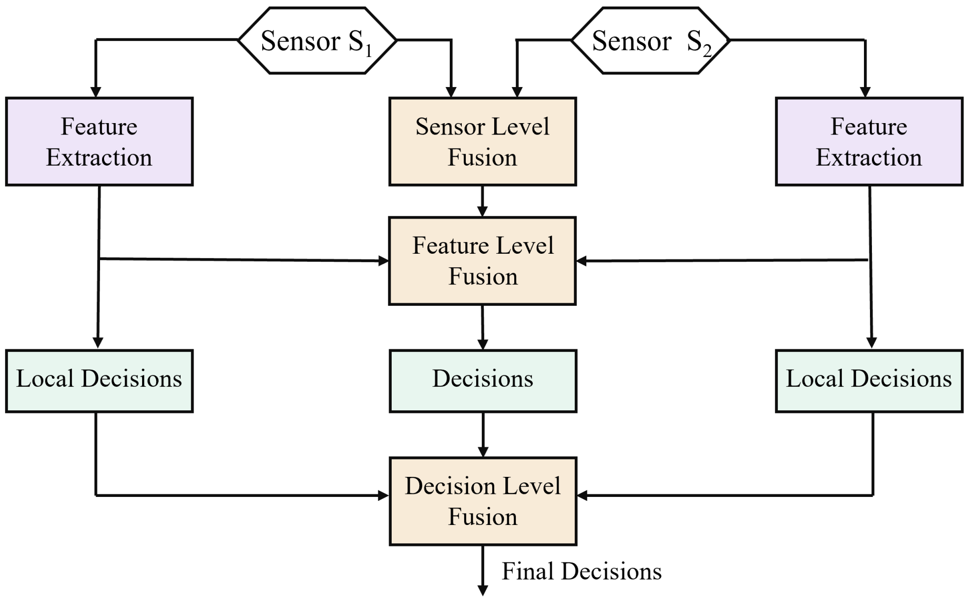


Figure 2.7.: Hybrid fusion

2.3 Conventional Multi-Sensor Fusion Technologies for Industrial Process Monitoring

2.3.1 Development of Sensor Technologies for Industrial Process Monitoring and Faults Inspection

Since the dawn of modern industry, engineers and researchers have been trying to monitor industrial processes to ensure the safety, stability and quality of industrial production. The rough timeline of the development of sensor technologies for process monitoring could be summarised in Figure 2.8. The very beginning may be traced back almost a century to the statistical control charts, and the most famous of these may be the Shewhart control chart proposed in the 1930s [75] as shown in Figure 2.9. This technique is used to monitor whether a process parameter is statistically controlled based on the assumption that the data are normally distributed. In this method, the mean of the monitored parameter is calculated as its centre line (CL), and then the upper control limit (UCL) and lower control limit (LCL) can be obtained based on the CL as shown in the fol-

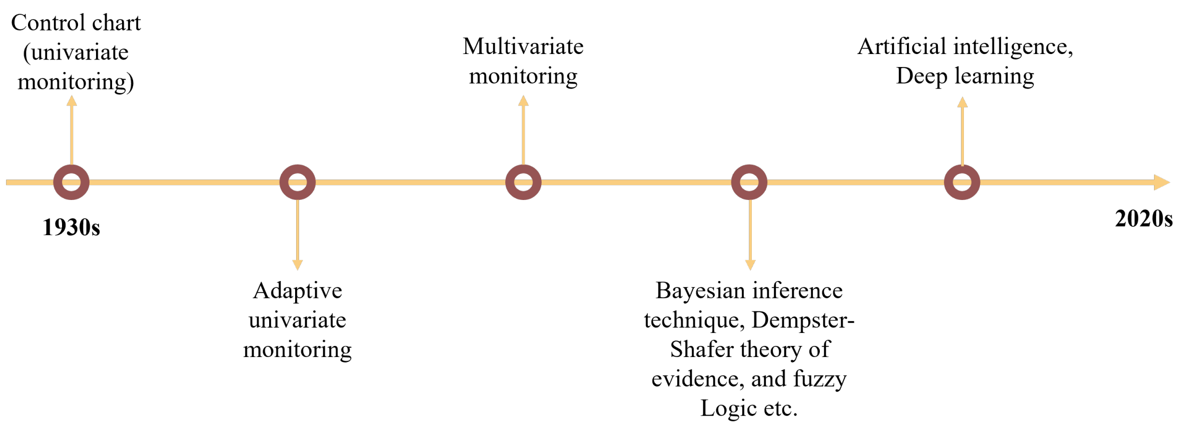

Control chart (univariate monitoring)
1930s
Adaptive univariate monitoring
Multivariate monitoring
Bayesian inference technique, Dempster-Shafer theory of evidence, and fuzzy Logic etc.
Artificial intelligence, Deep learning
2020s

Figure 2.8.: Hybrid fusion

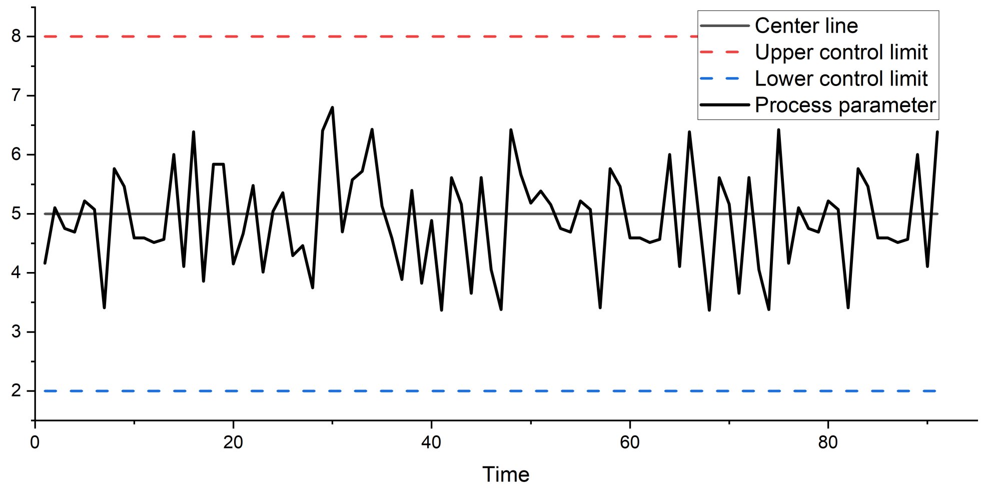


Figure 2.9.: Shewhart chart

lowing equations:

$$\begin{aligned} UCL_X &= \overline{X} + A_1\sigma_X \\ CL_X &= \overline{X} \\ LCL_X &= \overline{X} - A_2\sigma_X \end{aligned} \tag{2.1}$$

where σ_X is the standard deviation of the monitored parameter, A_1 and A_2 define the size of a fault-free working region. It can be found that the Shewhart chart provides a clear visual representation of whether a process parameter is abnormal or not, a small shift of mean cannot be effectively detected. Hence, the cumulative sum (CUSUM) [76] and exponentially weighted moving average (EWMA) [77] were proposed to allow the accumulation of deviations from the mean, making them more sensitive to this kind of small deviation. However, due to normal distributions and that the data are independent at each time steps are important assumptions of these methods, their applications may be restricted. Hence, some frequency domain monitoring schemes were proposed to avoid these assumptions, such as periodogram-based monitoring [78], discrete Fourier transform (DFT) based process parameter monitoring [79], and a variety of wavelet transform-based methods [80]. Although these methods were proposed decades ago, they are still useful tools nowadays, and a number of research can be found in the literature focusing on improving their adaptability [81] [82].

It can be found that the above-mentioned process monitoring methods focus on univariate monitoring, and they simply ignore the cross-correlation among process parameters which can contain critical information. Hence, multivariate monitoring methods were then proposed to solve this problem, such as multivariate EWMA [83], and multivariate CUSUM [84].

However, in modern industrial processes, the underlying non-linear, dynamic, multi-modal, space-time complex and high-dimension natures are usually observed [5], making the methods that monitor whether a range of process parameters are in the normal range far less sufficient to handle their complexity. Therefore, how to abstract high-level information, such as system operating status, fault status, and/or product quality, directly from a large number of process parameters is receiving increasing attention. As a result, sensor fusion-based technologies, which can model the dependency among the process parameters and infer high-level information start to play an important role. Some examples of commonly used algorithms are the Bayesian inference technique, the Dempster-Shafer theory of evidence, fuzzy Logic etc.

Moreover, the recent rapid developments in artificial intelligence (AI), especially deep learning, provide a powerful alternative to the sensor fusion domain for process monitoring applications and quickly become a trending research area [85] [86] [87]. The main reason behind this is that deep learning algorithms have shown a remarkable capability for feature learning from a complex input space in many domains, such as natural language processing and computer vision. Given the historical process data, they are able to find a mapping function from the input space to the output space automatically, even though this function may be highly non-linear. In addition, unlike most of the conventional methods, restricting assumption on input data is usually not required by deep learning algorithms, increasing their flexibility significantly [28]. These features are favoured when dealing with the complex parameters of modern industrial processes, making them a hot topic in recent years.

2.3.2 Commonly Used Sensor Fusion Algorithms for Industrial Process Monitoring

Principal Component Analysis (PCA)

An industrial process monitoring system that employs a high-dimensional input to characterise the properties of processes inevitably holds redundancy and correlated information, which may result in low monitoring performances and overfitting of the sensor fusion algorithm [88]. Hence, PCA is used intensively as it can be a very effective algorithm for data dimensionality reduction. The main idea of PCA is to project n-dimension features to k dimensions, and the projected features are known as k principal components. The k dimensions are mutually orthogonal and they can be calculated from the original space, ordered by the projected variance of the original data on the new axes. The workflow of PCA can be summarised as follows:

1) Given a n-dimension data, $X = \{x_1, x_2, x_3, \ldots, x_n\}$, to be reduced to k dimensions.

2) Data decentralisation, i.e. the features of each dimension are subtracted from their respective mean values.

3) Calculate its covariance matrix $\frac{1}{n}X^{r}X^{T}$.

4) Calculate eigenvalues and eigenvectors of covariance matrices by using Eigendecomposition or singular value decomposition (SVD)

5) Select the largest k eigenvalues, and the corresponding k eigenvectors are then formed into an eigenvector matrix P.

6) Calculate the newly constructed k-dimension data Y by $Y = PX$.

It can be found that, due to the fact that the calculation of the covariance matrix is necessary, the computational load can be increased hugely with the increase of features number, and the outliers inside of features may influence a lot.

Over the last few decades, dimensionality reduction has become an almost universally adopted technology in the field of industrial process monitoring before building multi-sensor fusion models. Of these, PCA is probably the most common one [89]. In [90], Heng et al. proposed a bearing monitoring model that can predict its Remaining Useful Life (RUL) based on PCA. The multi-sensor data were processed by PCA to reduce the complexity of their monitoring model. In [91], the PCA-processed data were the input of a valve monitoring system designed for reciprocating compressors. Similarly, PCA can also be widely found for electric motor monitoring [92], machining process monitoring [93], and the monitoring tasks in the nuclear power industry [94] etc.

In addition, many variants of PCA can also be found in industrial process monitoring, such as robust PCA [95], kernel PCA [96], probabilistic PCA [97], and recursive PCA [98] etc. It should be mentioned here that PCA is only one of many ways to reduce the dimensionality of data, and there are many other excellent dimensionality reduction algorithms in the literature, such as Linear Discriminant Analysis (LDA) [88], Neighborhood Component Analysis (NCA) [99], Partial Least Squares (PLS) [100], Independent Component Analysis (ICA) [101], etc.

Bayesian Inference

Baye's theorem provides a method to determine the current state of the object being observed by combining prior knowledge and current observations mathematically, which can be described by the following equation:

$$P(y \mid x) = \frac{P(x \mid y)P(y)}{P(x)} \tag{2.2}$$

where x can be the sensor measurements, and y can be the object's states under observation, such as faults or process conditions. It can be extended to multiple sensors by assuming the observation of each sensor is conditionally independent:

$$P\left(x_1, \ldots, x_i \mid y\right) = P\left(x_1 \mid y\right) ..P\left(x_i \mid y\right) = \prod_{i=1}^{i} P\left(x_i \mid y\right) \tag{2.3}$$

where x_i denotes the individual sensor measurement. Hence

$$P\left(y_j \mid x_i\right) \propto P\left(y_j\right) \cdot \prod_{i=1}^{i} P\left(x_i \mid y_j\right) \tag{2.4}$$

where y_j denotes the state of the target object. By selecting the maximum value of $P\left(y_j \mid x_i\right)$, the states can be determined. Bayesian inference is the recursive version of Baye's rule that takes time dependency into account by updating the prior knowledge based on the previous posterior as shown in the following equation:

$$P\left(y \mid X^k\right) = \frac{P\left(x_k \mid y\right) P\left(y \mid X^{k-1}\right)}{P\left(x_k \mid X^{k-1}\right)} \tag{2.5}$$

where X^{k-1} and X^k denote the previous and the current observations respectively. Some researchers believe that Bayesian inference can be analogised as a mathematical representation of the human reasoning process, due to the fact that humans tend to make judgements about a situation based on their past experience and update the judgements based on current observations [102].

As a popular sensor fusion method based on probability theory, Baye's theorem-based methods have been widely used for process monitoring. In [103], Anne et al. proposed a Bayesian method to monitor the working conditions of induction motors based on the fusion of vibration, current and acoustic emission sensors. Gaussian Naive Bayes (GNB) classifiers were used locally to infer the system conditions based on each of the individual sensors, and then, the decisions from these local classifiers were fused globally to generate the final decisions by Bayesian inference. This two-stage design made the final decision can be easily traced back to each of the sensors, as the decisions from each of the sensors were the inputs of the global Bayesian inference, making the further evaluation of a specific system fault possible. A similar two-stage design can also be found in [102], the authors applied local classifiers to the different components of a motor transmission system rather than the individual sensors, before applying the global sensor fusion. The other examples can be found in [104] and [92]. Generally, Bayesian inference models causality through probability, which allows the probability of system conditions to be evaluated, making it an important method in industrial process monitoring [102]. However, the Bayesian methods have some weaknesses that limit their applications. Firstly, the prior probabilities of the target events which can be essential for Bayesian methods may be difficult to be determined [16]. Secondly, no knowledge of an event is not allowed by Bayesian methods. For example, in the statement of whether or not it will rain, if the probability of rain is 0.7, then there must be a probability of 0.3 that it will not rain, and a state of uncertainty about whether or not it will rain is not allowed to exist here. Hence, the difference between ignorance (uncertainty) and randomness can not be discriminated against, making the modelling of no knowledge very difficult [105].

Dempster-Shafer (D-S) Evidence Theory

To address the challenges faced by Bayesian methods mentioned in the previous section, the Dempster-Shafer (D-S) evidence theory was proposed by Dempster in [106] as a generalised version of the Bayesian method and further developed by Shafer in [107]. On the one hand, D-S theory models the degree of certainty that an event will occur through an interval of certainty, enabling the modelling of ignorance or uncertainty. On the other hand, prior information on the frequency of an event is not required, reducing the difficulty of using sensor fusion algorithms [108].

In D-S theory, a finite set called frame of discernment, Θ, is firstly defined to represent all the possibilities of states about a given system being considered as follows:

$$\Theta = \{X_1, X_2 \ldots X_N\} \tag{2.6}$$

hence the set of all the subsets of Θ, i.e. the power set, can be expressed by:

$$2^\Theta = \left\{ \begin{array}{c} \varnothing, \{X_1\}, \{X_2\} \ldots \{X_N\}, \{X_1, X_2\}, \\ \ldots, \{X_1, X_2, \ldots, X_i\}, \ldots, \Theta \end{array} \right\} \tag{2.7}$$

which represents all the possible propositions of the real states of the system, including the empty set and its universe. Then, each element in this power set will be mapped to between 0 and 1, which can be denoted by:

$$m : 2^\Theta \to [0, 1] \tag{2.8}$$

This process is called the basic probability assignment (BPA) and the mapping rule is called mass function, requiring the following conditions:

$$m(\varnothing) = 0 \tag{2.9}$$

$$\sum_{A \subseteq \Theta} m(A) = 1 \tag{2.10}$$

Based on the value of $m(A)$, the belief $\mathrm{bel}(A)$ which can be regarded as the lower belief limit, and the plausibility $\mathrm{pl}(A)$ which can be regarded as the upper limit of belief can be derived from the following equations:

$$\mathrm{Bel}(A) = \sum_{B \subseteq A} m(B) \tag{2.11}$$

$$\mathrm{Pl}(A) = \sum_{B \cap A \neq \varnothing} m(B) \tag{2.12}$$

As a sensor fusion algorithm, D-S theory provides a combination rule that can fuse the belief from multiple information sources (sensors), and each of the sources

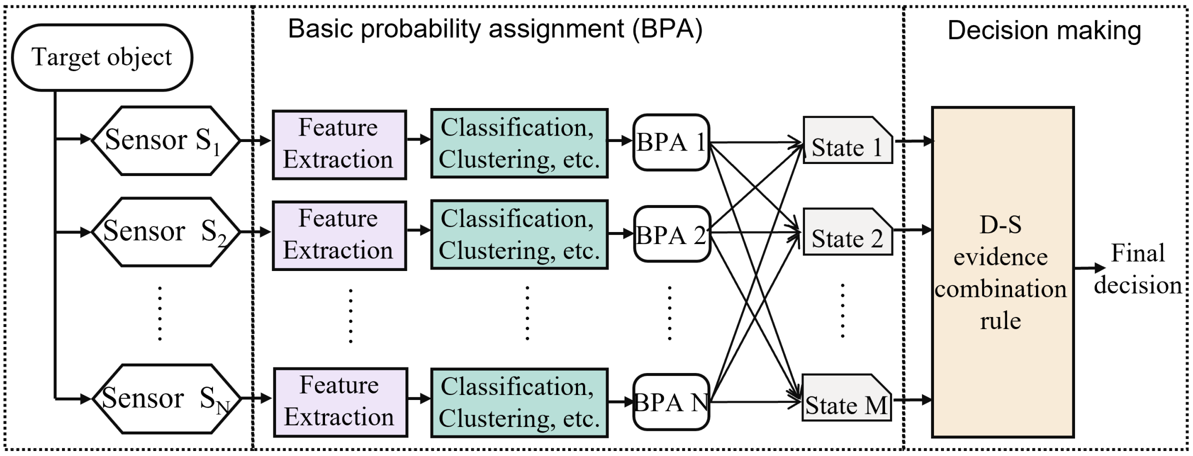


Figure 2.10.: General sensor fusion architecture based on D-S theory

is allowed to contribute belief to the joint belief based on its own degree of belief. Using two different information sources with mass $m1$ and $m2$ as an example, the combination process can be expressed by:

$$m(C) = (m_1 \oplus m_2)(C) = \frac{1}{1-k} \sum_{A \cap B = C} m_1(A) m_2(B) \tag{2.13}$$

where k is the coefficient measuring the conflict level between different information sources, which can be derived by:

$$k = \sum_{A \cap B = \varnothing} m_1(A) m_2(B) \tag{2.14}$$

The results from the above-mentioned combination are usually used as the basis for final decisions [109].

In terms of sensor fusion for industrial process monitoring, D-S theory can be an important tool. The general process of applying D-S theory can be illustrated by Figure 2.10. Firstly, BPA will be assigned to all the possible states of the system being monitored based on the evidence provided by each information source, followed by the joint belief calculation based on the combination rule. It can be found that the representativeness and reasonableness of the BPA process are the core of this approach. This is because evidence theory only provides a reasonable method for combining evidence from different sources, and the quality of the evidence provided by different sensors is the key factor that directly affects the performance of this method. Hence, there is much research focused on the BPA process to develop a D-S theory-based process monitoring system. In [110], Liu et al. used K-Nearest Neighbor (KNN) to assign the BPA to different working conditions to monitor a motor transmission system. In [111], the authors proposed a sensor fusion system for monitoring an Additive Manufacturing (AM) process. A Convolutional Neural Network (CNN) was used to assign BPA based on vision data, and then, it was fused with the BPA that came from the other process parameters, achieving the fusion between vision and numerical process parameters. Moreover, in [112], to improve the BPA process, the authors introduced the assessment of information quality based on Shannon entropy to the calculation of BPA, hence optimising the BPA. They successfully improved the monitoring system of an engine by this method. Nevertheless, in addition to the research on developing efficient BPA processes, the literature on optimising the D-S evidence combination rule for industrial process monitoring can also be found, for example, in [110] [113].

Although D-S theory has been proven efficient for sensor fusion in process monitoring scenarios, it has some drawbacks. It is generally incapable of fusing the evidence provided by highly conflicting data sources [114], and the evidence provided by every single source is also assumed to be independent, which may be very difficult to be ensured in real scenarios.

Fuzzy Logic (FL)

FL was first introduced by Lotfi Zadeh in [115], and it had gradually become a popular high-level sensor fusion method due to its ability to model fuzzy information. In this approach, the specific values of the sensors are fuzzified into an interval from 0 to 1 by a function called the membership function, and then the fuzzy output set can be generated by fusion rules defined in advance. The predefined rules are usually in an IF-THEN format. For example, IF sensor 1 is large and sensor 2 is small, THEN the output is small. For creating the rules, generally, there are two options: (1) expert knowledge can be employed, (2) the rules can

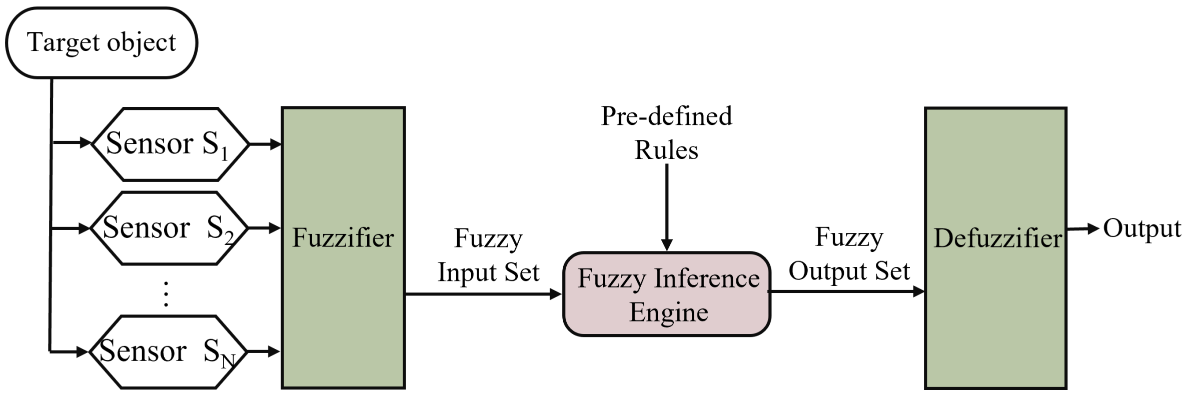


Figure 2.11.: General Fuzzy Logic working flow

be calculated by historical data. Finally, the fuzzy output set can be defuzzified to obtain the final output. The whole workflow can be illustrated in Figure 2.11. In industrial sensor fusion scenarios, sensor uncertainty is widespread and the influence of process parameters on the system can be sometimes ambiguous, making it difficult to determine a one-to-one mapping from sensor data to performance parameters. The FL can be a useful tool in such cases, this is because it avoids the need to establish one-to-one mapping relationships by abstracting sensor data into fuzzy descriptions such as 'small, medium or large' and then modelling the system behaviour by expert-specified rules.

Speaking of multi-sensor fusion for process monitoring, literature shows that the FL could be a hot topic in complex non-linear process monitoring tasks. In [116], Ge and Liu used FL on the decision level to monitor a chemical production process based on the data from 41 sensors. Instead of using raw data, the data were processed by a variety of technologies, such as PCA, PLS, and ICA etc., and then FL has used to fusion the results from them to make the final decision. In [117], Ammiche proposed an FL-based monitoring system for monitoring the cement production process to address its non-linearity and time dependency. In addition, machining-related process monitoring tasks, such as surface roughness monitoring and tool wear monitoring can be another domain that intensively employs FL [118]. In [119], Kuntoglu and Saglam fused the feed rate, cutting speed, depth of cut, acoustic emission, and tool tip temperature by FL to predict machining tool wear. Similarly, Gajate et al. used FL to model the signal from multiple sensors for tool wear monitoring. Compared with the method in [119], the difference was that a neural network was combined with FL, and the output of FL was used as the input of a neural network to make the final decision.

Compared to probabilistic methods, FL has advantages in dealing with fuzzy information and complex systems and is generally more suitable for high-level applications such as decision-level fusion. However, due to the subjective description of information, such as expressions like 'high, medium, low', and the choice of membership functions generally highly relies on the experience of engineers or researchers, its objectivity may be questioned [118]. In addition, while the number of sensors to be fused, the computational complexity of FL can be increased dramatically, making it challenging to use FL alone to monitor large multi-sensor systems [120].

2.4 Deep Learning Based Multi-Sensor Fusion Technologies

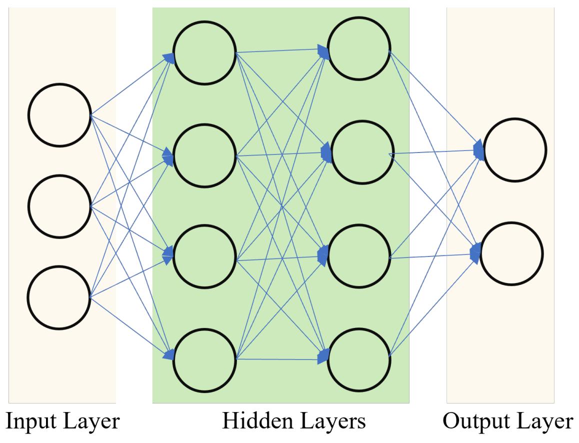
Deep learning is a data-driven approach. As many approaches to multi-sensor fusion that can be used to monitor industrial processes have been proven effective in theory and in practice over a long time, deep learning may not be necessarily essential. However, deep learning has several attractive advantages that make it a very important role of this domain. Firstly, deep learning models have the ability to automatically extract useful features and information from large amounts of data. This can significantly reduce or even eliminate the reliance on artificial feature engineering compared to traditional methods, which may take a large amount of time and effort. Secondly, deep learning has a strong potential to recognise sophisticated patterns and features out of data, which can be difficult or even impossible for many traditional methods. This advantage is becoming increasingly important when dealing with the increasing complexity of industrial process data. In other words, these two advantages mean that while most traditional methods are only effective at specific levels of fusion, e.g. D-S evidence theory is more suitable for decision-level fusion and PCA is more suitable for the fusion at the raw data level, deep learning-based approaches have no significant limitations in terms of fusion levels, giving it the potential to be an end-to-end solution.

In addition, the availability of industrial sensor data has increased significantly in recent years, which has built a solid basis for the use of deep learning. This is because, firstly, modern industrial sensors are significantly more affordable and reliable than those of the past, and they can be easily integrated into a variety of machines of processes. Then, the recent rapid development of advanced information and communication technologies such as Cyber-Physical Systems (CPS), IIoT, big data etc. enables a large amount of sensor data to be collected, stored and managed efficiently. Moreover, in addition to data availability, the performance of computing technology such as cloud/edge computing and hardware such as GPU/TPU is now much improved compared to the past, making it possible for deep learning, which requires large computing capacity, to be performed effectively and timely. The facts mentioned above are the reasons why deep learning is receiving more and more attention in the industry. This section will introduce the concepts of different deep learning algorithms and summarise the current status and challenges of deep learning applications in industrial process monitoring from the perspective of different deep learning algorithms. The complete working flow of deep learning will be described in section 2.4.1 as artificial neural network (ANN) is the most fundamental building element of deep learning, and the other algorithms can be regarded as the variants or different topologies of ANN. The working flow of the other algorithms will not be repeated in the following sections, as they can be similar.

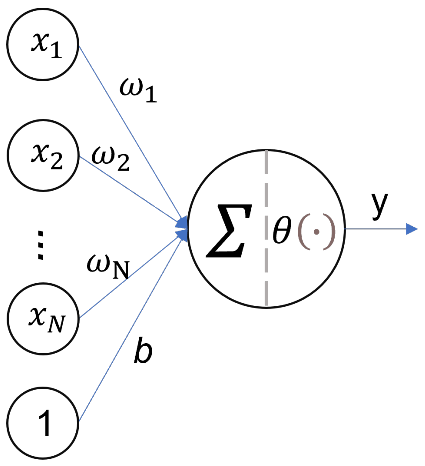
2.4.1 Artificial Neural Network (ANN)

The basic idea behind ANN was to replicate how biological neural network process information, where each neuron combines the signals it received from the neurons to which it is linked before passing them on to the next neuron in accordance with a simple activation rule, and such a network of interconnected neurons is a neural network. The mathematical version of the above-mentioned process is the ANN algorithm, and it was first proposed by McCulloch and Pitts in the 1940s in [121], and this can be regarded as the very beginning of ANN.

The general architecture of ANN is composed of an input layer, several hidden layers, and an output layer, which can be illustrated in Figure 2.12a. Each



(a) Basic Neural Network Architecture



(b) One Single Neuron

Figure 2.12.: Neural Network

neuron is linked to all the neurons in the previous layer except for the input layer, and there is no direct connection among the neurons within each layer. The computation process of each of the neurons in ANN is shown in Figure 2.12b, which can be described by the following equation:

$$y = \theta\left(\sum_{i=1}^{N} \omega_i x_i + b\right) \tag{2.15}$$

where ω_i are the weights of different inputs x_i, b is the bias, $\theta(\cdot)$ is the activation function which maps this summed inputs of a single neuron to its output, and in turn, the output is passed as the input of the next neuron. The activation function is a very important element. This is because, based on equation 2.15, an ANN without activation function is a linear regression model, which means its modelling capacity can be limited. By using a non-linear activation function, the non-linearity can be introduced to ANN, hence its scope of application can be significantly expanded. The commonly used functions are Sigmoid, Tanh, ReLU and its variants, Swish, and SoftMax, and a detailed description of these functions can be found in [122].

Optimising the parameters of ANN, i.e. the ω_i and b for all the layers is the key for using ANN, which is also called the training process. This can be achieved by minimising the distance between the outputs of ANN and the desired target values $\mathbf{y}_i$, i.e. the loss of ANN. Given a set of training data, $(\mathbf{x}_i, \mathbf{y}_i, i = 1, \ldots, B)$, where B is the batch size of the training samples. The loss of ANN can be evaluated by the following equation:

$$\mathcal{J}(\varphi) = \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}\left(\mathbf{y}_i, \mathbf{y}_i^*\right) \tag{2.16}$$

where $\mathcal{J}(\varphi)$ denotes the mean loss of a batch of training data given the parameters of ANN φ, $\mathcal{L}(\cdot)$ denotes the loss function which is used to evaluate the loss between the output of ANN $\mathbf{y}_i^*$ and its real values $\mathbf{y}_i$. Then the parameters of ANN can be updated recurrently based on the loss obtained from equation 2.16 by the following equation, which is the backpropagation procedure [123]:

$$\varphi_j = \varphi_{j-1} - \eta \nabla_{\varphi} \mathcal{J}\left(\varphi_{j-1}\right) \tag{2.17}$$

where η is the learning rate that controls the step size of each parameter update iteration. The above process can be repeated until the loss of the model converges and a complete training process is completed.

The nature of ANN can be a non-linear mapping function. ANN has gradually become the focus of research because, in 1989, Hornik et al. proposed the Universal Approximation Theorem which proved that there is no theoretical upper limit to the complexity of the functions that can be fitted by an ANN [124]. This theorem stated that if an ANN has at least one hidden layer and its activation function Ψ is non-decreasing and satisfies $\Psi : R \longrightarrow [0,1]$, $\lim_{\lambda\to\infty}\Psi(\lambda) = 1$, $\lim_{\lambda\to-\infty}\Psi(\lambda) = 0$ (e.g. sigmoid activation function), it can approximate any functions from one finite space to another with arbitrary accuracy, by giving a sufficient number of hidden neurons. This is the theoretical assurance that ANN and deep learning methods have significant promise for a variety of applications.

Speaking to industrial process monitoring applications, the data collected by the different sensors are often used as inputs to the ANN and the outputs can be high-level information such as the working conditions of the system, thus achieving the fusion of data from multiple sensors. In the literature, process monitoring using ANNs alone could be conservative and shallow networks are usually used. Therefore the input data of ANN are often in the form of single value sensor data such as temperature, speed, etc., rather than high-frequency complex signals such as vibrations to avoid excessive difficulties for shallow ANN. For those using high-frequency complex signals, manual feature extraction is often performed to represent the original signal with several single values before using an ANN. This is due to the fact that although there is no upper limit to the complexity of the functions that can be fitted by an ANN, the challenge is whether the optimal parameters can be obtained by optimisation [123]. In addition, the fully connected structure will also introduce a large computational load that increases exponentially with the number of input features. For example, given a hidden layer with 1,000 neurons, if the input of an ANN has 1,000 features, there will be 1,000 x 1,000 = 1,000,000 parameters to be optimised.

In [125], Rafiee et al. developed an ANN-based gearbox monitoring system. The vibration signal was firstly decomposed into 16 different wavelet packets by wavelet transform, and then the standard deviations of these packets were used as the input of a 3-layer ANN to classify whether the gearbox was in a faulty condition. In [126], Aydin et al. used ANN to monitor the metal-cutting process. The cutting speed, depth, force, and feed rate were employed as ANN's input to predict the wearing conditions of the tool. The above-mentioned methods can be the typical methodology of using ANN for process monitoring, similar research can also be found in [127] [128]. However, when solving multiple sensor problems, the space of the extracted features can be huge, which can lead to the failure of ANN. Hence, feature selection or dimension reduction technologies such as PCA are usually performed to reduce the input space. For example, in [129], the genetic algorithm was used to select a small subset of features before using ANN. Similarly in [130], ReliefF, an algorithm that can search dependencies among features, was used.

2.4.2 Convolutional Neural Network (CNN)

As mentioned in the previous section, ANNs are normally not ideal for handling the high sampling rates data directly which results in large input space. Hence, CNN was proposed to mitigate this problem [131]. Using a 2-dimension input as an example, the general structure of CNN can be illustrated in Figure 2.13. Firstly, a number of kernels of size $m \times n$ can be randomly initialised to perform

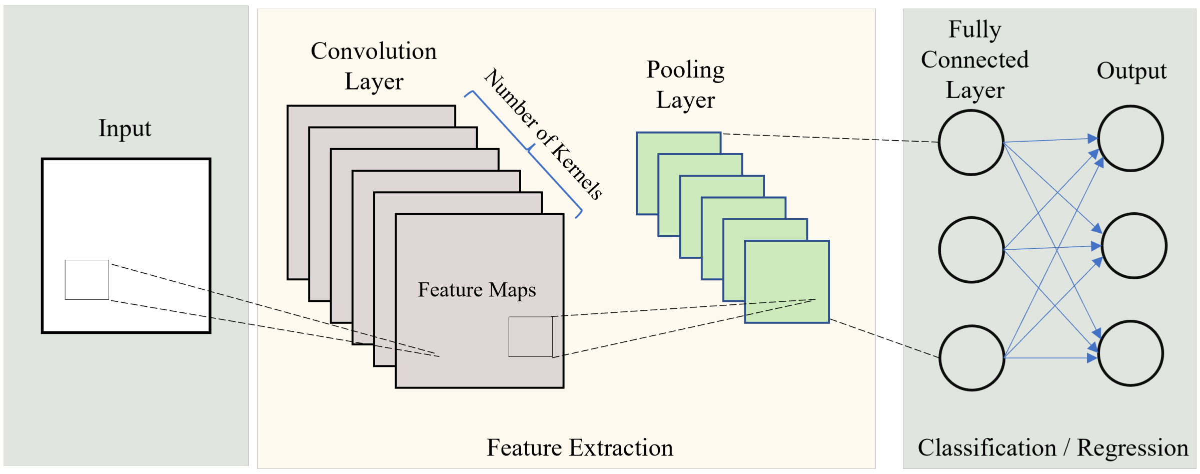


Figure 2.13.: Diagram of a single layer CNN

convolution calculation with the input to generate feature maps based on the following equation (the calculation process for one kernel):

$$G(i,j) = (X * K)(i,j) = \sum_m \sum_n X(i+m, j+n)K(m,n) \tag{2.18}$$

where $G(i,j)$ denotes the value in the feature map at i-th row and j-th column, $K(m,n)$ denotes the value in the kernel at m-th row and n-th column, and X is the input. Then, a pooling layer is usually applied subsequently to downsample the feature maps. This layer replaces the values in a small rectangular neighbourhood of feature maps with their statistical features. For example, the max pooling can be expressed by the following equation:

$$y = \max_{(p,q)\in\mathcal{R}} x_{pq} \tag{2.19}$$

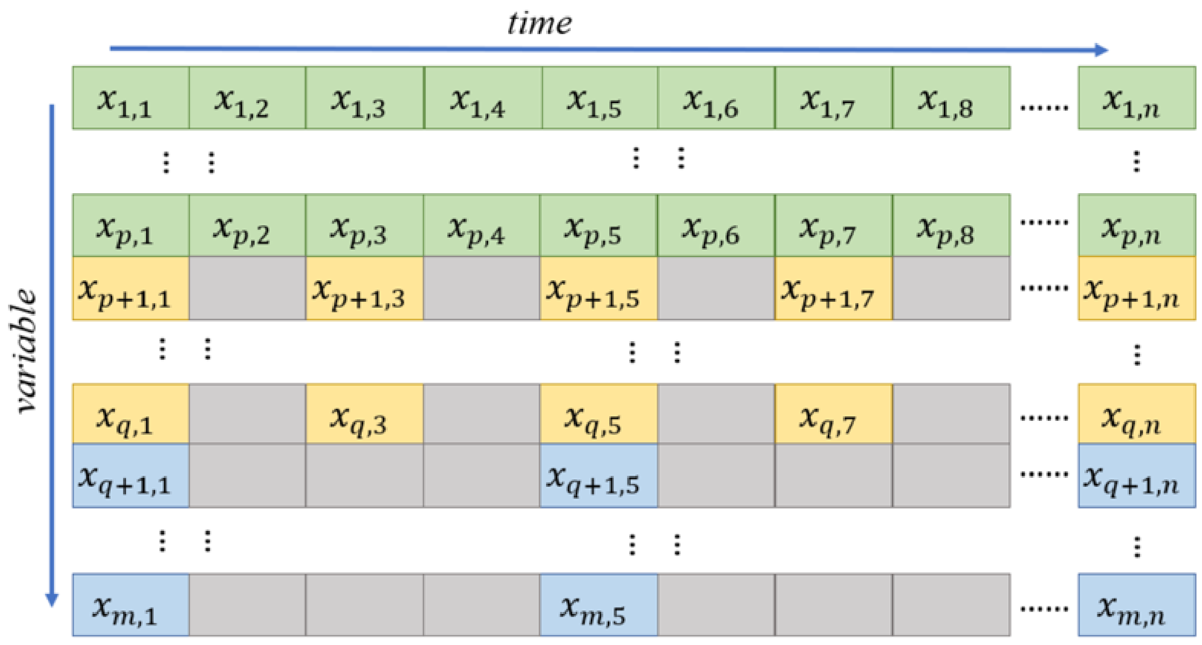
where y denotes the output value of the pooling operation at a rectangular area $\mathcal{R}$, x_{pq} denotes the element in $\mathcal{R}$ at (p,q). The other commonly used statistical features include average, weighted average, L^2 norm, etc. There has to be mentioned that similar to ANN, nonlinear activation is also necessary for CNN, and it can either be utilised before or after pooling. The above-mentioned convolution layer and pooling operation can be repeated several times to extract features, and finally, all the feature maps from the last layer can be flattened and sent to an ANN which is used for classification or regression.

Compared with ANN's fully connected architecture, CNNs have the following 3 characteristics [123]:

- Sparse connectivity. Instead of a full connection, CNNs use smaller kernels to scan the entire input in steps of a certain size, which effectively reduces the number of network parameters and hence the computational load and training difficulty.
- Parameter sharing. This means each kernel shares parameters when scanning all locations of the input, thus reducing the storage requirement of model parameters.
- Equivariant representations. The output of the convolution operation will be shifted in accordance with the input data shift, so the learned representations remain the same.

Therefore, CNNs are normally more appropriate for processing high-frequency data, especially signals where local features are critical. A good example of this is the major success of CNNs in image processing [132]. In terms of industrial sensor data, high sampling rate data can be very important and intensively used. For example, the high-frequency current signature can be an excellent representation of motor conditions [133]. Vibration and acoustic emission data which can be tens of kilohertz can be critical for the monitoring of bearings [134], gears [135], motors [133], machining [136], drilling [137], welding [138] etc. CNN provides a good alternative algorithm to extract features automatically from such high-frequency data and make decisions, instead of artificial feature engineering which requires domain knowledge and experience.

In [139], Jing et al. conducted a comprehensive comparative study of CNN for gearbox condition monitoring tasks using vibration signals. They pointed out that using CNN to extract features automatically from vibration signals improved the accuracy of working condition classification by 10% compared with using manual feature engineering both in the frequency and time domains. In [140], Zhao et al. verified that in motor transmission monitoring, CNN is not only effective in extracting features of vibration signals, but they are also effective in resisting noise contamination, which is widely present in vibration signals. In their experiment, different types of noise with signal-to-noise ratios (SNRs) ranging from 5 to –5 dB were injected, such as white Gaussian noise, Laplacian noise, and pink noise. They found that noise only had a limited effect on the monitoring algorithm. In terms of multiple-sensor fusion problems, CNN-based methods can be easily extended to the multiple-sensor version. Researchers often reorganise data from multiple sensors into a two-dimensional matrix as input and then use CNNs for feature extraction. An example can be found in [141] as shown in Figure 2.14. In addition, due to the outstanding achievements of CNNs in the field of image processing, there is also some research in the literature using a similar approach to visual image processing for the analysis of high-frequency sensor data. For example in [142] and [143], the high-frequency signals can be converted to images (time-frequency spectrum) by Fast Fourier Transform (FFT) or



time
variable
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$ $x_{1,7}$ $x_{1,8}$ ...... $x_{1,n}$
$x_{p,1}$ $x_{p,2}$ $x_{p,3}$ $x_{p,4}$ $x_{p,5}$ $x_{p,6}$ $x_{p,7}$ $x_{p,8}$ ...... $x_{p,n}$
$x_{p+1,1}$ $x_{p+1,3}$ $x_{p+1,5}$ $x_{p+1,7}$ ...... $x_{p+1,n}$
$x_{q,1}$ $x_{q,3}$ $x_{q,5}$ $x_{q,7}$ ...... $x_{q,n}$
$x_{q+1,1}$ $x_{q+1,5}$ ...... $x_{q+1,n}$
$x_{m,1}$ $x_{m,5}$ ...... $x_{m,n}$

Figure 2.14.: Multiple sensors input [141]

Wavelet Transform, and then a CNN can be used to extract features from these images and make decisions.

2.4.3 Recurrent Neural Network (RNN)

In the previously mentioned ANN and CNN, the former input is not related to the latter input, which means they can be not ideal for modeling the time dependency. Although in the approach using CNNs the time series can be reorganised into a vector or matrix and then the convolution kernel can be applied to scan the entire sequence, the convolution kernel only extracts information from the data in the current perception field and does not take into account the influence of previous data on the current feature extraction process. In other words, the parameter-sharing mechanism of CNN will use the same set of parameters to extract features spatially from a sequence while ignoring time dependence. To introduce time-dependent modelling and increase the capabilities for modelling sequential data, RNNs were proposed [144]. The overall structure of RNN can be demonstrated in Figure 2.15. Generally, RNNs can be divided into 2

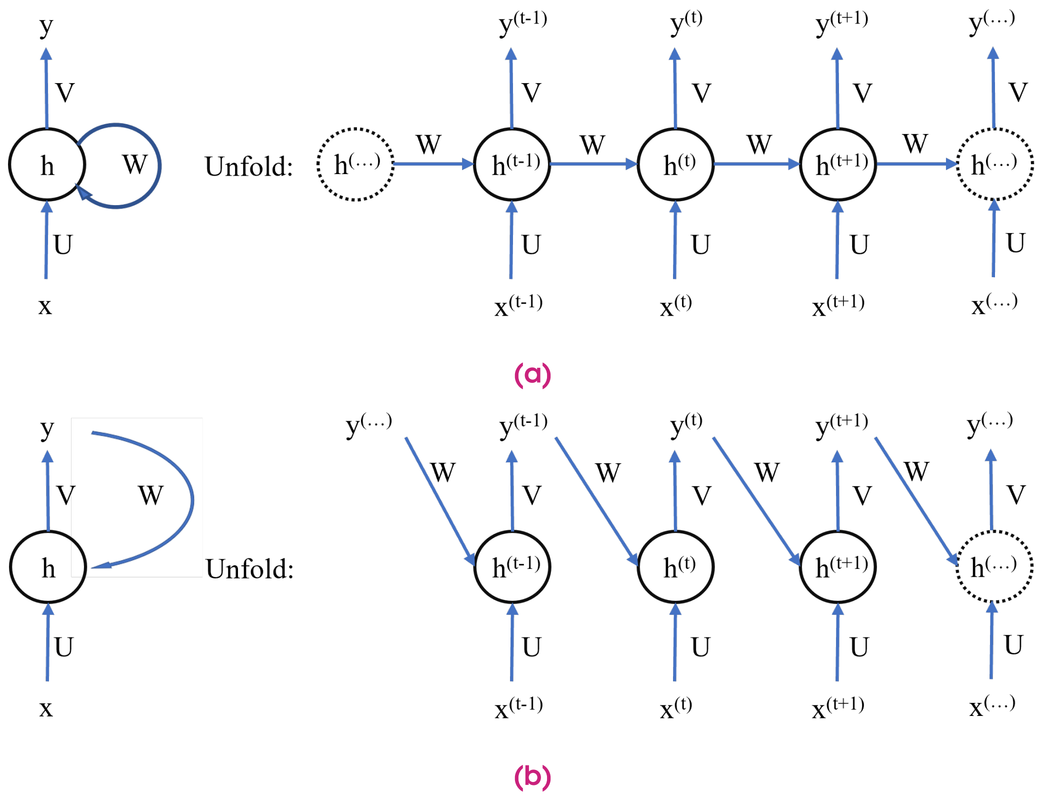


Figure 2.15.: Recurrent Neural Network

categories based on how to incorporate the previous output:

- Using the hidden states of previous output as the input of the current feature extraction as shown in Figure 2.15a. It can be described by the following equations:

$$\begin{aligned} \boldsymbol{h}^{(t)} &= \sigma\left(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}\right) \\ \boldsymbol{y}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \end{aligned} \tag{2.20}$$

  where, $\sigma(\cdot)$ is the activation function, $\boldsymbol{W}$, $\boldsymbol{U}$, and $\boldsymbol{V}$ are weight matrix, $\boldsymbol{b}$ and $\boldsymbol{c}$ are bias.

- Using the previous output directly as the input of the current feature extraction as shown in Figure 2.15b. It can be described by the following equations:

$$\begin{aligned} \boldsymbol{h}^{(t)} &= \sigma\left(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{y}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}\right) \\ \boldsymbol{y}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \end{aligned} \tag{2.21}$$

In RNNs, in contrast to CNNs, parameter sharing occurs in the time dimension, which means a set of parameters can be used to extract features at all time steps of the input sequence, and the output at previous time steps can also be incorporated as input to the current feature extraction process. Thus, RNNs can be better at modelling time dependence. In addition, as shown in Figure 2.15, the output of RNN at each of the hidden states can be accessible, enabling the sequence-to-sequence mapping, and extending the scope of application of neural networks significantly. However, it has to be mentioned that, due to the inherent recurrent operation of RNNs, the gradients tend to vanish or explode during the optimisation process as they have to propagate over a number of time steps. This means it can be difficult for RNNs to model a long sequence while keeping the information from the early time steps [145], also called the forgetting problem. As a result, a gating mechanism was introduced to RNN to address this. Generally speaking, this mechanism is able to alleviate the forgetting problem by the following strategies:

- The percentage of past information that can be used and the percentage of present information that can be stored in memory can be controlled by gates.
- Part of the information from the past can be integrated directly into the output without the recurrent operation.

Therefore, long-term dependence can be better preserved and harnessed. Among the network structures with this mechanism, Gated Recurrent Unit (GRU) [146] and Long Short Term Memory (LSTM) [147] are the most widely used. Of these, LSTM can be the more widely used in the field of industrial sensor fusion application in literature, and its structure can be demonstrated in Figure 2.16.

RNN is a very important algorithm in industrial process monitoring, especially in RUL estimation tasks, because the system's degradation patterns can be highly time-dependent (or work-cycle dependent) [148]. In RUL tasks, a number of sensors can be used to monitor the key parameters of a system, generating

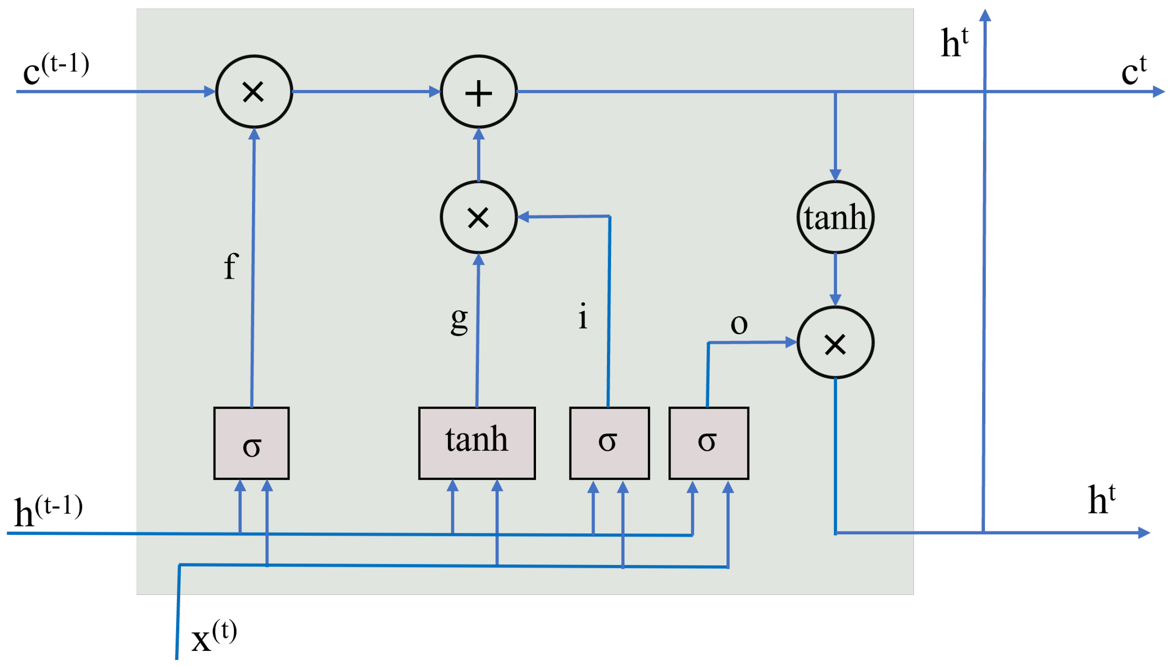


Figure 2.16.: Multiple sensors input [141]

multiple time series in different sampling rates, which makes the input space a complex and high-dimensional space [149]. The final target of RUL is to map the high-dimensional input space to a one-dimensional Health Index (HI) which indicates how far until the system fails. Therefore, the dimension reduction technologies and feature extraction in the multi-streams time series can be the keys to these tasks. Although many conventional sensor fusion algorithms might be the options for RUL estimation, such as PCA [150], isometric feature mapping reduction [151], and logistic regression [152], Wang et al. and Khelif et al. pointed out that they might potentially distort the degradation patterns, causing the large prediction errors when the system being evaluated is at the early or end of its life cycle [153] [154]. As a result, RNNs have attracted the attention of many researchers due to their excellent data-driven feature extraction capabilities and they do not require any assumptions or domain knowledge on degradation patterns [155]. Generally, multiple sensor data or manually extracted features can be the input of RNNs, and the output can be the estimated RUL. For example, in [156], LSTM was used to predict the RUL of electromagnetic pumps based on vibration and pressure sensors. Similarly, in [157], the original RNN and

LSTM were combined to estimate the RUL of the turbofan engine based on the data from 21 different sensors.

In addition, instead of using RNNs to find the mapping rules between sensor readings and RUL directly, there are many research works organised the RNN into an autoencoder structure to obtain a latent representation of the original multi-sensor data, and then the generated latent representation can be used as the input of different RUL estimation algorithms. This is because this type of method normally shows better generalisation ability and prediction accuracy [149]. For example, in [149], the authors embedded the original multi-sensor signals into a latent space by an RNN-based autoencoder, and then the generated representation was used to estimate the RUL of turbofan engines. A comprehensive review of this type of method can be found in [158].

2.4.4 Transformer and Attention Mechanism

Inspired by the fact that humans tend to focus only on the key information in the field of vision, the Attention Mechanism (AM) first appeared in the field of computer vision and gradually became a hot topic [159] [160]. Then, Bahdanau et al applied the AM to Recurrent Neural Network (RNN) to process natural languages. They found that the AM not only visualised deep learning models to some extent but also addressed the fatal flaw of RNN: the forgetting problem when processing long sequences [161]. Later on, researchers found that a deep model built entirely on the principles of the AM improved the performance in machine translation tasks considerably [162]. Here the RNN architecture was abandoned as it was believed that the forgetting problem was rooted in the large number of iterations of RNN, this model was named a Transformer [162].

The overall architecture can be illustrated in Figure 2.17. The original Transformer consists of encoder blocks and decoder blocks. While the encoder blocks extract features from the input sequence, the decoder blocks generate an output sequence based on the feature extracted from the input. The

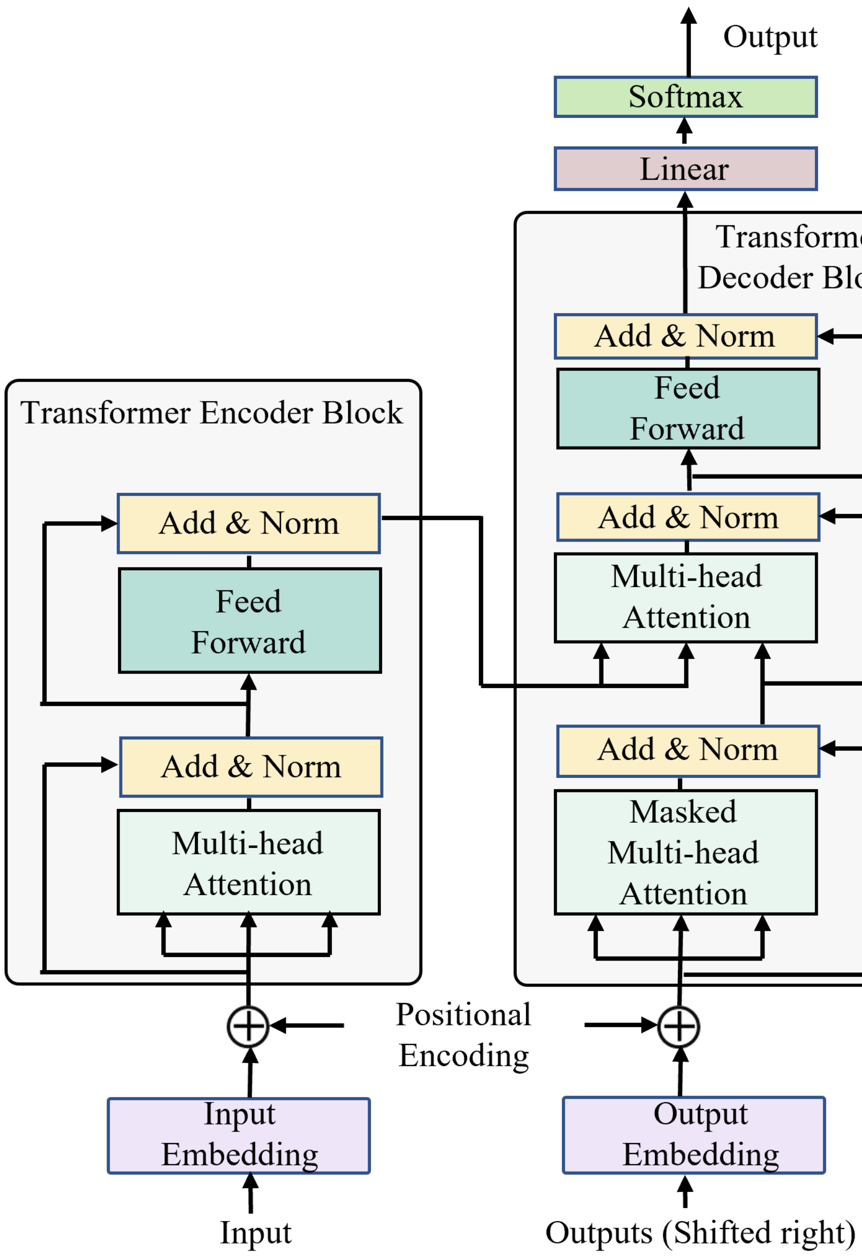


Figure 2.17.: Transformer Architecture

two Transformer blocks are composed of two main components: the multi-head self-attention mechanism and the fully connected feed-forward network, which can be the cores of the Transformer's function.

The self-attention mechanism is one of AM and is used to calculate the importance scores of each element in a sequence, and then each element can be weighted before further processing. It should be mentioned that the importance scores are obtained by a locally trainable ANN, which allows the Transformer to dynamically allocate its attention based on each input sequence, enabling important information to be filtered out effectively. In addition, the self-attention mechanism processes each element in a sequence in parallel and therefore offers the following benefits over traditional networks:

- Compared with the recurrent operation of RNNs, the parallel processing leads to the Transformer generally not having the problem of forgetting that exists in RNNs, making it more efficient for processing long sequences and less prone to vanishing gradients [162].
- Due to the inherent parallel data processing, the training process of the Transformer can make better use of high-performance AI acceleration chips, such as GPUs and TPUs, making the training process faster and more efficient.
- In contrast to the local pattern matching of CNNs, the self-attention mechanism treats the elements at different distances equally, making the long-range dependencies easier to be discovered [163] [164].

However, due to the self-attention mechanism can only perform linear projection which limits its modelling capacity, the fully connected feed-forward network was added to the Transformer after the self-attention mechanism to introduce non-linear transformation capacity.

The Transformer architecture was first proven to be effective in NLP tasks [162] and has achieved significant success until today, such as the Pre-trained Transformer (GPT) family [165] and the very famous ChatGPT recently [166]. Furthermore, it also showed strong modelling capabilities in other areas, such as Vision Transformer (ViT) for vision recognition [167] and Alphafold for protein structure prediction [168]. As a result, in recent years, the Transformer architecture begins to attract the attention of researchers in the field of industrial process monitoring, especially for processing multiple sensor data streams. In [169], Jing et al.

found that the original Transformer architecture can be effective for processing multiple streams of vibration signal processing in rotation machinery fault diagnosis. They first divided raw vibration signals into multiple segments at equal time intervals and reorganised these segments into a 2-dimension matrix as the input of Transformers. Although they did not make any adaptive modifications to the Transformer architecture and did not introduce any manual feature extraction, on the Case Western Reserve University Bearing Fault dataset [170], top-ranking accuracy was achieved. Similarly, in [171], a multiple time series air handling units monitoring system based on the Transformer was proposed. They compared the Transformer with GRU, a traditional time series processing neural network, and the experiment results showed that the transformer model outperformed the GRU in both performance and training easiness. However, although the Transformer has great advantages in discovering the long-range dependency in multi-streams time series, it can be less effective for local feature recognition and sensitive to noise [172]. Hence, Pei et al. explored the combination of CNN and the Transformer to overcome the drawbacks, as local feature recognition and noise resistance can be the strong points of CNN [172]. They employed the Transformer block as the primary feature extractor and performed a point-to-point mapping of the raw data before applying a CNN to extract the higher-level features as shown in Figure 2.18. They found that, compared with the original Transformer, CNNs, and Gating RNNs, the Transformer combined with CNN can be much better in accuracy under different levels of noise (SNR values with the range of-6-6 dB). A similar methodology can also be found in [173] and [174], where they applied CNN first before using the Transformer.

In addition to the end-to-end approaches mentioned above, the Transformer can be also used to find the non-linear transformation between the raw sensor data and the representative latent space, which can be very important for traditional process monitoring algorithms. For example, in Canonical Correlation Analysis (CCA) based process monitoring, multiple sets of many-to-one linear transformations of multi-sensor data need to be identified before calculating

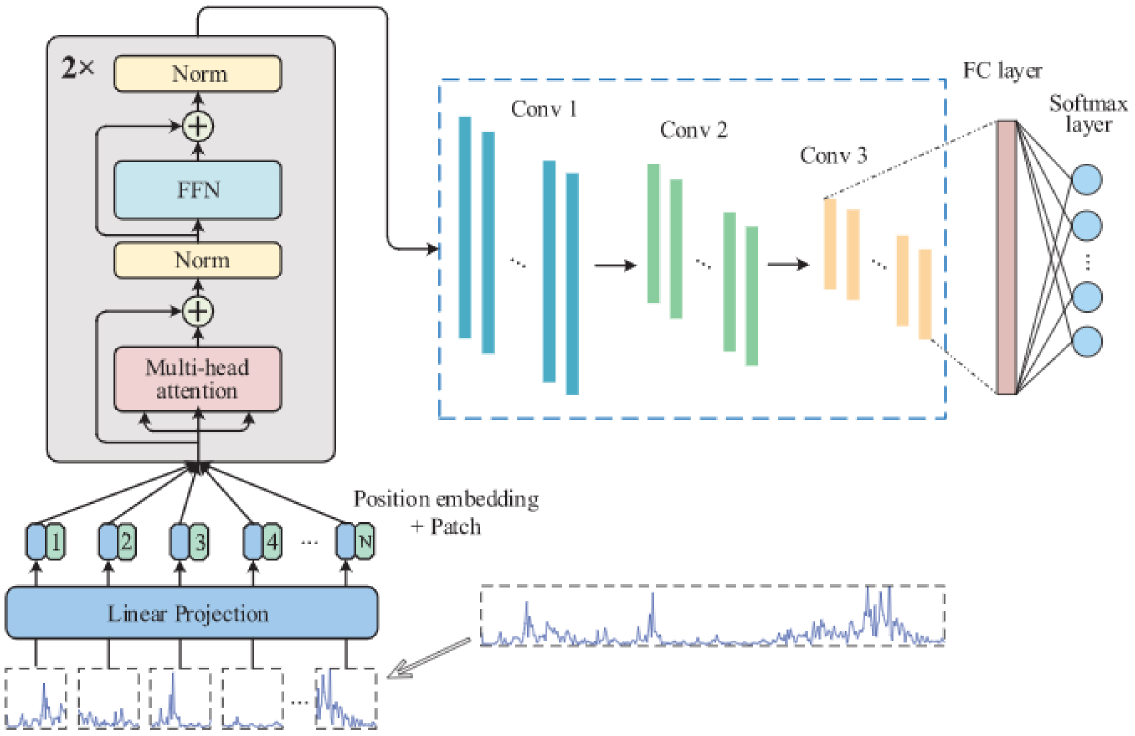


Figure 2.18.: Overall architecture of the model proposed in [172]

the correlation coefficients of the transformed data which can be used to evaluate the health of the system. The key to this type of method can be how to find the transformations that can properly represent the characteristics of the process being monitored [175]. In [175], Chen et al. replaced the linear transformations of the CCA-based monitoring algorithm with the Transformer to enable non-linear transformation, improving the performance of the CCA-based method for nonlinear dynamic processes monitoring. Similarly, in [176], the key contribution of their research can be the use of the Transformer to find the mapping function between raw sensor data and a latent space before further processing.

2.4.5 Deep Learning and its challenges

The concept of depth in deep learning implies an increase in the number of network layers. With the increasing depth of the neural network, the number of non-linear transformations that can be performed by the deep learning models increases, which enables the models to abstract more complex representations from input data and models complex relationships and patterns. This allows deep neural networks to extract features from complex data automatically and eliminates the need for artificial feature engineering, which can be an important reason why deep learning has been used and achieved great success in a variety of applications. However, training a deep network will not be as easy as training a shallow network, which can be reflected in three main aspects:

a) **Challenge 1: Difficult to train to convergence due to gradient vanishing or explosion** [177] [178]. During the backpropagation process by which the deep learning model updates its weights, the partial derivative of the loss function with respect to the weights will be calculated and then gradually propagates to the very first layers. When the network layers are deeper, the gradients are likely to be repeatedly multiplied by small numbers or large numbers in the chain rule of backpropagation, which results in the gradient vanishing or explosion problems respectively. This means that the model is unable to update the weights of the first few layers of the network, resulting in the model failing to converge [179]. Currently, there are many existing methods to address these problems, such as weight initialisation, gradient clipping, and normalisation techniques. Weight initialisation techniques can effectively balance the variance of the input and output at each of the layers, preventing the gradient from being too small or too large, enabling the training of deeper networks [180]. Gradient clipping is used to set an upper limit for gradients to void extreme weight updates that prevent the model from convergence. This technique was first found effective in practice and later proved mathematically in [181]. In terms of normalisation, such as batch normalisation and layer normalisation, they are usually applied before non-linear activation functions to stabilise the distribution of the input, which prevents the inputs from drifting into the saturation zone of the activation function, thus preventing the gradient from being updated [182] [183]. Using a combination of these techniques, the trainable depth of neural networks can be improved to tens of layers for stochastic gradient descent (SGD) with backpropagation [184].

b) **Challenge 2: After deep networks are able to converge, due to the overpowering fitting capability, they can be prone to suffer from overfitting and thus lose their generalisation capability, resulting in unsatisfactory results on the unseen data** [185]. This problem can be mitigated by regularisation techniques that can reduce the complexity of the model, such as dropout, weight decay, and early stopping techniques. Dropout techniques can reduce the structural complexity by randomly ignoring a certain percentage of the neurons within a layer during every training iteration, forcing the model to be more robust and tends to learn independent features [186]. The weight decay technique works by introducing a penalty term to the loss function, encouraging the weights to be small during optimisation. The commonly used regularisation can be the L_2 regularisation as shown in the following equation:

$$\boldsymbol{Obj} = \text{Loss} + \frac{\lambda}{2} * \sum_{i}^{N} w_i^2 \tag{2.22}$$

It can be found that in the process of minimising the objective function $\boldsymbol{Obj}$, the weights tend to be closer to zero due to the inclusion of penalty terms $\frac{\lambda}{2} * \sum_i^N w_i^2$, so the model with less complexity can be obtained compared with the model trained without the weight decay technique, preventing the overfitting problem [187]. As for the early stop technique, it simply stops training when the performance on the validation data begins to drop. Since while the model's performance rises in the training dataset, a drop in performance in the validation dataset is often the signal that overfitting has begun [188].

c) **Challenge 3: Degradation problem caused by the increasing depth.**

As reported in [189] and [190], the degradation problem indicates the deep learning models are likely to degrade while the depth of the network increases. This may not be a result of overfitting as the training error can increase with the increase of network depth. He et al. simply added identity mapping layers to a shallower architecture and found that the constructed deeper counterpart of the shallow network can not be comparably good or better [184]. To address the degradation problem and enable the deeper networks, residual learning was proposed in [184] as shown in Figure 2.19. Based on their experiment results, the

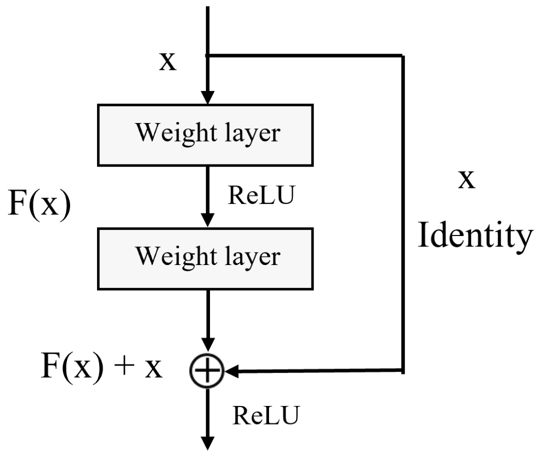


Figure 2.19.: Residual learning

difficulty of optimisation for deep networks with residual learning can be significantly decreased by allowing the information flow to skip one or more layers, and they achieved state-of-the-art performance among all models of the same periods at a public dataset. This may be the first time in the development of deep learning that extremely deep networks have started to take advantage, and the degradation problem can be still acceptable even though the depth of their proposed model was increased to 1000. The resulting models were called Residual Nets (ResNets), and they can be some of the first deep models to perform well both in academia and commercial applications [123].

With the above challenges being solved or mitigated, the depth of trainable networks has been increased to a very high level and shows very high performance in areas with large data volumes. Examples include the AlphaFold 2, trained from a database of 170,000 proteins provided by UniPort [191], revolutionising the field of protein spatial structure prediction, and ChatGPT based on GPT series, trained by the texts from the internet which can be almost regarded as an infinite database [192]. However, in the field of multi-sensor industrial process monitoring, the application of models with large depths can be often limited, and this problem will be discussed in the next section.

2.4.6 Challenges of Deep Learning for Industrial Process Monitoring

Models with large depths offer great advantages in extracting complex patterns from the data, which is one reason why deep learning has become an important algorithm in industrial multi-sensing fusion in recent years. However, there are many challenges with deep learning-based algorithms in industrial applications that prevent their large-scale application.

High demand for training data volume

The first challenge is that the amount of industrial multi-sensor data is often insufficient to train deep learning models. Industrial processes are usually nonlinear, dynamic, and space-time complex, resulting in high dimensional, multimodal and multi-sampling frequency sensor data [5]. Therefore, it is often necessary to build large-depth models to extract features from such complex input space, since the sufficient depth of a deep learning model is often related to the complexity of the data [193]. Hence, when the amount of data is insufficient for training a deep neural network, researchers often have to use manual feature extraction combined with shallow networks to process multi-sensor data [130] [194] [195]. As manual feature extraction inevitably introduces information loss and often requires domain knowledge, the powerful feature extraction capabilities of deep learning with end-to-end benefits can not be fully exploited.

However, although industrial digitalisation is accelerating, the amount of industrial data accumulated is still far from adequate. This is because sensor data acquisition for industrial scenarios is often not an easy task due to the following reasons:

- Placement, selection and maintenance of sensors require specialist skills and experience, and investment in software and hardware.
- The quality of the sensor data can be affected by sensor noise, drift, calibration, and harsh industrial environments.
- Data integration of multiple sensors can be challenging due to the need for careful calibration and synchronisation of the sensors measuring different modalities.
- Data security-related issues may prevent data holders from sharing the data, as the data may contain sensitive information about processes, equipment, and products.

As a result, many researchers are investigating the use of deep learning in a data-limited context. Transfer learning is one of the feasible methods which aims to leverage pre-trained models and transfer knowledge from related domains, to reduce the requirement for training data. In transfer learning, deep networks are first pre-trained in the source domain and then have most of their weights frozen or fine-tuned in the target domain at a very small learning rate, thus reducing the amount of data required in the target domain. There are two different approaches to performing transfer learning in the field of industrial process monitoring. The first approach is to use the same or similar industrial processes as the source domain to pre-train the models. For example in [196], Sun et al. transferred the model trained from a machining tool to a different machining tool to predict the RUL. In [197], Mao et al. pre-trained their model from different gearing systems before tuning the model at the target gear being monitored. Both studies achieved high performance and clearly showed how transfer learning can help reduce training data. The second approach is to convert the sensor signals into 2D images and then use a pre-trained vision model to extract features. This method has demonstrated excellent performance when processing high-frequency signals. In [30], Shao et al. first transformed the vibration data into images by wavelet transform, then extracted their features with a model trained from ImageNet [198], and finally trained a shallow classifier using data from the target domain, thus achieving the goal of using deep learning with a small amount of industrial data for the monitoring of motor transmission system. Similar research can also be found in [199].

However, although transfer learning can be effective in reducing the training data, it has strict restrictions. Firstly, industrial processes can be highly diverse, making similar industrial process data may be equally unavailable. If the source domain is too dissimilar to the target domain, transfer learning can be ineffective. Secondly, computer-vision-based transfer learning, which relies on converting sensor signals into two-dimensional images, currently only has evidence of good performance on high-frequency data in the literature, since the time-frequency features which can be represented by images can be good representations of this kind of signals. This kind of transfer learning can be very rare in processing other numerical sensor data and multi-sensor data because it is difficult to find valid image-like representations of them.

Lack of Interpretability

The second challenge is that deep learning models usually lack interpretability, which means that their internal working mechanisms and how they make predictions are not transparent. In process monitoring, the black-box nature can be problematic, as it can be difficult to identify the root cause of undesired conditions that arise, especially for safety-critical monitoring where process failure may have serious consequences. In contrast, a lot of the traditional approaches, such as Decision Trees, Linear Regression, and NARMAX, can be highly interpretable. These methods can provide information on which input variables and how they contribute to model decisions, making the analysis of potential problems possible. Therefore, the use of deep learning in the industry is currently limited compared to traditional methods.

High Computation Consumption

High computational resource consumption is another major challenge for deep learning in industrial process monitoring. The success of deep learning relies heavily on the increase in modern computing capabilities, as it can be extremely resource-consuming (computing power of the chip, memory and electricity, etc) [200]. In industry, however, it can be much more difficult to obtain computational resources for the same capabilities than in other areas. Industrial personal computers (IPCs) have high requirements for durability and reliability because they usually operate in harsh environments, such as high temperatures, dust, moisture, and vibration, and they are designed to work for a much longer time with zero errors than normal PCs [201]. As a result, computation resources for industrial scenarios can be limited and costly, making the industry conservative about the use of large-depth models in process monitoring. In addition, for industrial applications, a constrained time budget can be usually faced, as the decisions made by monitoring algorithms can be sometimes integrated into the control system to enable real-time response based on the system conditions [202]. Hence, real-time performance can be also important. As a result, reducing the computation complexity of deep learning and the required computational resource consumption is an important direction in industrial applications, especially for applications in IIoT and edge computing scenarios [203].

2.5 Research Gaps

Based on the literature review and the challenges of applying deep learning in industrial sensor fusion summarised in the previous section, the research gaps are identified and summarised in this section.

a) Research gap 1: Despite the many advantages of the Transformer architecture and Attention Mechanism over other deep learning algorithms and the success it has achieved in a variety of areas, it has had relatively less application in the area of multi-sensor fusion for industrial process monitoring in the literature. Apart from vision and natural language, it remains to be answered whether and how the Transformer and Attention Mechanism can be applied to the field of multi-sensor information processing. Objective 1 can be an attempt to fill the gap mentioned above. This study is the first to use a Transformer-like architecture in the task of the dataset used in the literature and achieved state-of-the-art performance among published papers using the other deep learning algorithms.

b) Research gap 2: The development of a unified methodology for creating feature representations for multi-sensor data is still to be investigated. In the literature, the majority of multi-sensor fusion research relies heavily on manual feature extraction to build up a multi-sensor feature representation. This approach can be labour-intensive on the one hand, and on the other hand, relies on domain-specific knowledge and it may introduce a high degree of uncertainty about the representativeness of the extracted features. Objective 2 is expected to provide a novel data-driven approach based on deep learning algorithms for the establishment of multi-sensor feature representations. In contrast to conventional feature engineering, this method is expected to develop an end-to-end deep learning approach to automatically model sensor data and their dependencies across different modalities, thus avoiding complex and uncertain artificial feature engineering.

c) Research gap 3: As explained in the previous section, the current use of deep learning in industrial scenarios faces the challenge of insufficient data. While there are many approaches in the literature that use transfer learning to alleviate this problem, generally such approaches still require data from similar processes, which can be equally difficult to have. Research on whether models trained in data-rich but non-similar processes can be transferred to the field of multi-sensing fusion can be hard to find in the literature. Objective 3 is to contribute to this gap. To the best of my knowledge, the research for achieving Objective 3 is the first research that adapts the deep model (the Transformer) trained from natural language to the industrial sensor fusion domain, reducing the required volume of training data. In addition, in the literature on sensor fusion based on deep learning methods, there is little research on how to identify the key sensors for the final decisions, this is because of the black-box nature of deep learning. Objective 3 also contributes to improving the interpretability of deep learning in sensor fusion tasks.

d) Research gap 4: Due to the high computational complexity of the Transformer, although it has the advantage of featuring learning, it can be inefficient when dealing with large numbers of sensors, which significantly limits its application in industry. However, the Transformer can be a relatively new model in the field of sensor fusion applications and has received little research in the literature aimed at optimising it to make it more suitable for processing multi-sensor data. Objective 4 can be the attempt to optimise its computational efficiency, specifically for sensor fusion tasks.

2.6 Summary

This chapter systematically reviewed the application of multi-sensing fusion algorithms in industrial process monitoring. Firstly, in Section 2.1, the different types of industrial process monitoring and their respective advantages and disadvantages are described, followed by Section 2.2 where the overview of sensor fusion technologies was introduced. In Section 2.3, the historical development of sensor data processing technologies for the industry was discussed, as well as some commonly used conventional sensor fusion algorithms. Finally, in Section 2.4 the role of deep learning in process monitoring with respect to different algorithms was reviewed. The challenges of deep learning for the industry were listed in this chapter, namely, lack of data, lack of transparency, and high computational complexity. This thesis aims to address or alleviate these challenges and strives to enable industrial scenarios to benefit from the development of deep learning and artificial intelligence technologies.

3

Sensor Anomaly Detection Using Dual Channel Attention Mechanism in Automated Vehicles

3.1 Introduction

With the digitalisation process of the industry, Cyber-Physical Systems (CPS), enabled by advanced communications and computing technologies such as IIoT, Artificial Intelligence (AI), and others, are playing an increasingly important role in the industry [204]. As a consequence, industrial systems are increasingly dependent on the data provided by sensors, which places greater demands on the reliability of sensor data. In many industrial scenarios, sensors are heavily used to monitor various processes, providing a large amount of data that can be analysed to extract valuable information about the underlying physical processes and even integrated into the control or decision system [204]. Hence, anomaly detection can be essential where unexpected sensor behaviour can cause serious consequences, such as in manufacturing, transportation, and healthcare. The aim of sensor anomaly detection is to identify the readings that deviate from the expected values, which can result from many reasons, such as sensor malfunction and cyberattacks [205]. As a result, timely intervention can be enabled to avoid potentially severe consequences.

This work proposes a Dual-channel Attention-based CNN (DA-CNN) based on a self-attention mechanism for sensor anomaly detection. In the self-attention mechanism, data from all time points can be processed in parallel, instead of the recurrent computation in RNNs, making it significantly better than RNNs in capturing long-range dependence without suffering from forgetting problems [162]. Due to the advantages of CNN in local feature extraction, the proposed DA-CNN combines two different deep-learning building blocks, self-attention and CNN, to improve its performance for anomaly detection in multi-sensor time series, especially for mild anomalies.

3.2 Problem Statement

The concept of sensor anomaly refers to the situation in that sensor readings deviate from their expected values by producing faulty data and failing to reflect the actual physical processes, causing potential errors in decisions that depend on these readings. This work focuses on developing a supervised learning-based method to identify anomalous data in multi-sensor time-series signals to avoid serious consequences. The output of the proposed model can be the anomaly conditions of the current time window, hence this work formulated the anomaly detection problem as a multivariate time series classification (TSC) problem. According to [206] [207], the commonly observed sensor abnormal patterns can be summarised by the following types:

- Instant: a sudden and unanticipated alteration in the sensor time series between two adjacent normal readings as shown in Figure 3.1.
- Constant: a temporary and stable observation that deviates from the

normal readings and lacks correlation with the underlying physical processes as shown in Figure 3.2.

- Gradual drift: a slow and incremental shift in observed data over a given time frame as shown in Figure 3.3, leading to a significant difference between the sensor readings and the actual system state over time.
- Bias: a temporary and constant deviation from the normal readings as shown in Figure 3.4.
- Miss: no readings from sensors, which can be regarded as 'instant' or 'constant' from an anomaly detection point of view according to [46].

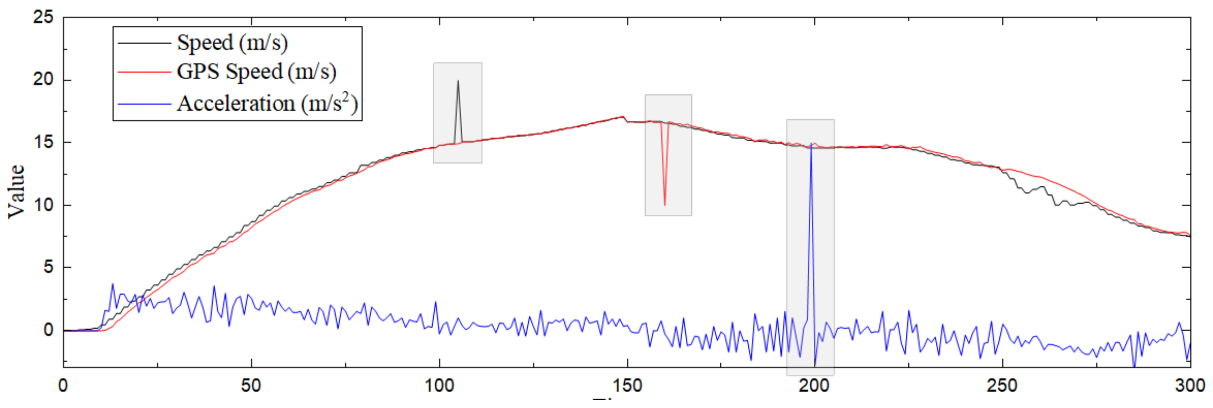


Figure 3.1.: Instant Anomaly

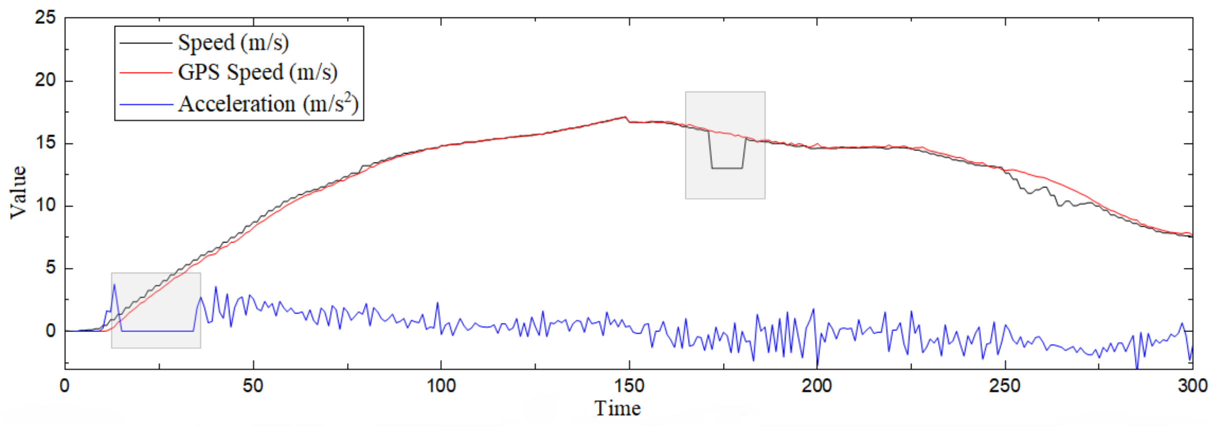


Figure 3.2.: Constant Anomaly

In order to maintain comparability, the CAVs dataset used in [46] and [52]

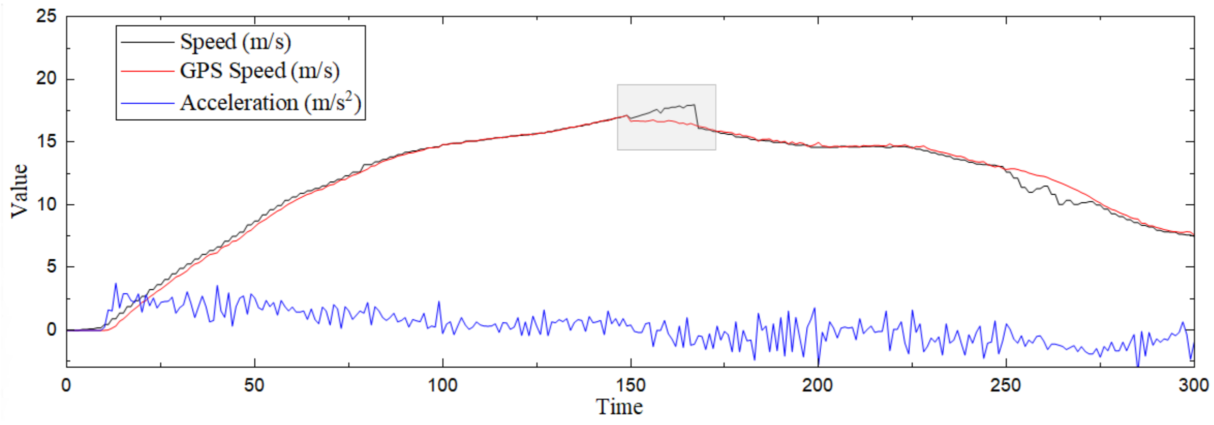


Figure 3.3.: Gradual Drift Anomaly

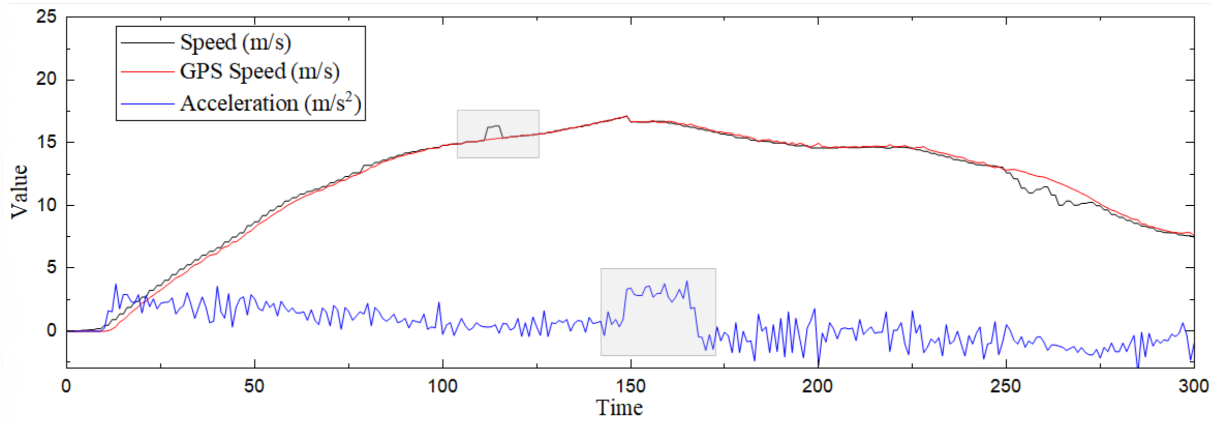


Figure 3.4.: Bias Anomaly

was employed. This dataset was provided by the Safety Pilot Model Deployment (SPMD) program as reported in [208] and can be found in their research data exchange (RDE) database [22]. This dataset was composed of a variety of information generated during the driving process of vehicles, including the Basic Safety Messages (BSM), driving trajectories, driver-vehicle interaction data, and environmental information, collected by the onboard Data Acquisition System (DAS), Global Positioning System (GPS), and roadside units. To keep consistent with [46] and [52], in this work, three sensors were employed to evaluate the proposed sensor anomaly detection method, namely, (1) Sensor 1: speed measured on the vehicle, (2) Sensor 2: speed given by GPS, and (3) Sensor 3: in-vehicle acceleration. As the environment of driving was not controlled and the car driving decision (acceleration or deceleration) can also be considered random, therefore the speed information of the car can be assumed as a non-stationary process which can be expressed by the following equation:

$$y_t = y_{t-1} + u_t, u_t \sim \text{IID}\left(0, \sigma^2\right) \tag{3.1}$$

where y_t denotes the speed or acceleration at time t, IID means independent and identically distributed.

Since this dataset did not provide anomalous sensor data, the same anomaly injection algorithm was used as in [46] and [52] as shown in Algorithm 1 to model sensor anomaly behaviour caused by cyber-attacks and sensor failures. As stated in [46] and [52], in a practical system, the probability of two sensors being abnormal at the same time can be very low due to the high reliability of the sensors, so we assume that only one sensor can be abnormal at the same time point. For anomaly types, the first four of the five patterns mentioned above were used, which can be some of the most threatening to CAVs [209] [210].

Algorithm 1: Anomaly Injection

```
$t \leftarrow$ time step
$T \leftarrow$ total number of data points
$p \leftarrow$ anomaly possibility
$i \leftarrow$ sensor index
$n \leftarrow$ number of sensors
$x_i \leftarrow$ sensor data
$x_i' \leftarrow$ sensor anomaly
for $t \in T$ do
    if $a \sim U(0,1) \leq p$ then
        $i \leftarrow randint(n)$
        $x_i' \leftarrow x_i + Anomaly$
    else
        $x_i' \leftarrow x_i$
    end if
end for
```

It is important to note here that, in this study, no data normalisation operation was used here. This is because, firstly, if sensor anomalies were injected after normalisation, those severe outliers would be significantly different from the normal values and thus make the task less difficult. Secondly, if the data were normalised after anomaly injection, those severe outliers would cause the variance of the original data to become extremely small, thus increasing the demand for the precision of the data processed by the model and resulting in an unnecessary computational burden.

3.3 Methodology and Architecture

In this section, an introduction to the overall model architecture of DSA-CNN is provided along with a detailed explanation of the two fundamental building blocks of DSA-CNN, namely DAM block and CNN block. This is followed by an explanation of how the DAM block extracts and integrates the temporal patterns (time-wise attention) and spatial patterns (sensor-wise attention) based on the self-attention mechanism.

3.3.1 Overall Architecture of Dual-channel Attention CNN

The overall architecture can be described by Figure 3.5. The multiple sensor time series data will be first passed into the class token concatenation function which will be explained in detail in the following section to add a class token that will be used as the feature representation for classification. Then, the output of class token concatenation can be fed to the Dual-channel attention mechanism (DAM) block to generate the feature maps of the attention mechanism, followed by a CNN block to further extract features. The DAM block and CNN block will be repeated several times before using a linear layer to perform the final classification. The output of this model can be the anomaly conditions of the data being passed to this model.

In terms of the input, let X denote the multiple sensor streams as shown in the following expression:

$$X = [X_1, X_2, \ldots, X_C] \tag{3.2}$$

where X_C denote the C-th sensor data stream. Each sensor stream can be composed of a series of readings over a predefined time window as shown in the following equation:

$$X_C = [x_1, x_2, \ldots, x_L] \tag{3.3}$$

where L is the number of readings in a time window. Hence the input space of the model can be expressed by $\mathcal{X} \in \mathbb{R}^{C \times L}$.

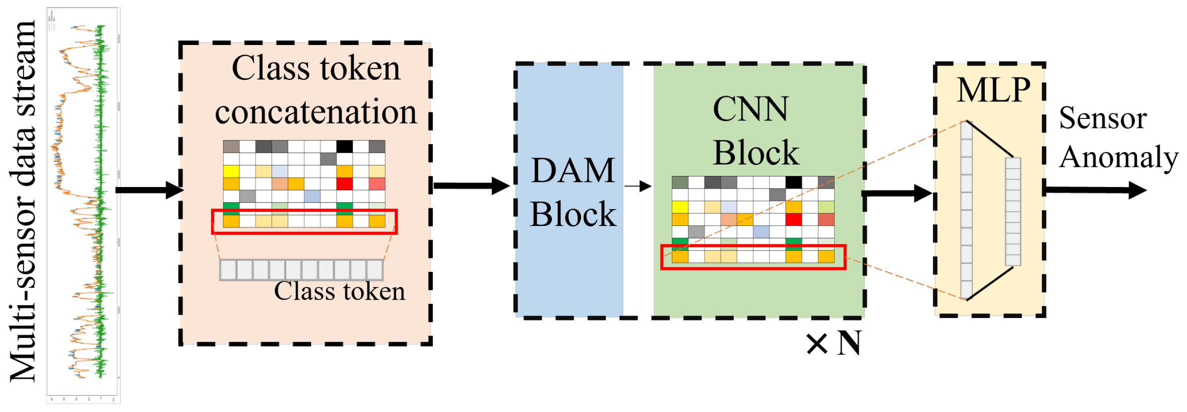


Figure 3.5.: DA-CNN Architecture

3.3.2 CNN Block

The proposed CNN Blocks employ 1d-CNN as a basic building element, and the architecture can be demonstrated by Figure 3.6. As shown in Figure 3.6, the

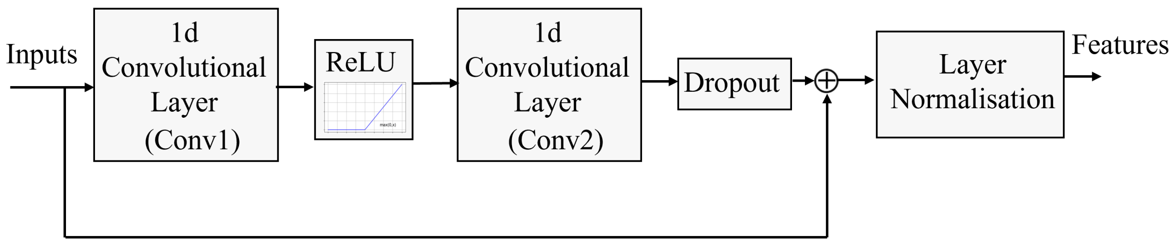

Inputs
1d Convolutional Layer (Conv1)
ReLU
1d Convolutional Layer (Conv2)
Dropout
Layer Normalisation
Features

Figure 3.6.: CNN Block

inputs can be first processed by a 1d-CNN layer and fed into a ReLU activation layer, before passing to the second 1d-CNN layer. Then, after the dropout operation, the original inputs can be added to the current feature maps via a shortcut connection. Finally, a normalisation layer is employed to form the final feature maps. The computation flow can be described by the following equation:

$$Out_{CNN} = \mathrm{LN}(\mathrm{Dropout}(\mathrm{Conv2}(\mathrm{ReLU}(\mathrm{Conv1}\,(X)))) + X) \tag{3.4}$$

where LN denotes the Layer normalisation, $Conv1()$ and $Conv1()$ are the convolution operation, $ReLU()$ is the activation function, X is the input multiple sensor time series.

1d-CNN

CNNs have been proven effective and achieved some of the state-of-the-art results for TSC among data-driven methods in literature [211] [212]. This is due to the sensitivity of CNNs to local features can be higher and their feature learning can be spatial invariant, compared with ANNs that treat all features equally [123]. In this study, 1d-CNN was used here instead of the normal CNNs with two-dimensional convolutional kernels. This is because the attention mechanism was relied on to model the spatial dependencies, CNNs were only expected to focus on a single series. For using 1d-CNN to TSC, the kernel size can be a key factor affecting performance [213]. Most of the works using 1d-CNN regarded the kernel size as a hyper-parameter and optimised it by grid search which can be time-consuming and computation-expensive. As this work does not rely on 1d-CNN to capture the major spatiotemporal features, the kernel size will not be regarded as a hyper-parameter and it was simply restricted to (1, 3).

The computation process of 1d-CNN can be defined as the following equation:

$$C_j = \mathrm{b}_j + \sum_{k=0}^{C_{\text{in}}-1} \mathrm{W}_{j,k} * \mathrm{X}_k \tag{3.5}$$

where X is the multi-variate time series, $*$ denotes the convolution operation, C_j is the output of the $j-th$ convolution filters, C_{in} is the number of input convolution channels, b and W are the biases and weights of the filters respectively. In the proposed CNN blocks, the dimensionality of the output feature map can be kept equal to the dimensionality of the input data to maintain extensibility. This can be achieved by: (1) organising the number of output channels of the second convolutional layer to be equal to the number of input channels of the first convolutional layer, (2) configuring the convolutional kernels to 3 in size with a stride of 1 and padding of 1. This makes the length of input and output can be kept the same as shown in the following equation:

$$L_{out} = \frac{L_{in} + 2 \times padding - (kernel\_size - 1) - 1}{stride} + 1 \tag{3.6}$$

Activation Layer

As the convolution operation can be considered as a linear mapping, a non-linear activation layer is used after the first 1d convolution layer to introduce non-linear modelling capabilities and thus enhance the representativeness of the learned features, enhancing the capacity to model complex input space. In the CNN Blocks of the proposed model, the ReLU is employed as the activation function, which can be expressed by the following equation:

$$\mathrm{ReLU}(H) = \max(0, H) \tag{3.7}$$

where H is the feature map passed to ReLU. This is because ReLU has a significant advantage over other activation functions such as tanh and sigmoid in terms of computational complexity, mitigation of the gradient saturation problem, and sparsity [214].

Layer Normalisation

The normalisation layer can be a critical component in the deep neural network. By normalising the inputs to a layer to zero mean and unit variance, the training process can be stabilised and the performance of the network can also be improved [182] [183]. Batch Normalisation (BN) and Layer Normalisation (LN) can be the prevailing normalisation methods. While BN normalises each feature within a sample, LN normalises all features within each sample. To be specific, BN erases the relative magnitudes of the different features but preserves the relative magnitudes of the different samples. Hence, BN can be more effective when the statistical behaviours among different samples are more important. In contrast, LN wipes out the relative magnitudes between different samples but keeps the relative magnitudes between different features within a sample. Therefore, it can be more suitable for tasks where the features within a sample are closely related. Speaking to the TSC task on CAV with unmanaged driving behaviour, as the speeds and acceleration can be treated as non-stationary processes, the statistical features among samples can be very uninformative for sensor anomaly detection. As a result, LN can be more suitable for this task.

LN performs normalisation over the last 2 dimensions based on the following equation:

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta \tag{3.8}$$

where y and x denote the normalised matrix and input matrix respectively, ϵ is a minimum preventing the denominator to be zero, γ and β are trainable affine transform weight and bias. $\mathrm{E}[x]$ and $\mathrm{Var}[x]$ denote the mean and variance of input x, which can be defined by the following equations:

$$E(x) = (1/t) \sum_{i=1}^{t} x_i \tag{3.9}$$

$$\mathrm{Var}[x] = (1/t) \sum_{i=1}^{t} (x_i - E(x))^2 \tag{3.10}$$

3.3.3 Dual-channel Attention Mechanism (DAM)

In multiple sensor anomaly detection tasks, abnormal data can appear at any time and in any sensor, making both temporal and spatial (sensor-wise) features very important. In this work, the Dual-channel Attention Mechanism (DAM) was proposed to integrate the learning of spatiotemporal features into the training process, hence the spatiotemporal features can be extracted progressively and automatically.

The overall architecture of DAM can be illustrated by Figure 3.7. The input

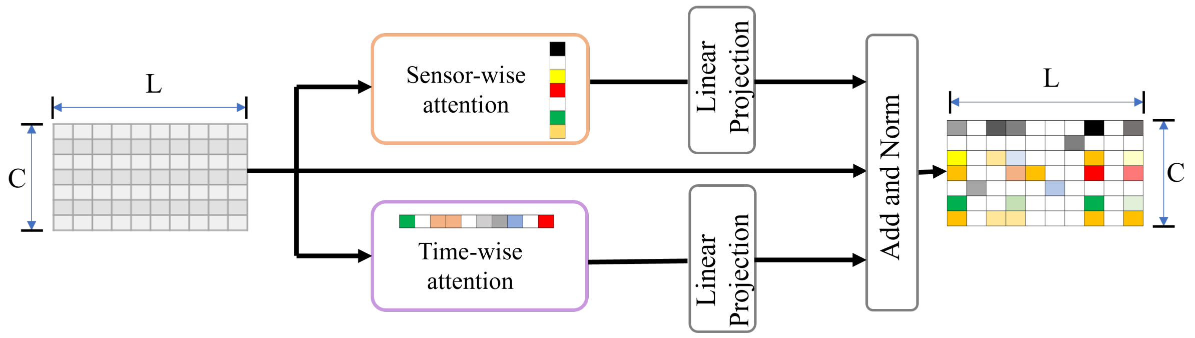


Figure 3.7.: Dual-channel Attention Mechanism (DAM)

can be passed to two attention modules simultaneously, namely sensor-wise attention and time-wise attention. The former is responsible for extracting spatial dependencies among different sensors and the latter is for extracting temporal dependencies among different time points. Then, the feature maps generated by these two modules and the original input will be added together, forming a shortcut connection, before normalisation. The output of the DAM is therefore a feature map with integrated spatiotemporal features. The computation flow of DAM can be described by the following equation:

$$OUT_{DAM} = \mathrm{LN}(\mathrm{LP}(\mathrm{SWA}(X)) + \mathrm{LP}(\mathrm{TWA}(X^T)^T) + X) \tag{3.11}$$

where LP denotes linear projection, SWA and TWA mean sensor-wise attention and time-wise attention respectively. The details of sensor-wise attention and time-wise attention will be explained in the following section.

Sensor(Time)-wise attention

The sensor-wise attention processes the dependencies among different sensors based on the self-attention mechanism proposed in [162] as shown in Figure 3.8.

The self-attention could be analogous to retrieval systems: a query vector is used to search information and then the search engine will try to look for the keys in its database and pair the query vector, finally, the value vector corresponding to the keys will be the output. In the self-attention mechanism, the input sequences are mapped to the query vectors (Q), key vectors (K) and value vectors (V) by linear projection as shown in Fig. 3.8 and expressed by the following equations:

$$Y = XW^T + B \tag{3.12}$$

where $X \in \mathbb{R}^{C \times L}$, $W \in \mathbb{R}^{3L \times L}$, $Y, B \in \mathbb{R}^{C \times 3L}$, and

$$Q, K, V = Y[0 : C, 0 : L], Y[0 : C, L : 2L], Y[0 : C, 2L : 3L] \tag{3.13}$$

where $Y[a : b, c : d]$ denotes the a to b rows and c to d columns of Y.

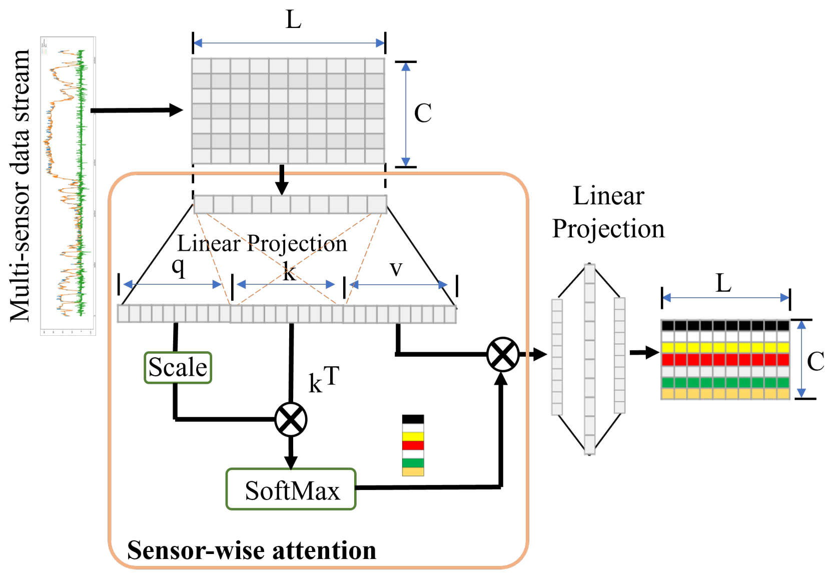


Figure 3.8.: Sensor-wise attention

The optimised mapping matrix (the weights of these linear layers) can be obtained in the backpropagation process. Then, the attention weight is calculated by:

$$Attention\_weitht = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \tag{3.14}$$

where d_k is the dimension of key vector and $1/\sqrt{d_k}$ is the scalar which is used to avoid the dominant term when calculating the softmax function, which may

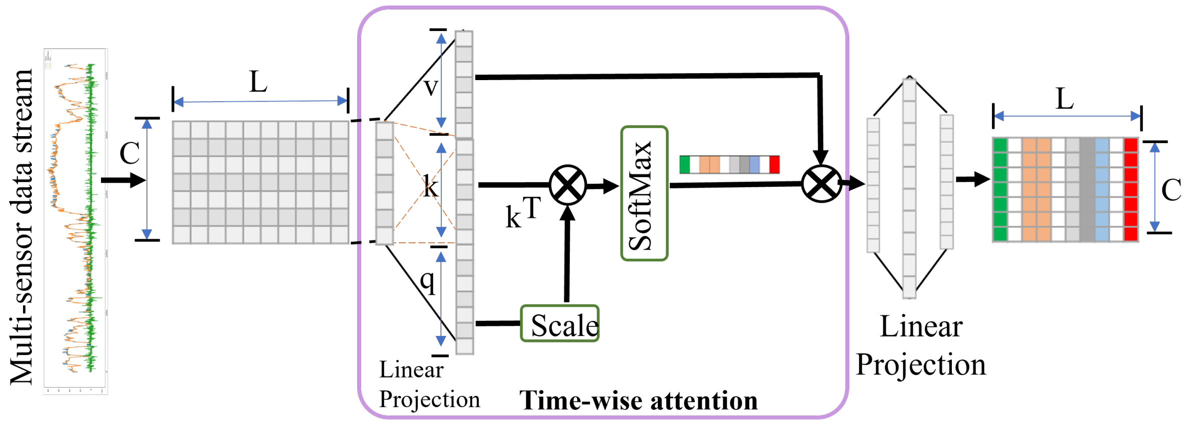


Figure 3.9.: Time-wise attention

make the gradient difficult to calculate [162]. If Q and K are independent and conform Gaussian distribution: $N(0,1)$, the variance of their dot product, $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$, will be d_k. This effect is not preferable when addressing high-dimension data. This AM is called Scaled Dot-Product Attention [162]. Finally, the attention weight is multiplied by the value vector and the final weighted mapping is obtained as shown in equation 3.15,

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.15}$$

It is obvious that the attention weights control the information flow within the network. When the weight of a certain area of input data becomes zero, no information can flow to the next layer of the network. In addition, each element (row) in the attention output sequence is the attention evaluation result of the entire input sequence. Hence, the dependencies among different sensors can be modelled and integrated. It can be found from the above calculations that, the dimension of the output feature maps remains the same as the dimension of the input data, namely $Attention(Q,K,V) \in \mathbb{R}^{C\times L}$.

Similarly, in time-wise attention, the base unit for carrying out the calculation of the attention mechanism can be replaced by the time steps, as shown in Figure 3.9. Time-wise attention can be simply achieved by transposing the input and then performing similar computations to obtain the Q, K and V before calculating the attention weights:

$$Y = XW^T + B \tag{3.16}$$

where $X \in \mathbb{R}^{L\times C}$, $W \in \mathbb{R}^{3C\times C}$, $Y, B \in \mathbb{R}^{L\times 3C}$, and

$$Q, K, V = Y[0 : L, 0 : C], Y[0 : L, C : 2C], Y[0 : L, 2C : 3C] \tag{3.17}$$

Therefore, the feature maps with the integrated time dependencies over the entire time window can be obtained. Then, the feature maps of Time-wise attention can be transposed to keep the equal dimensionality with Sensor-wise attention, before adding them to the original input to make a shortcut connection. Finally, the constructed feature map can be normalised, forming the output feature map of the DAM block as shown in Figure 3.7.

3.3.4 Class Token Concatenation

Based on equation 3.15, it can be found that each row of the output of the attention mechanism (corresponding to each sensor in this work) is a weighted sum of all rows of the feature map from the previous layer, and the weights are calculated based on the data from each sensor. Theoretically, it is reasonable to take any row in the final output of the model's feature map and use it as a multi-sensor feature representation to perform classification, however, to avoid the influence of specific sensor data on the feature representation of multi-sensor data, we added a vector of random numbers to the input data as a token for classification [162]. The features of all sensors can be weighted and integrated into this token, hence this vector can be used as a final feature representation. This operation is illustrated in Figure 3.5.

3.3.5 Algorithm

The overall computation flow of the proposed DA-CNN can be illustrated by algorithm 2.

Algorithm 2: Dual-channel Attention CNN (DA-CNN)

```
Input: $X \leftarrow$ CAV Sensor Readings
Output: Normal reading, Anomalous reading
1  $X \leftarrow X + Anomalies$                      # Anomaly injection
   Model Initialisation:
2  for l in range(Number of layers in DA-CNN) do
3      Initiate $Weights^{(l)} \sim N(0,1)$
4      Perform Singular Value Decomposition to $Weights^{(l)}$
           Ensure: $Weights^{(l)} Weights^{(l)T} = I$
   Class Token Concatenation:
5  $Token \leftarrow rand((1,L))$
6  $X \leftarrow Concatenate((X, Token), dim = 0)$
   Forward:
7  for i in range(N) do
8      # DAM block calculations
9      $X_{SWA} \leftarrow LP(SWA(X))$
10     $X_{TWA} \leftarrow LP(TWA(X^T)^T)$
11     $X \leftarrow X_{SWA} + X_{TWA} + X$
12     $X \leftarrow LN(X)$
13     # CNN block calculations
14     $Residual \leftarrow X$
15     $X \leftarrow ReLU(Conv1(X))$                 # Conv1
16     $X \leftarrow Dropout(Conv2(X))$              # Conv2
17     $X \leftarrow LN(X + Residual)$
18 $Outputs = \text{Classifier}(X) = XW_{classifier} + b_{classifier}$
   Training:
19 for epoch in range(Number of epochs) do
20     Calculate Loss
21     Calculate Accuracy, Precision, Sensitivity, F-Score
22     Backpropagation
Return: Output
```

3.4 Experiments and Results

3.4.1 Hyper Parameters and Training Process

The parameters of the model proposed in this work are shown in Table 3.1.

Table 3.1.: Hyper Parameters

	CNN Block
Conv1	channel_in = 15, channel_out=60, kernel size = 3, padding = 1, bias=False
Conv2	channel_in = 60, channel_out=45, kernel size = 3, padding = 1, bias=False
LN	ϵ = 1e-6
Dropout	0.2
	SWA
Self-attention	qkv dimension (L) = 15
Linear projection	Dropout = 0.1, in = 15, out = 15
	TWA
Self-attention	qkv dimension (L) = 3
Linear projection	Dropout = 0.1, in = 3, out = 3
	Classifier
Linear layer	in = 15, out = number of classes
	Training
Model depth (N)	7 (Number of DA-CNN blocks)
Initialisation	Orthogonal, Gain = 1.41
Batch size	16
Learning rate	0.0001

This experiment was conducted on Google Colab environment with NVIDIA Tesla P100 PCIe 16 GB, and PyTorch was used as the deep learning framework. As we formulated the sensor reading anomaly detection task as a classification task, the cross entropy loss was used as the loss function which can be defined by the following equation:

$$\mathrm{Loss}(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{3.18}$$

where y and $\hat{y}$ are the true labels and model prediction using one-hot encoding.

In terms of model initialisation, orthogonal initialisation was employed, namely, the weight matrix $W^{(l)}$ of each layer will be initialised to an orthogonal matrix, satisfying:

$$W^{(l)} W^{(l)^T} = I \tag{3.19}$$

where I is an identity matrix, and l is the layer index. This is because the orthogonal initialisation is able to make the error term norm-preserving during the backpropagation process [215], which can be expressed in the following equation:

$$\left\| \delta^{(l-1)} \right\|^2 = \left\| W^{(l)^T} \delta^{(l)} \right\|^2 = \left\| \delta^{(l)} \right\|^2 \tag{3.20}$$

where $\delta^{(l)}$ is the loss term of the l layer of the model, and it has shown great efficiency practically for training the model based on the attention mechanism [192] [216]. Since in the proposed model, the activation function was all ReLU and the average gradient of ReLU around 0 can be approximated as 0.5, the initialised weight matrix was multiplied by $\sqrt{2}$ to preserve norm-preserving property. This is denoted by $Gain = 1.41$ in Table 3.1.

3.4.2 Results Evaluation Method

The performance of the proposed method was evaluated by the following scores which can be normally used for assessing classification algorithms:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \tag{3.21}$$

$$\text{Sensitivity} = \frac{TP}{TP + FN} \tag{3.22}$$

$$\text{Precision} = \frac{TP}{TP+FP} \tag{3.23}$$

$$\text{Specificity} = \frac{TN}{TN+FP} \tag{3.24}$$

$$\text{F1 Score} = \frac{2(\text{ Sensitivity } * \text{ Precision })}{\text{Sensitivity } + \text{ Precision}} \tag{3.25}$$

where TP, TN, FP, and FN mean True Positive, True Negative, False Positive, and False Negative respectively. In terms of the sensor anomalies detection domain, as undetected abnormalities are far more damaging than classifying normal readings as abnormal, the sensitivity score was highlighted in the results. As the F1 score can be a comprehensive measure of precision and sensitivity, it was used as a reference to compare the performance of different models.

3.4.3 Single Anomaly Detection

In this section, the performance of DA-CNN is evaluated with respect to the occurrence of a single type of anomaly. To ensure comparability, the same anomaly pattern, severity, and possibility (5%) were used as in [46] and [52]. The results from CNN Kalman Filter (CNN-KF) [46], Multi-stage attention mechanism with an LSTM-based CNN (MSALSTM-CNN) [52], and the proposed DA-CNN, in instant, bias, constant and drift detection are compared in Table 3.2 to 3.5 respectively. The mean F1 score and sensitivity comparison for each type of anomaly is summarised in Figure 3.10.

Instant anomaly detection

Table 3.2 shows the performance of DA-CNN compared with the methods working on the same dataset in literature for instant anomaly detection. As the severity of the anomaly increases, an improvement in the performance of all methods

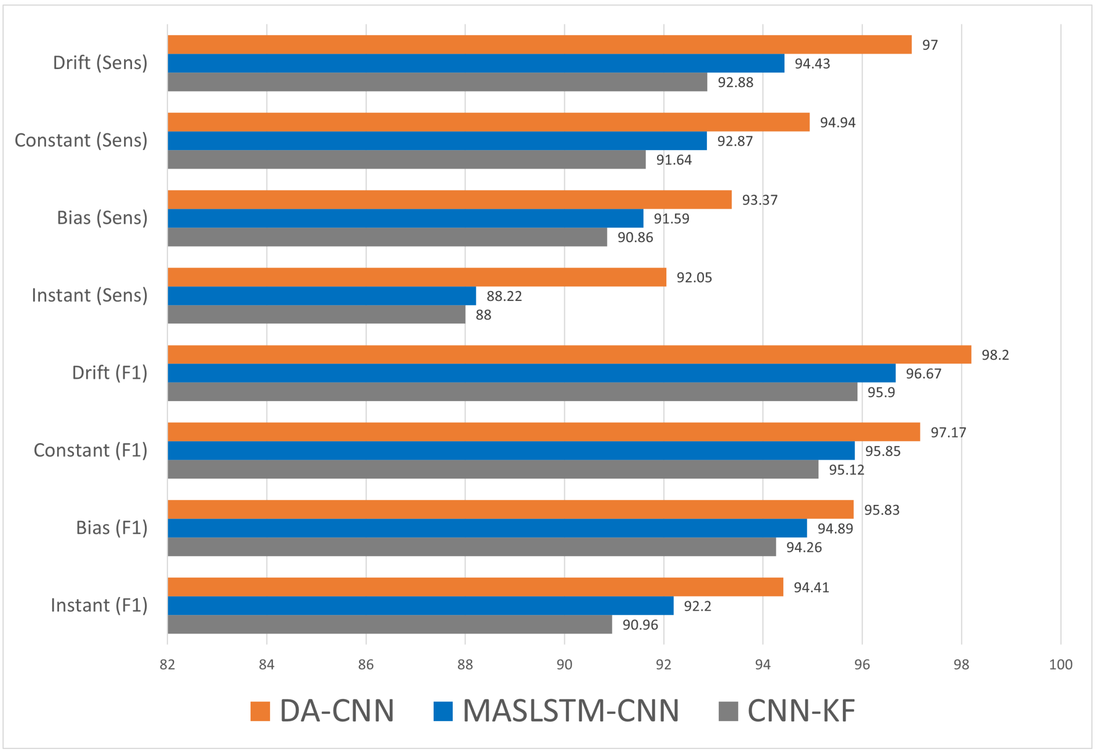


Figure 3.10.: Performance comparison of different models for single anomaly detection

can be observed. However, the proposed DA-CNN achieves higher F1 scores for all severity levels of instant anomaly, compared with the SOTA performance achieved by MSALSTM-CNN in the literature. Speaking to sensitivity, except for the magnitude of $500 \times N(0, 0.01)$, DA-CNN can be also better than the other methods significantly. It is worth noting that DA-CNN shows outstanding performance in detecting mild anomalies. While the F1 score of the MSALSTM-CNN at severity $25 \times N(0, 0.01)$ is 70.18%, the DA-CNN achieves 78.82%, and the sensitivity at this severity can also be improved by 16.65% by DA-CNN. In case of severe anomalies, DA-CNN shows a slight improvement.

Bias anomaly detection

Table 3.3 shows the performance comparison for bias anomaly detection. For all methods in this table, the longer the duration of the anomaly, the higher the detection performance of the models for the same magnitude. Similarly, for anomalies of the same duration, the higher magnitude of the anomaly, the higher

Table 3.2.: Instant

	CNN-KF(%) [46]				MSALSTM-CNN(%) [52]				DA-CNN(%)				
Anomaly Magnitude	Acc	Sens	Prec	F1	Acc	Sens	Prec	F1	Acc	Sens	Prec	F1	Spec
25 x N(0,0.01)	80.0	51.5	97.6	67.4	84.10	54.63	98.12	70.18	85.00	**71.28**	88.16	**78.82**	88.71
100 x N(0,0.01)	93.6	86.2	97.9	91.7	95.8	89.60	98.43	93.80	95.42	**91.00**	97.85	**94.30**	98.58
500 x N(0,0.01)	98.3	96.0	99.7	97.8	96.02	**99.79**	97.86	99.02	99.58	99.00	99.99	**99.50**	99.30
1000 x N(0,0.01)	98.8	97.1	99.8	98.4	98.98	98.16	99.21	98.68	99.58	**99.05**	99.99	**99.52**	99.27
10000 x N(0,0.01)	99.7	99.2	99.8	99.5	99.43	98.93	99.75	99.34	99.91	**99.92**	99.92	**99.92**	99.36

the detection performance. Based on the results in this table, the proposed DA-CNN achieves improvements in both sensitivity and F1 scores among most of the severity, compared to SOTA performance in the literature, except for the magnitude of $U(0,1)$ with a duration of 3, where MSALSTM-CNN achieves the best F1 score (90.57%) and ours is 90.38%. However, it can be found that DA-CNN has significant advantages in anomaly detection of small duration. In anomalies of duration 3 and 5, DA-CNN improved on average by 2.57% and 1.28% in sensitivity and F1 score respectively compared to MSALSTM-CNN.

Table 3.3.: Bias

		CNN-KF(%) [46]				MSALSTM-CNN(%) [52]				DA-CNN(%)				
Anomaly Magnitude	Duration, d	Acc	Sens	Prec	F1	Acc	Sens	Prec	F1	Acc	Sens	Prec	F1	Spec
U(0,5)	3	94.6	89.3	99.5	94.2	95.10	90.53	99.47	94.78	95.92	**93.86**	98.79	**96.26**	98.53
U(0,5)	5	94.8	90.6	99.6	94.9	95.51	91.99	99.55	95.62	96.13	**93.80**	99.76	**96.69**	97.35
U(0,5)	10	95.9	94.4	99.1	96.7	96.56	95.74	99.06	97.37	97.12	**96.69**	99.05	**97.84**	99.28
U(0,3)	10	94.4	95.6	98.6	95.5	94.99	93.82	98.50	96.10	97.55	**96.24**	99.73	**97.95**	99.86
U(0,1)	10	88.0	84.8	95.9	90.0	88.55	85.81	95.89	**90.57**	88.69	**86.18**	95.02	90.38	88.56

Constant anomaly detection

Table 3.4 shows the performance comparison for Constant anomaly detection. It can be observed that the performance of CNN-KF, MSALSTM-CNN, and DA-CNN methods consistently improves as the anomaly duration increases while keeping the anomaly magnitude constant (U(0,5)). Their performance decreases as the anomaly magnitude reduces while keeping the duration constant (d=10). The DA-CNN method, however, demonstrates superior performance in terms of sensitivity and F1 score across all experiments. In addition, the proposed method displays consistency in its performance, whereas the CNN-KF and MSALSTM-CNN methods exhibit some variation. Nevertheless, all three methods show slightly declining performance in the last row with the lowest anomaly magnitude (U(0,1)).

Table 3.4.: Constant

		CNN-KF(%) [46]				MSALSTM-CNN(%) [52]				DA-CNN(%)				
Anomaly Magnitude	Duration, d	Acc	Sens	Prec	F1	Acc	Sens	Prec	F1	Acc	Sens	Prec	F1	Spec
U(0,5)	3	94.9	89.9	99.6	94.5	95.05	90.25	99.51	94.65	95.73	**92.71**	99.04	**95.77**	99.89
U(0,5)	5	95.1	91.7	99.0	95.2	95.40	92.30	98.97	95.51	96.98	**94.15**	99.66	**96.83**	99.19
U(0,5)	10	96.2	94.9	99.1	97.0	96.61	95.61	99.28	97.41	98.59	**97.71**	99.59	**98.64**	99.56
U(0,3)	10	95.3	93.9	98.7	96.2	96.44	95.44	98.93	97.15	98.33	**97.18**	99.59	**98.37**	99.35
U(0,1)	10	91.2	87.8	98.6	92.7	93.02	90.76	98.69	94.55	96.56	**92.97**	99.76	**96.25**	99.90

Drift anomaly detection

Table 3.5 shows the results of gradual drift anomaly detection using the CNN-KF, MSALSTM-CNN, and DA-CNN models. The MSALSTM-CNN and DA-CNN methods exhibit satisfactory performance, while DA-CNN outperforms MSALSTM-CNN in sensitivity and F1 score for all experiments. The highest F1-score of 98.97% is observed in row 2 by the DA-CNN method when the duration is 20 and the anomaly magnitude is linespace(0,4). The lowest F1-score is observed in row 3, with 94.2% achieved by the CNN-KF method when the anomaly magnitude is the smallest. Furthermore, compared with the MSALSTM-CNN and CNN-KF, the DA-CNN demonstrates consistent performance throughout the experiments, with minor fluctuations. In [46], Wyk et al. stated that detecting gradual drift can be one of the most challenging tasks of anomaly detection. In the hardest situation, namely the magnitude of linespace(0,2) with a duration of 20, DA-CNN can still improve the sensitivity and F1 score by 2.2% and 1.12% respectively.

Table 3.5.: Drift

		CNN-KF(%) [46]				MSALSTM-CNN(%) [52]				DA-CNN(%)				
Anomaly Magnitude	Duration, d	Acc	Sens	Prec	F1	Acc	Sens	Prec	F1	Acc	Sens	Prec	F1	Spec
Linespace(0,4)	10	94.7	93.1	99.2	96.1	96.01	95.85	99.14	97.46	98.59	**96.99**	99.61	**98.29**	99.73
Linespace(0,4)	20	96.0	95.6	99.3	97.4	96.21	96.03	99.27	97.62	99.11	**98.67**	99.27	**98.97**	99.27
Linespace(0,2)	10	93.0	89.6	99.3	94.2	94.36	93.09	99.07	95.58	98.39	**97.32**	99.47	**98.38**	99.26
Linespace(0,2)	20	93.7	93.2	98.7	95.9	94.09	92.78	99.52	96.03	97.34	**94.98**	99.43	**97.15**	99.89

3.4.4 Mixed Anomaly Detection

In the experiment, the performance of DA-CNN was also examined for the occurrences of all types of anomalies. The performance evaluation of different algorithms with respect to different sensors is given in Table 3.6. As this type of experiment is not provided in [52], only [46] and the proposed method are shown in this table. Based on the results, DA-CNN outperforms the other algorithms on all three sensors, achieving the highest scores on almost all performance metrics, especially for sensitivity and F1 scores. This suggests that DA-CNN provides a more accurate and robust approach to anomaly detection than the other algorithms evaluated.

Table 3.6.: Mix

	KF(%) [46]				CNN(%) [46]				CNN-KF(%) [46]				DA-CNN(%)				
Sensor	Acc	Sens	Prec	F1	Acc	Sens	Prec	F1	Acc	Sens	Prec	F1	Acc	Sens	Prec	F1	Spec
1	97.8	90.5	95.9	93.1	98.0	93.3	99.5	96.3	98.1	94.2	99.4	96.7	99.32	**98.42**	98.17	**98.30**	99.35
2	98.0	90.6	96.0	93.2	97.0	90.4	98.7	94.4	97.6	92.9	98.4	95.5	98.70	**96.12**	98.14	**97.12**	99.25
3	96.1	85.1	89.0	86.9	94.7	83.3	97.6	89.8	95.7	87.3	97.3	92.0	95.57	**89.36**	97.11	**93.07**	98.60

3.4.5 ADDITIONAL EVALUATIONS

This section provides additional evaluations, namely the ROC, curve, AUC, and the FPR95 metric (the false positive rate at the true positive rate equal to 95%), which can be critical in anomaly detection tasks. The ROC and FPR95 can be found in Figure 3.11 - 3.15

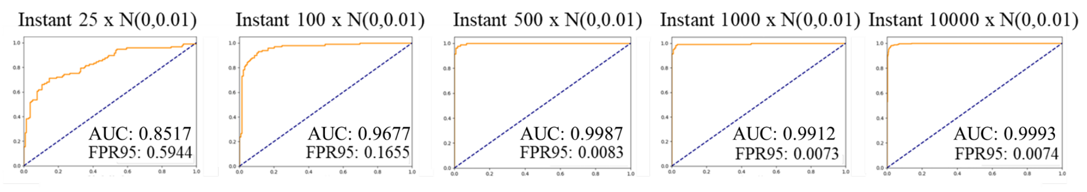


Figure 3.11.: ROC Curves, AUC, and FPR95 of Instant Anomalies

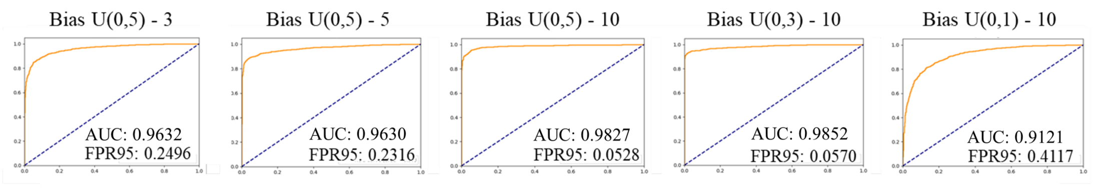


Figure 3.12.: ROC Curves, AUC, and FPR95 of Bias Anomalies

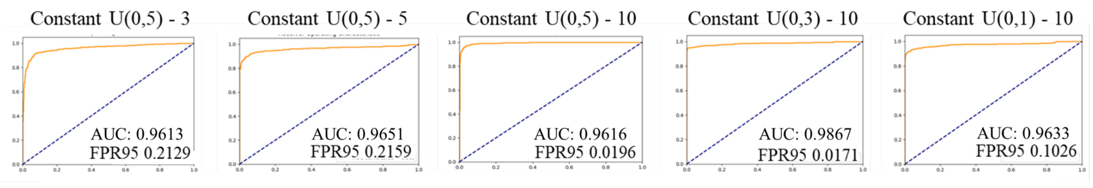


Figure 3.13.: ROC Curves, AUC, and FPR95 of Constant Anomalies

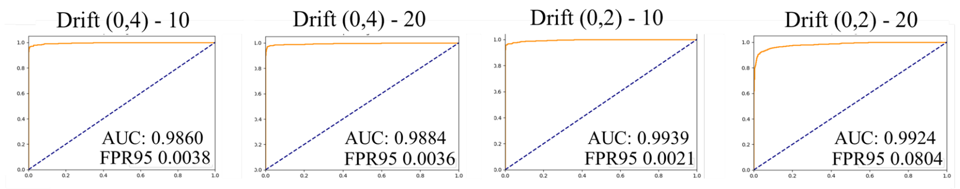


Figure 3.14.: ROC Curves, AUC, and FPR95 of Drift Anomalies

3.5 Ablation Experiment

In order to verify the positive effect of one of the core modules of the model proposed in this paper, namely the DAM module, on performance, this section

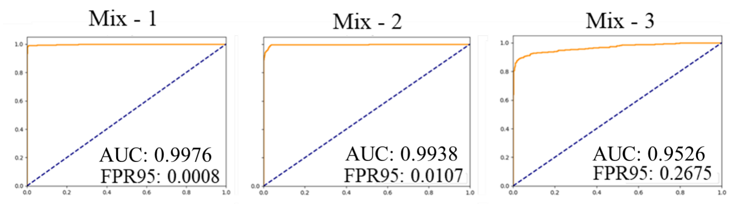


Figure 3.15.: ROC Curves, AUC, and FPR95 of Mixed Anomalies

presents ablation experiments.

Two distinct experiments were conducted to validate the efficacy of integrating sensor-wise and time-wise attention mechanisms. In the initial experiment, we exclusively employed sensor-wise attention across both channels, focusing on the individual characteristics of each sensor. Conversely, the second experiment solely utilized time-wise attention in both channels, concentrating on temporal dynamics. This ablation study was designed to rigorously assess the impact and effectiveness of combining sensor-wise and time-wise attention, providing a comprehensive understanding of their synergistic potential in the proposed model.

Table 3.7.: Performance difference among DSA-CNN, its sensor-wise attention-only version, and its time-wise attention-only version

Tasks	DSA-CNN				Sensor-wise attention only				Time-wise attention only			
	Acc	Sen	Prec	F1	Acc	Sen	Prec	F1	Acc	Sen	Prec	F1
Instant 25 x N(0,0.01)	85.00	**71.28**	88.16	**78.82**	82.50	70.70	84.33	76.92	63.75	44.90	57.14	50.29
Bias U(0,1)-10	88.69	**86.18**	95.02	**90.38**	84.13	83.13	90.35	86.59	71.33	76.83	76.92	76.88
Constant U(0,5)-3	95.73	**92.72**	99.04	**95.77**	95.65	92.13	99.48	95.67	92.23	88.31	96.37	92.17
Drift Linespace(0,2)-20	97.34	**94.98**	99.43	**97.15**	96.56	93.48	99.30	96.30	88.07	84.95	89.71	87.27
Mix - 3	95.57	89.36	97.11	**93.07**	94.58	**89.89**	93.49	91.70	92.23	85.58	90.78	88.10

To maintain comparability, all training parameters, data, the rest of the model architecture, as well as the environment were kept the same. The ablation experiments were only conducted on the most challenging tasks on each of the anomaly types (The tasks that received the lowest F1 scores). The results can be found in Table 3.7. It can be found that neither the use of sensor-wise attention alone nor time-wise attention can achieve the performance of using them simultaneously, which validates the effectiveness of the proposed DAM in anomaly detection.

3.6 Discussion

It is clear that the DA-CNN proposed in this paper offers significant advantages in terms of sensitivity and F1 score, especially in minor sensor anomalies. Higher sensitivity means fewer undetected anomalies, which can be crucial for safety-critical application scenarios, such as the CAV velocity and acceleration data used in this work, which can be the important decision bases for autonomous driving systems [217].

As an important sensor anomaly detection algorithm, Kalman filtering has the advantages of being easy to implement and computationally efficient, making it widely used. It can predict the near future or current system state by tracing back to a few past system state data, so by comparing the difference between the estimated data and the measurements and setting a threshold, it can determine if the sensor reading now is in an abnormal state. However, Kalman filtering has three disadvantages in sensor anomaly detection. Firstly, its performance relies on artificial noise estimations, for example in [46] where the sensor noise was assumed to be Gaussian white noise, which is sometimes not representative of the real noise in the actual system. Secondly, plain Kalman filtering often requires the assumption that the system is linear and time-invariant [218]. This assumption can be satisfied when the time window taken for the analysis of CAV velocities is small enough, but the time window for the analysis of time-series data often cannot be too small to ensure that sufficient information can be incorporated, so the non-linearity introduced by the large time window becomes a source of error in the Kalman filter estimations. Finally, defining thresholds for normal intervals can be often difficult, as the intensity of noise in a real system can be likely to vary according to environmental conditions. These might be the reason why the Kalman filter-based approach in [46] achieves relatively poor performance despite the fact that the use of CNNs to extract a representation of the system state can overcome to some extent the problems of noise and non-linearity within the time window.

LSTM is an RNN algorithm that excels in time series data modelling. It largely alleviates the forgetting problem of traditional RNN by controlling the extent to which long-term and short-term memory can be involved in downstream computation through the gate structure. It, therefore, plays an important role in the detection of anomalies in time-series sensor data. Due to the powerful feature extraction and non-linear modelling capabilities of deep learning methods, LSTM-based methods do not require an accurate estimation of noise, unlike Kalman filtering, and do not rely on the assumption of linear time-invariant systems. This can be the possible reason why the LSTM-based method used in [52] outperforms the Kalman filter-based method in [46]. However, since LSTM can be still recursive in nature, the features of the initial time step can be still sometimes difficult to learn, leading to potential improvements in the performance of LSTM-based methods in detecting small anomalies. As for the self-attention mechanism used in this work, since it has no recursive structure, information at all time steps can be treated equally, avoiding the problem of forgetting. Thus DA-CNN showed a significant improvement in experiments dealing with small anomalies. Furthermore, in contrast to the use of the attention mechanism in [52] for all time steps in the LSTM, this work proposes a dual-channel attention mechanism, which can model not only the dependencies among different time steps but also the dependencies between different data sources (sensors), so that the feature maps obtained by DA-CNN can be more integrated and comprehensive, leading to the significant performance improvement of DA-CNN shown in the previous section.

However, due to the current supervised learning-based training approach, the performance of the proposed model in dealing with unknown complex anomalous patterns has not been evaluated. A common deep learning-based approach in the literature for handling unknown pattern anomalies can be the unsupervised learning-based autoencoder architecture, which means reconstructing the original normal signal using a deep learning model, then using one of the latent spaces as a feature representation, and then the distance between the latent space of the anomalous data and the latent space of the normal data can be calculated as a criterion for the occurrence of anomalies [149] [158]. The performance of DA-CNN in this type of methodology can be an important future direction to be evaluated.

3.7 Summary

Anomaly detection in multi-sensor data streams can be an important foundation for industrial process monitoring tasks, and it can be also an essential component in many safety-critical systems, such as CAV. In this paper, a method, DA-CNN, was proposed based on the self-attention mechanism and CNN to detect the anomalous behaviour of CAV velocity and acceleration sensors. This work proposed a dual-channel self-attention structure to enhance the sensitivity of the model with respect to small anomalies, and this structure can integrate the learning of spatiotemporal features into the training process of deep learning. The proposed method was tested on a publicly available CAV dataset, and the sensor anomaly was simulated based on the 4 most damaging anomalies for CAVs [209] [210]. Based on the experimental results, the proposed DA-CNN achieved 2.57%, 2.07%, 1.78%, and 3.83% performance gains in sensitivity for drift, constant, bias, and instant anomalies, and 1.53%, 1.32%, 0.94%, and 2.21% gains in F1 score, compared with the SOTA performance model in the literature proposed in [52]. The experimental results also show that DA-CNN has a clear advantage in handling small anomalies.

The main contributions can be summarised as follows:

- A novel approach, DA-CNN was proposed based on the self-attention mechanism for sensor anomaly detection in multivariate time series.

This model is able to detect whether the multi-stream time series sensor readings of a time window contain anomaly values.

- A novel self-attention-based deep neural network block, Dual-channel Attention Mechanism (DAM) was proposed. In the proposed block, the spatiotemporal features which can be vital for anomaly detection [219] can be extracted and integrated progressively and automatically by sensor-wise and time-wise attention channels during the learning process, eliminating the artificial signal processing stage for extracting the spatiotemporal features before designing an anomaly detection algorithm.

- This work has accomplished a performance benchmark that can be state-of-the-art among the works within the field of CAVs. The experiment results show that the proposed DA-CNN has a clear advantage in handling small anomalies, and improves the sensitivity significantly among all the experimental conditions being evaluated. This means the proposed method has fewer undetected anomalies which can be harmful in real systems compared with the other methods in the literature.

This work demonstrates the excellent feature extraction capabilities of Transformer-like network structures for sensor anomaly detection. However, due to the current supervised learning-based classification training approach, feature learning performance in the presence of unknown anomaly patterns is an important future research direction.

4

Deep Transfer Learning with Self-attention for Industry Sensor Fusion Tasks

4.1 Introduction

In industrial sensor fusion tasks, when dealing with a large and complex input space, such as a large number of sensors with different sampling rates, more complex and deeper networks are often necessary, as deeper networks have stronger feature extraction capabilities [220]. This leads to a significant increase in the depth and width of the model, resulting in high demands on the amount of training data. For industrial scenarios, collecting a large amount of training data means a significant increase in time and expense, which is sometimes not even possible. Alternatively, artificial feature extraction, such as time-domain statistical feature extraction [29], frequency-domain feature extraction [221], plus a shallow neural network [141] [222] can be used to reduce the need for training data. For such a method, the performance is heavily dependent on the quality of manually extracted features, and feature engineering often requires extensive experimentation and an understanding of industrial processes, which can be time-consuming and difficult. Therefore, a method that requires a relatively small amount of data and does not rely on feature engineering is preferred. The hard choices mentioned above can be described by Fig. 4.1

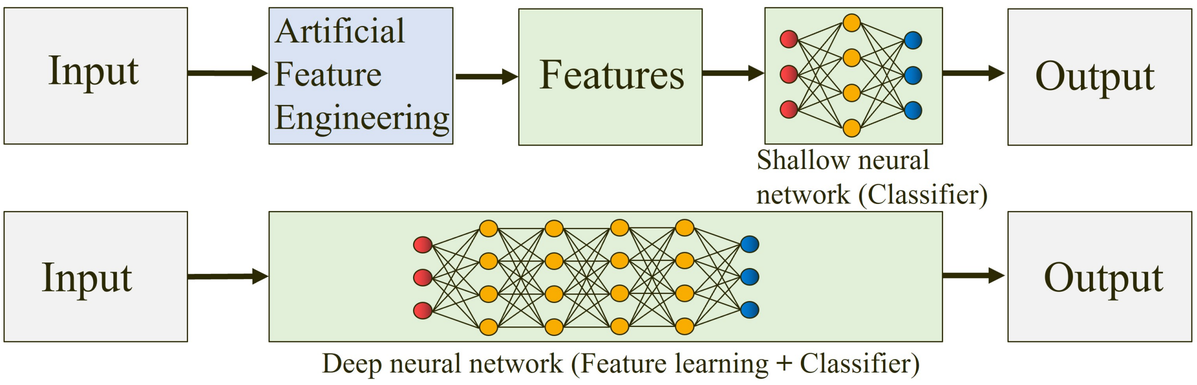

Input
Artificial Feature Engineering
Features
Shallow neural network (Classifier)
Output
Input
Deep neural network (Feature learning + Classifier)
Output

Figure 4.1.: Comparison between shallow neural network plus artificial feature engineering and deep neural network which can extract features automatically

In addition, the lack of interpretability as discussed in the previous chapter is another disadvantage of Deep Learning in industrial applications. The mechanism by which input data is mapped to model output of Deep Learning is difficult to obtain. Hence, while using Deep Learning, it is difficult to know which sensor has the decisive influence in a multi-sensor system.

In this work, a novel transfer learning methodology for sensor fusion that transfers a deep model from the natural language processing (NLP) domain, coined 'Transformer' [162] to industrial applications was proposed. NLP is a data-rich domain, deep models in this domain are relatively easy to be adequately trained. Lu et al. found that the feature extraction ability of such a complex deep model has the potential to be transferred to other modalities since natural language is a modality with a huge amount of data and features [192]. They found that a pre-trained Transformer with fine-tuning offers great performance on numerical operations, image classification, and protein folding prediction tasks. Therefore, the transfer of feature representations identified from NLP to sensor fusion problems could enable the application scenarios, with high input dimension but insufficient data, to train very deep neural networks from raw to use deep networks. Thus, the proposed method is expected to make use of the strong learning ability of the deep neural network and the advantage of the training ease of the shallow neural network to improve the sensor fusion task for manufacturing. This can be achieved by freezing all the attention and feed-forward layers and training the input embedding, layer norm, and final full-connected output layers only. It means that the computation of feature extraction which is inherent in natural language will be transferred to sensor fusion tasks.

In addition, the self-attention mechanism used by Transformer can provide insights into the final decision made by the deep network and obtain the weight information, namely the attention map, of sensor data. This can be used for the identification of critical sensors and hence makes up for the lack of interpretability when using the deep learning model in industrial sensor fusion tasks.

4.2 Problem Statement

Monitoring complex industrial processes requires utilising various sensing modalities to obtain diverse process data, which presents a significant challenge on how to effectively fuse the multi-sensor information to achieve comprehensive monitoring. Deep learning has recently emerged as a promising approach due to its capability of automatic feature learning. However, conventional shallow neural networks often show insufficient learning capacity and require artificial feature engineering, and the deeper networks that can overcome these limitations typically require large volumes of data, which can be impractical in industrial settings. Hence, the problem being solved in this research is how to enable the use of deep networks for industrial sensor fusion. As a result, industrial process monitoring can benefit from the learning capability of deep networks while avoiding excessive demands on the amount of training data.

In addition, as interpretability is preferred by industrial applications, another problem being investigated is how to obtain the decision basis for a deep model that can help identify the key sensors in a large number of sensors.

4.3 Methodology and Architecture

4.3.1 Similarities between Natural Language Processing and Sensor Fusion

Typically, when deep learning is used to deal with NLP problems, we often map the words of a certain language into an embedding space which could be regarded as the unique identification information of each word. Transformer can automatically learn not only the internal features of each word with the help of multiple attention heads but also the features of the relationships among the input word sequence on a single head of attention as shown in Fig. 4.2 [162]. This learning mechanism is preferred by sensor fusion tasks, where we want to establish a unified feature representation that includes both inter-sensor and intra-sensor information, where the inter-sensor information means the information contained in the interrelation among different sensors and the intra-sensor information means the information contained in a single sensor.

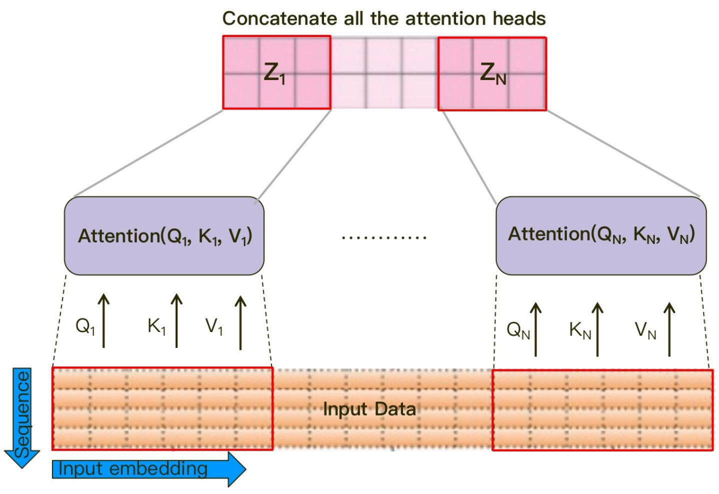

Concatenate all the attention heads
Z_1
Z_N
Attention(Q_1, K_1, V_1)
....
Attention(Q_N, K_N, V_N)
Q_1
K_1
V_1
Q_N
K_N
V_N
Input Data
Sequence
Input embedding

Figure 4.2.: Multi-heads attention

In NLP tasks, the model needs to recognize the meaning of a single word, and it also needs to infer the information contained in a sentence or a paragraph. Similarly, in a multi-sensor system, the information of each sensor could be compared to a single word in NLP, and the information of all sensors can be regarded as a sentence or paragraph. If the multiple sensor data from different modalities can be mapped to a mapping space with a unified format, the extraction of multi-sensor information could also be analogized to the understanding of paragraph semantics as shown in Fig. 4.3. This gives the possibility to use NLP models

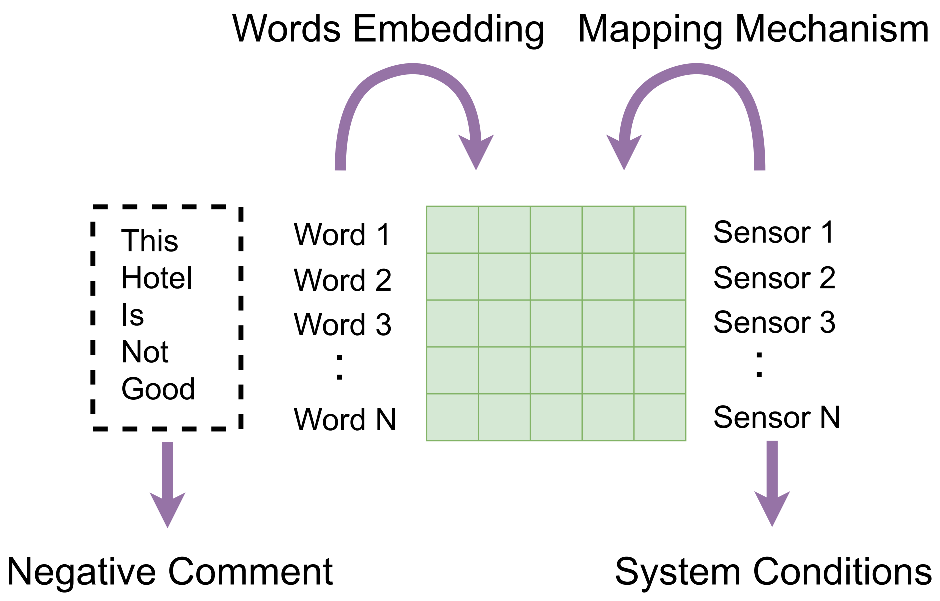


Figure 4.3.: Sensor Data Mapping

on sensor fusion tasks. Technically speaking, the self-attention mechanism may excel in handling multi-sensor data due to its inherent capacity to model interactions and dependencies between any two data patches, regardless of their position in the input sequence. This flexibility makes it particularly suitable for multi-sensor environments, where data from different sensors may have complex, non-linear relationships that vary over time. By computing attention scores that reflect the importance of each sensor's data point in the context of others, self-attention allows the model to dynamically weigh and integrate information across the entire set of inputs. This means that it can effectively capture the temporal and spatial correlations between diverse sensor signals, enabling it to learn from the rich, multidimensional data these sensors provide.

However, finding an optimised embedding space is not an easy task. In this paper, we use a linear layer to learn the mapping rules automatically instead of designing mapping rules artificially.

4.3.2 Model Architecture and Computation Process

The architecture of the proposed model can be described as shown in Fig. 4.4. The overall computational flow is shown in Algorithm 3. Each part of this model will be explained in the following items.

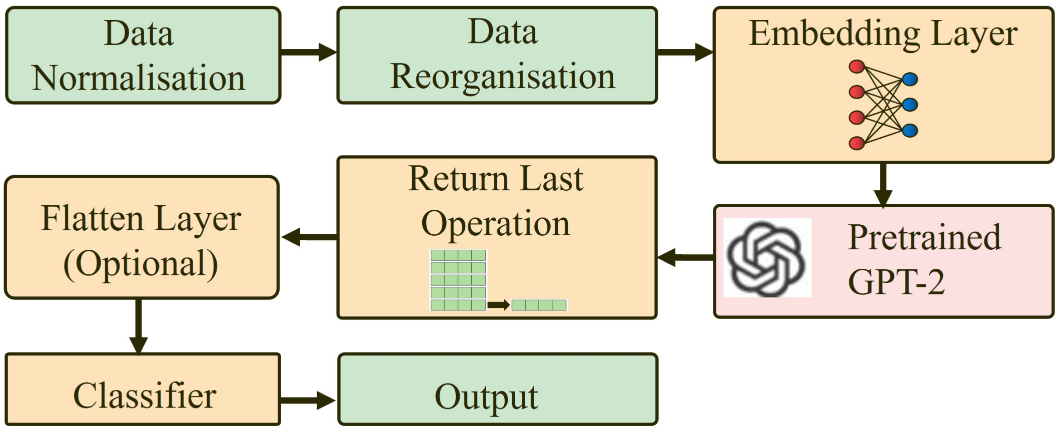


Figure 4.4.: Proposed Model Architecture

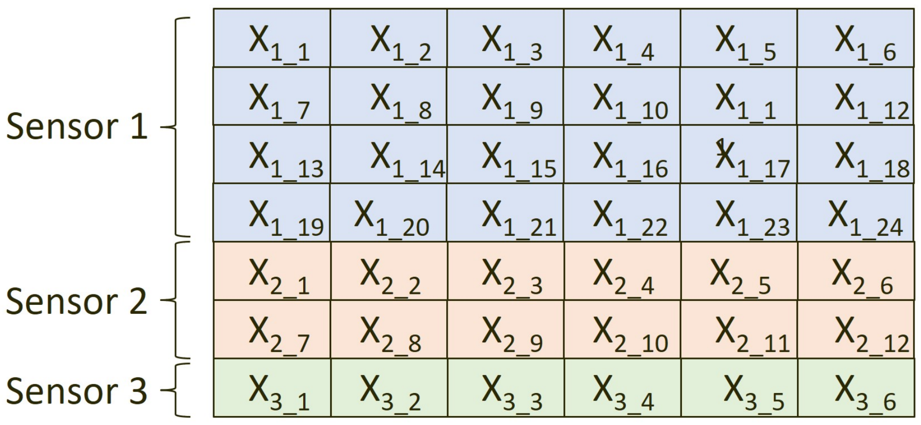
Data Normalisation

As the value range of different sensors will vary by orders of magnitude, in order to facilitate training, the data from each sensor will be normalized using the following formula:

$$X_{norm} = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{4.1}$$

Data Reorganisation

The input of the Transformer is a two-dimensional matrix. Each row is a word embedding vector and the two-dimensional input matrix is formed by stacking the embedding vectors of each word. In this research, word embedding is replaced with sensor data, hence different rows of input are different sensor data. Since different sensors usually have different sampling rates corresponding to the nature of the object being measured, in order to maintain the same embedding vector length, data with a high sampling rate need to occupy multiple embedding vectors, as shown in Fig. 4.5. To keep the length of the embedding



Sensor 1
Sensor 2
Sensor 3
X1_1 X1_2 X1_3 X1_4 X1_5 X1_6
X1_7 X1_8 X1_9 X1_10 X1_1 X1_12
X1_13 X1_14 X1_15 X1_16 X1_17 X1_18
X1_19 X1_20 X1_21 X1_22 X1_23 X1_24
X2_1 X2_2 X2_3 X2_4 X2_5 X2_6
X2_7 X2_8 X2_9 X2_10 X2_11 X2_12
X3_1 X3_2 X3_3 X3_4 X3_5 X3_6

Figure 4.5.: Example of data reorganisation using 1-second sensor data as an input data point (Sampling rates: sensor 1: 24Hz, sensor 2: 12Hz, sensor 3: 6Hz)

vector the same, the sampling rates of the sensors need to be in multiples, and to achieve that some sensor values may be discarded or data padding may be used. The reorganised input space can be described by:

$$\mathcal{X} \in \mathbb{R}^{d \times L} \tag{4.2}$$

where d is the sequence length, and L is the dimension of each element in the input sequence.

Embedding Layer

The mapping of sensor data to the embedding space is done by a linear layer with a dropout rate of 0.1 and orthogonally initialised weights. The mapping rules will be automatically learned as the parameters are updated during the training process. At the same time, this layer also has the functions of preliminary feature extraction and dimension adjustment for the upcoming layers. The embedding layer can be described by the following equation:

$$\text{Embedding}(X) = ReLU\left(XW_{embedding} + b_{embedding}\right) \tag{4.3}$$

where $W_{embedding}, b_{embedding}$ are the weight matrix and bias matrix. ReLU(*) is the activation function. $X \in \mathbb{R}^{d\times L}$, $W_{embedding} \in \mathbb{R}^{L\times L_{embedding}}$, and $b_{embedding} \in \mathbb{R}^{d\times L_{embedding}}$. $L_{embedding}$ is the embedding dimension which is used for matching the downstream layers.

Generative Pre-trained Transformer 2 (GPT-2)

In this research, a Transformer is expected to be the feature extraction engine, therefore only its decoder part is required. OpenAI provides a pre-trained decoder of Transformer called GPT-2 with 1.5 billion parameters [223]. It is trained from a 40GB non-task-specific training dataset which was crawled from 8 million web pages. In sensor fusion tasks for industrial scenarios, it is difficult to train a deep model with a large amount of data, however, this is not very difficult to achieve in NLP scenarios. The architecture of GPT-2 is shown in Fig. 4.6 [224]. It is

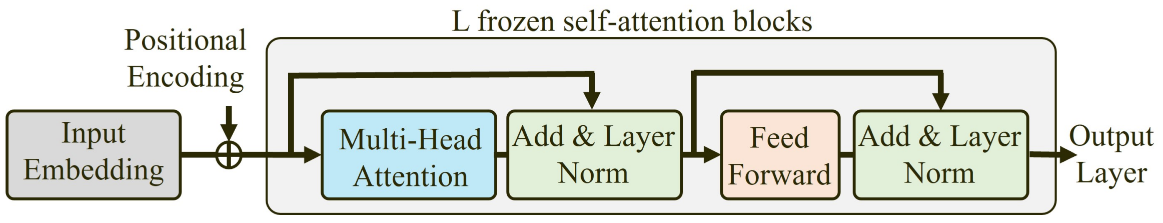


Figure 4.6.: GPT-2 Architecture

composed of 12 attention layers and each attention layer consists of a 12 heads attention which is used to generate an attention map, two shortcut connections with layer normalisation, and a fully connected feed-forward network. The input dimension of GPT-2 is (N, 768), where N is the sequence number, namely the number of rows of the input matrix. For example in Fig. 4.3, there are 5 rows, so the sequence number is 5. The second number 768 is the size of each row. Similar to the embedding layer described earlier, each row in this sequence will share the same feed-forward network.

In this proposed method, similar to [192], 99.9% of the GPT2 model parameters which were trained from the natural language dataset were frozen. Only the normalisation layer, linear input and output layer were fine-tuned with a learning rate of 0.001 as this learning rate can balance the training speed and stability based on practice, and Adam was used as its optimiser.

Return Last Operation

Because of the mask mechanism of GPT-2, only the output result at the last position of the output sequence contains all the input sequence information, which could be compared to the last hidden state in RNN. Therefore, since our work is not a Seq2Seq task, only the last position in the output sequence will be used as the input of the classifier.

Flatten Layer (optional)

Depending on the sampling rate of the sensor, if the data of the last sensor occupies multiple rows in the sequence, we need to return all the rows of this sensor, because the expected output result is the inter-sensor information. Thus, in this case, a flattened layer is needed.

Classification Layer

After the above calculations, we have obtained a unified feature representation of the fused sensor data. The last layer of this model is a feed-forward neural network for classification. The last position in the output sequence of the GPT2 attention block is the input of the classifier, and the number of output neurons is the number of classes. This work only used a single-layer feed-forward network, and the cross-entropy loss was used as its loss function.

4.4 Experiments and Results

Based on the description in the last section, the proposed model consists of an embedding layer, GPT-2, and a classifier. As we transferred the parameters of the pre-trained GPT-2, the GPT-2 has to be kept to its original structure. Since the input and output dimensions of GPT-2 are (N, 768), the output dimension of the embedding layer and the input dimension of the classifier were set to (N, 768) to match the dimension of GPT-2. The input dimension of the embedding layer depends on the dimension of the specific dataset and the output dimension of the classifier can be determined by the target categories of specific tasks.

The hyperparameters and model configuration can be found in Table 4.1. The feed-forward layer and multi-head attention layer were frozen, as these two parts contain all the features learned from natural language. The layer normalisation was set to trainable since the data distribution is different for different datasets. The learning rate was set to 0.001 because it was found in practice that smaller learning rates lead to slow training and larger learning rates lead to increased instability. The optimiser and initialisation remained the same as those used in training the original GPT-2 model [223].

The proposed framework was tested on three different datasets based on condition monitoring of a hydraulic system, the bearing condition of an electric motor, and the gear and bearing working conditions of a gearbox. The experimental results from the three mentioned datasets are provided in this section.

Algorithm 3: Self-Attention-Based Deep Transfer Learning Method For Sensor Fusion

```
Input: Multi-sensor data samples and the corresponding labels
Step 1 (Data preprocessing):
Normalising the multi-sensor data:
    $X_{norm} = \frac{x_i - x_{min}}{x_{max} - x_{min}}$
Constructing multi-sensor inputs to $\mathcal{X} \in \mathbb{R}^{d \times L}$ for Embedding.
Step 2 (Sensor data embedding):
Constructing the embedding space:
    $\text{Embedding}(X) = ReLU\left(XW_{embedding} + b_{embedding}\right)$
Step 3 (Transfer learning and feature extraction):
Construction of the feature representation by GPT-2 pre-trained from
language dataset as shown in Fig. 4.6.
Step 4 (Return Last Operation):
Return the last element of the output sequence.
if The last sensor occupies more than 1 row of embedding space then
    Do flatten operation and return this vector
else if then
    Return the last row of the output sequence
Step 5 (Fault classification):
Constructing the fault classifier, which is a single fully connected layer:
    $\text{Classifier}(X) = XW_{classifier} + b_{classifier}$
Training the model with cross-entropy loss.
Output: The target categories.
```

Table 4.1.: Model parameters and training details

	Experiment	Condition monitoring of a hydraulic system	Bearing condition	Gearbox condition
Pretrained GPT2 model	Return last operation	True	True	True
	Position embedding	None	None	None
	Layer normalisation	Trainable	Trainable	Trainable
	Multi-head attention	Frozen	Frozen	Frozen
	Feed forward layer	Frozen	Frozen	Frozen
	Flatten layer	False	True	False
Training	Parameter initialisation	Orthogonal initialisation (Gain = 1.41)	Orthogonal initialisation (Gain = 1.41)	Orthogonal initialisation (Gain = 1.41)
	Learning rate	0.001	0.001	0.001
	Optimiser	Adam	Adam	Adam
	Batch size	8	64	16
	Loss function	Cross entropy	Cross entropy	Cross entropy

4.4.1 Experiment 1: Condition monitoring of a hydraulic system

Task description

This classification task requires determining the operating conditions of a complex hydraulic system based on 17 sensors with different sampling rates (7 x 100Hz sensors, 2 x 10Hz sensors, 8 x 1Hz sensors). The dataset was created by Helwig et al. [23] on a test rig that was able to simulate a reversible degradation of system performance. This hydraulic system was composed of a primary working circuit and a secondary cooling circuit. Different loads were cyclically applied to a pre-defined work cycle. The data was recorded in every one-minute snapshot, therefore the total number of attributes for one data snapshot will be $8(sensors) \times 60(s) \times 1(Hz) + 2(sensors) \times 60(s) \times 10(Hz) + 7(sensors) \times 60(s) \times 100(Hz) = 43680$. This experiment was repeated 2204 times, hence the dataset had 2204 snapshots included.

This experiment has a total of five tasks to determine the operating conditions of the five different parts of the system as shown in Table 4.2. This system may have multiple faults at the same time.

Table 4.2.: Experiment 1-Operating conditions

Task	System conditions
1 Cooler condition	(1) Close to failure (2) Reduced efficiency (3) Full efficiency
2 Valve condition	(1) Optimal switching behaviour (2) Small lag (3) Sever lag (4) Close to total failure
3 Internal pump	(1) No leakage (2) Weak leakage (3) Sever leakage
4 Hydraulic accumulator	(1) Optimal pressure (2) Slightly reduced pressure (3) Sever reduced pressure (4) Close to failure
5 Stable flag	(1) Condition were stable (2) Non-static conditions

Data organisation

The sensors have three different sampling rates, 1Hz, 10Hz, and 100Hz respectively. The one-minute snapshot data of the 1Hz sensor is used as the length of the embedding vector (60 columns), hence the 10 Hz and the 100 Hz sensors occupy 10 and 100 embedding vectors respectively. Therefore, one snapshot of input data from these 17 sensors, a total of 43680 attributes, will be reshaped as a matrix with 728 rows and 60 columns.

Table 4.3.: Experiment 1-Accuracy Comparison

Research group	Method	Cooler	Valve	Pump	Accumulator	Stable flag	Mean
Helwig et al. [23] (2015)	LDA	100%	100%	98.0%	90.4%	N/A	97.1%
Berghout et al. [225] (2021)	Auto-NAHL	100%	100%	100%	96.4%	N/A	99.1%
Wu et al. [226] (2020)	EGMSVMs	100%	100%	100%	76.5%	N/A	94.1%
Gupta et al. [227] (2021)	SECM	N/A	N/A	N/A	N/A	N/A	92.3%
Lei et al. [228] (2019)	PCA+XGBoost	N/A	96.58%	N/A	N/A	N/A	N/A
Huang et al. [141] (2021)	Deep CNN	100%	100%	99.0%	99.4%	N/A	99.6%
Prakash et al. [130] (2020)	ANN+XGBoost	99.54%	N/A	N/A	N/A	N/A	N/A
Our results	**Experimental method**	**100%**	**100%**	**98.2%**	**91.4% (96.4%)**	**94.4%**	**98.7%**

Results of prediction accuracy

The experimental results of model prediction accuracy and comparison with other research are shown in Table 4.3, and the stable flag was not included in the calculation of mean accuracy.

In this table, each percentage means accuracy which can be described by the following equation:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \tag{4.4}$$

The reason for using accuracy to assess model performance is that there is no order of magnitude difference in the number of samples in each category in this dataset. The accuracy in brackets in this table means the accuracy after sensor selection has been applied, and this will be explained in detail in the next section. Based on Table 4.3, it can be found that our proposed method achieves an accuracy of 98.7%, ranking in the top three of published work in recent years based on this dataset.

Interpretability: Key sensor identification based on attention mechanism

In a multi-sensor system, a large number of sensors can be employed, however, not all of them will have an impact on the final decision. When we are dealing with a complex industrial system, it can be difficult to identify critical sensors. Using too many redundant sensors may increase the complexity of the system, waste communication bandwidth and increase the computational burden. Hence, deep learning methods that are interpretable to some extent are preferred by the industry.

The concept of interpretability of deep learning models can be divided into the following two aspects:

- Models are transparent to humans. The corresponding model parameters and the model decisions can be predicted before the model is trained for a specific task [229].
- Decision interpretability. After a model makes a decision, humans can understand the reasons for that decision [230].

This subsection shows the decision interpretability of the proposed method.

In general, AMs are used to control the information flow of deep networks. In the process of backpropagation, the part of the data that has less impact on the results will be masked gradually, and only the information that is decisive for the final decision will be retained. The importance of input sensor data is reflected by the attention weights [231]. Hence, compared with the conventional black-box deep learning models that give no information on which sensors it relies, the key sensors can be identified by visualising the attention weights. As shown in Table 4.3, the classification accuracy of the fourth task is only 91.4%. This indicates that there may still be room for improvement. Therefore, we chose this task to show the key sensor identification capability of AM.

As the data from 17 sensors are reorganised into 728 vectors of 60 columns, the original attention heat map is a matrix of 728 rows and 728 columns. The x-axis represents the input sequences and the y-axis represents the output sequences, for example, the first row is the attention scores of the first element in the output sequence corresponding to all inputs. The dark blue colour means this part of the data is masked and no information can flow to the next layers. As mentioned in the previous section, the input sequence has 728 vectors of size 60. Since the sensors with different sampling rates occupy different numbers of input vectors, attention scores for input vectors from the same sensor need to be added together. The processed attention heat map is shown in Fig. 4.7. As mentioned above, the last row of the processed attention heat map, in this case, the 17th row, is the output attention score based on all of these 17 sensors.

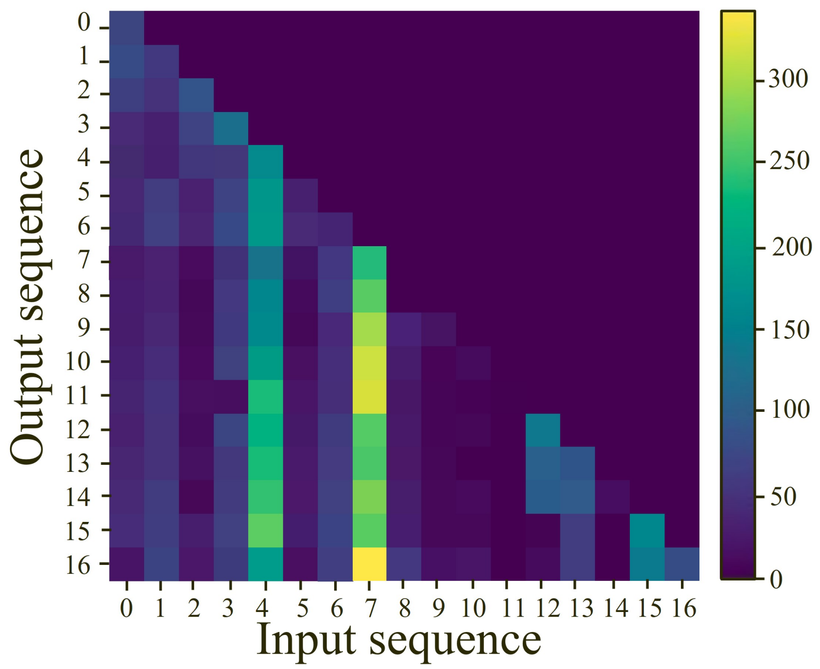


Figure 4.7.: Attention heat map for accumulator task of experiment 1

The attention weights shown in the 17th row can be visualised as Fig. 4.8. Based on this information, the sensors are divided into two groups, the highest attention weights group and the lowest attention weights group. 8 of the sensors are contained in the highest attention weights group, whereas another 8 sensors

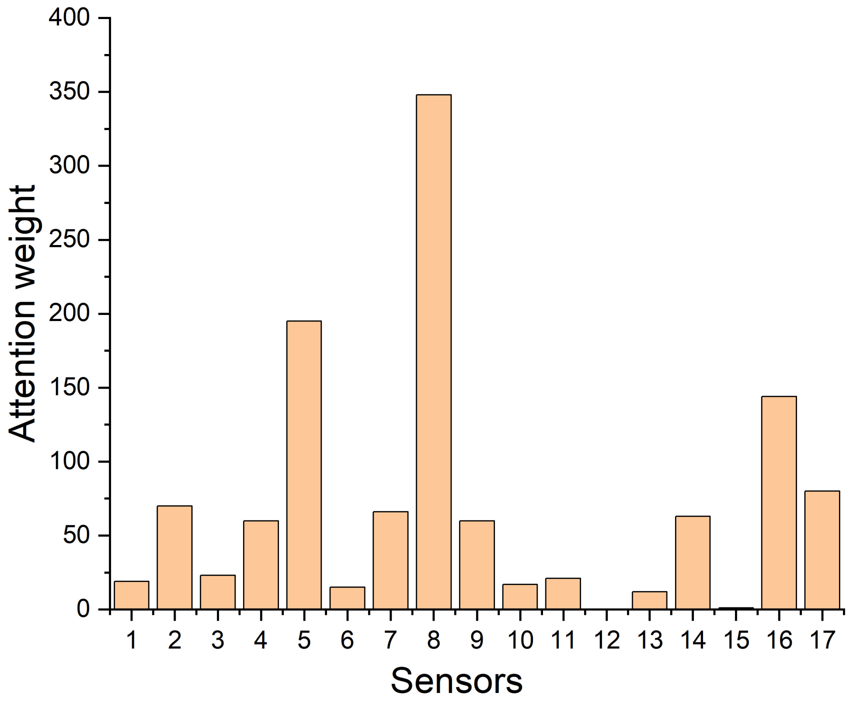
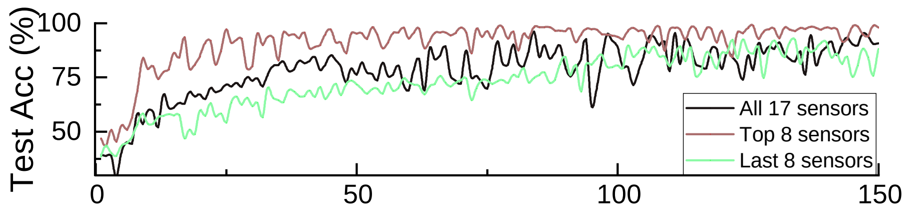


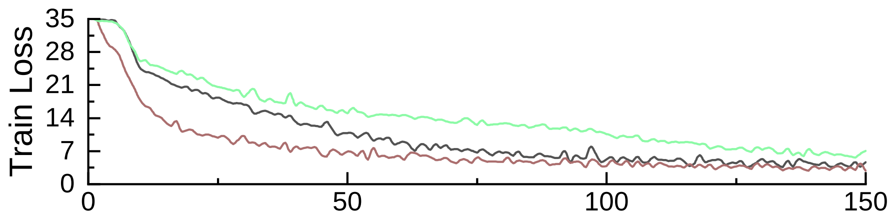
Figure 4.8.: 17 sensors attention weights

are contained in the lowest attention weights group. Then, instead of using all these 17 sensors, the model is trained by these two groups separately. The training history is shown in Fig. 4.9 and the accuracy of using the top 8 attention sensors and the last 8 attention sensors are 96.4% and 83.0%.

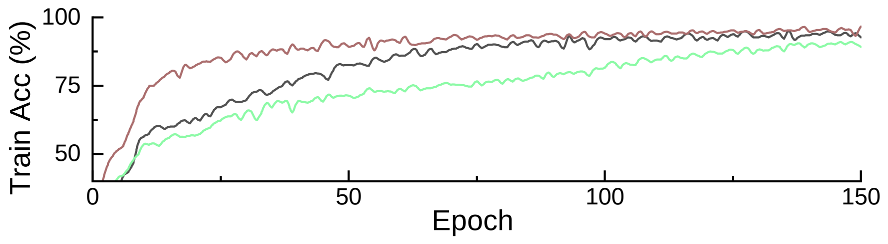
It can be found that using the highest attention weights group only to train the model can obtain better results than using all the sensors, which means faster convergence, higher accuracy and less fluctuation. The reason for this phenomenon may be that selecting only the more important sensors can effectively reduce the dimension of the input space and thus reduce the difficulty of feature learning of the model. This result illustrates the key sensors identified by the proposed method contain sufficient information for decision-making. As the proposed method provides access to the decision basis of the model, it has a higher degree of interpretability than traditional black-box models.



(a)



(b)



(c)

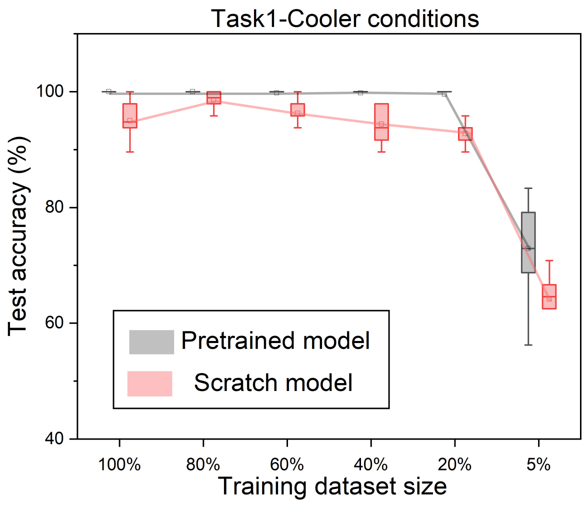
Figure 4.9.: Training history of 2 groups of sensors vs. Using all sensors

Model performance tests under different amounts of training data

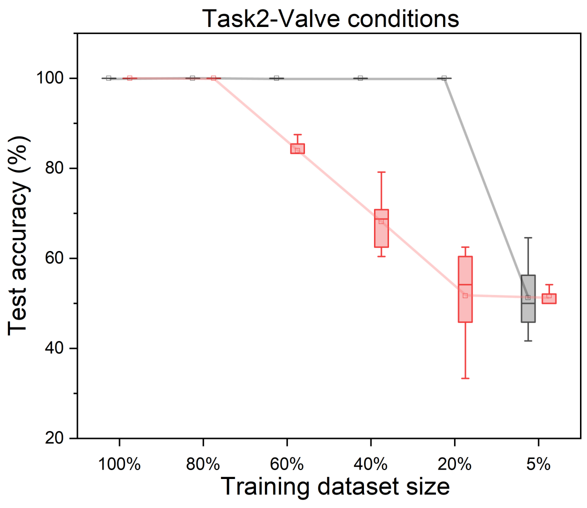
This section compares the performance difference under different amounts of training data between the pre-trained model and scratch model to evaluate whether the parameters trained from natural languages can help to reduce the necessary amount of training data. In this experiment, the training dataset was trimmed to 6 subsets with different sizes, 100%, 80%, 60%, 40%, 20%, and 5% of the original size respectively. The testing dataset for model evaluation was kept the same as in Experiment 1: Condition monitoring of a hydraulic system subsection 3) for all tests.

The test results are shown in the box plot shown in Fig. 4.10. It can be found that regarding the prediction accuracy and its stability, the pre-trained model outperforms the scratch model for all 5 tasks under all different training data amounts. In task 1 as shown in Fig. 4.10 (a), although there is a significant performance drop when the size of training data is reduced from 20% to 5% for both models, the lower accuracy and larger fluctuation of the scratch model can be observed. In task 2 as shown in Fig. 4.10 (b), when the amount of training data is greater than or equal to 80% of its original size, these two models perform almost the same. However, the performance of the scratch model decreases obviously when the train data size shrinks to 60%, while the pre-trained model keeps stable until 20% of training data. As for tasks 3 and 4, the scratch model fails to capture enough information to predict system conditions as shown in Fig. 4.10(c) and (d). In terms of task 5 as shown in Fig. 4.10(e), compared with the fact that the performance of the pre-trained model is still relatively high even at 5% of training data, the scratch model degrades remarkably after shrinking the size of training data to 80%.

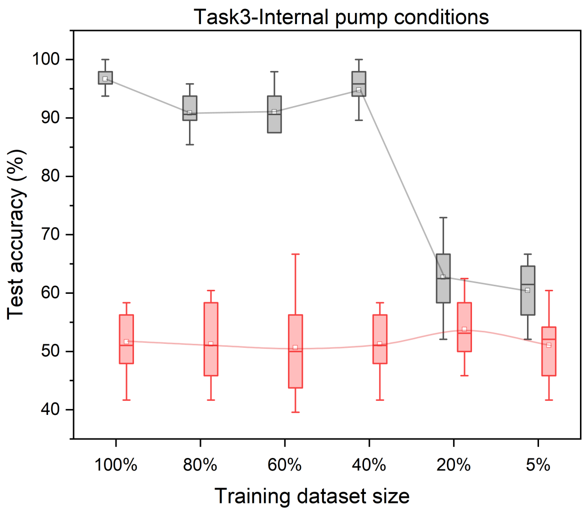
In summary, the model transferred from the natural language domain can effectively reduce the necessary amount of training data when using deep learning in the industrial sensor fusion domain. This means the workload and the time consumed for collecting industrial data can be effectively saved. Current trans-



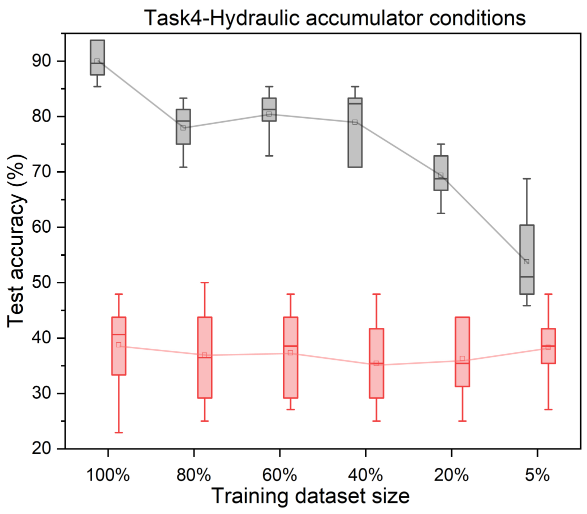
(a) Task1



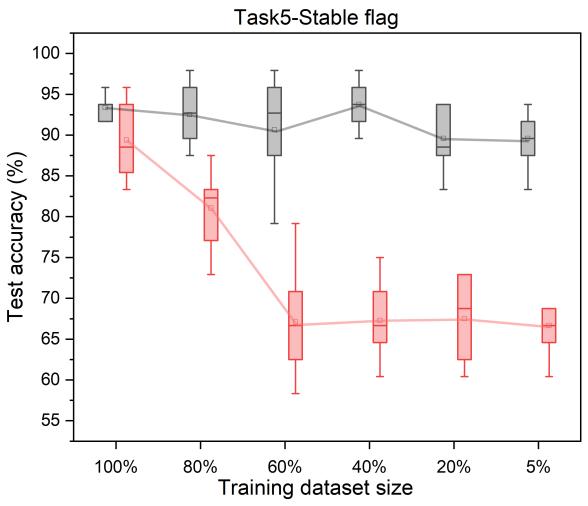
(b) Task2



(c) Task3



(d) Task4



(e) Task5

Figure 4.10.: Classification accuracy under different amount of training data

fer learning solution for industrial sensor fusion requires similar industrial process data to pre-train the deep learning model [232] [172]. Our proposed method proves that similar industrial process data may not be the only option to perform transfer learning for industrial sensor fusion. Hence, the limitations of using deep learning in industrial scenarios can be reduced significantly.

Discussion of results and comparison of performance with the alternative methods

The accuracy comparison of the works published in recent years can be found in Table 4.3, and the comparison of the characteristics of these different methods can be found in Table 4.4. As the method proposed by Gupta et al. was focused on detecting system degradation earlier, and it is different from the condition monitoring focused on in this paper, their work is not included in the comparison. As shown in Table 4.4, manual feature engineering is required for all methods except the methods proposed in [228], [141], and our proposed method. In contrast to other methods, our approach also does not require dimension reduction. This demonstrates the end-to-end nature of our proposed approach.

In [23], Helwig et al. extracted a large number of features in the time and frequency domain for all sensors, and for each extracted feature they analysed its correlation with system faults by Pearson's correlation coefficient and Spearman's rank correlation coefficient. Then, only the highly relevant features are used as the input of their classifier called Linear Discriminant Analysis (LDA). Their accuracies for different tasks shown in Table 4.3 were achieved by different combinations of features obtained by different correlation analyses. This means that when working on a new task, a large number of feature combinations have to be traversed to find the best model. Similar feature engineering can also be found in [225] and [226]. In [130], Prakash et al. used XGBoost technology to select features before feeding data to a shallow neural network. The studies mentioned above all rely heavily on the use of artificial features to represent information from multiple sensors, thus combining the sensors with different sampling rates and reducing the dimension of the input space to ensure that the complexity of the input space does not exceed the capacity limit of the classifier. In contrast, in our proposed method, manual feature extraction is not required. On the one hand, the sensor data will be mapped to a unified embedding space, thus combining sensors with different sampling rates. On the other hand, the deep learning model trained from natural language, as feature extractors, can extract and select the features from the embedding space automatically. Hence, it can significantly reduce the workload and avoid the difficulty of artificial feature engineering.

In terms of the research of Huang et al. in [141], they used multiple independent parallel convolutional neural networks to extract features for each sensor. The output of each of the convolutional neural networks was kept the same, thus, the sensors with different sampling rates can be combined and automatic feature extraction was also achieved. They have achieved excellent accuracy on this dataset. However, such a network structure widens significantly as the number of sensors increases. Cohen et al. pointed out that it is necessary to increase the network depth if the width increases [220]. As the size of the industrial dataset is usually limited, the depth of a network that can be trained is also limited. Therefore, the number of input sensors has to be reduced to prevent the network from being too wide. Huang et al. noted that the large input dimension of their model was unacceptable as this will lead to training failure [141]. Hence, they reduced the input dimension from 43680 to 6000 based on artificial sensor selection. In contrast, our proposed method is relatively more insensitive to high input dimensions. Since the amount of data in natural language is very large, this allows for training deeper models and therefore it can handle higher input dimensions compared to shallow networks. This is a preferred advantage when dealing with large sensor numbers and a lack of a priori knowledge of these sensors. In our experiment, we used the original size of the input (43680) without any artificial dimension reduction.

Moreover, in our proposed method, the model decision basis provided by the self-attention mechanism is also not available in other methods, and this alleviates the black-box nature of deep learning models.

Table 4.4.: Experiment 1-Method comparison

Research group	**Feature engineering**	**Dimension reduction**	**Transfer learning**
Helwig et al. [23]	Yes	Yes	No
Berghout et al. [225]	Yes	Yes	No
Wu et al. [226]	Yes	Yes	No
Lei et al. [228]	No	Yes	No
Huang et al. [141]	No	Yes	No
Prakash et al. [130]	Yes	Yes	No
Our method	No	No	Yes

4.4.2 Experiment 2: Bearing dataset

Task description

This dataset, created by the Case Western Reserve University Bearing Data Center [233], is based on a task that identifies the bearing conditions of an electric motor. Two vibration sensors are mounted on the drive end and the fan end respectively. The faults occurred in three locations, bearing rolling element, inner raceway, and outer raceway. Each location has three different levels of fault severity, small, medium, and large. Thus, these nine different fault categories plus the healthy condition make a total of 10 different categories. It is a classification problem of 10 categories, and the data of the 10 categories were collected for four different working loads (0-3hp). Several sub-datasets are created based on the methods provided in [234] [30] as shown in Table 4.5, and the proposed method is evaluated on these 6 sub-datasets. The sixth sub-dataset was slightly different from the others, it was based on predicting the bearing condition under a high working load (3hp) based on the data collected from a low working load (0-2hp).

Table 4.5.: Experiment 2-Subdataset

Subdataset	1	2	3	4	5	6
Training data	0 hp	1 hp	2 hp	3 hp	0-3 hp	0-2 hp
Testing data	0 hp	1 hp	2 hp	3 hp	0-3 hp	3 hp

Data organisation

As the vibration signal from the 2 sensors was recorded during a certain time period for each type of condition, and the length of each file was more than 120,000 data points, they need to be chunked into small portions. A 2x120 was chosen as the window size of one portion (120 data points for each vibration sensor) and the data was reorganised to a matrix with 4 rows and 60 columns as the input of our deep network.

Experiment results

The experiment results of the proposed method and the results obtained in previous research are shown in Table 4.6.

Table 4.6.: Experiment 2-Results comparison

Method	Subdataset					
	1	2	3	4	5	6
[235]	88.9%	-	-	-	-	-
[236]	-	-	-	92.5%	-	-
[234]	98.8%	98.8%	99.4%	99.4%	99.8%	96.8%
[30]	100%	100%	100%	99.96%	99.95%	98.80%
Our method	99.12%	99.61%	99.49%	99.9%	99.84%	81.52%

It was observed that the best results of previous research have been achieved in [30]. They used a 16-layer Visual Geometry Group (VGG-16) as a feature extractor to extract features in time-frequency images of vibration signals. Compared with their results, the proposed method has similar accuracy in tasks 1-5 (nearly 100%). However, this method shows a worse accuracy than the accuracy in [30] in task 6. This may be because this research used the raw data in the time domain as compared to the frequency domain analysis used in [30]. As mentioned in the task description, the requirement of this task is to use low working load data to predict failures under heavy working loads and the speed of the motor remains constant. In frequency domain bearing vibration analysis, the rotation speed of the shaft is a main factor affecting the frequency feature distribution of vibration signal [237], and the severity of defects, rotation speed, and working load mainly affect the amplitude of each frequency element [221]. Therefore, as the rotation speed remains constant in this experiment, the frequency domain analysis is less sensitive than the time domain analysis.

4.4.3 Experiment 3: Gearbox dataset

Task description

This dataset was created by Shao et al on a dynamic simulator which comprised of a motor, a shaft, a gearbox, and a brake [30]. This task required the identification of the gear and bearing working conditions of the gearbox based on the vibration signal. Gear and bearings have four faulty working conditions and one healthy working condition.

Data organisation

The time window size for the vibration signal was kept at 4000, this implies that the input data had 4000 data points, organised as a 40x100 matrix. This model treats each 40x100 input as 40 different data sources and attempts to capture features from inter- and intra- data sources.

Experiment results

The experiment results are shown in Table 4.7. As can be seen from the table, the accuracy obtained by our proposed method is slightly higher than that obtained by the best results in previous studies [30] [238]. Compared with the

Table 4.7.: Experiment 3-Results comparison

Classification method		Bearing		Gear	
		Low load	High load	Low load	High load
[238]	SAE-DNN	87.5%	92.1%	92.7%	91.9%
	GRU	91.2%	92.4%	93.8%	90.5%
	BiGRU	93.0%	93.6%	93.8%	90.7%
	LFGRU	93.2%	94.0%	94.8%	95.8%
[30]	Pre-trained CNN	99.94%	99.42%	99.64%	99.02%
Our proposed method		99.10%	99.85%	99.90%	99.92%

method proposed that used wavelet transformation of vibration signal as input data [30], the proposed method uses the raw vibration signal and does not require any artificial feature extraction or data transformation.

4.5 Discussion

The experiment results demonstrate that it is feasible to use the deep model transferred from NLP, in this case, a Transformer-based model called GPT-2, to handle the task of multi-sensor fusion for industrial applications. The proposed method offers advantages in the following aspects:

- Interpretability: The attention heat map indicates the decision basis of the deep learning model which can be used for key sensor identification, thereby assisting engineers to conduct system diagnosing or maintenance or reducing the redundancy of the system. The results from experiment 1 demonstrate that using the data from important sensors to classify system conditions is more accurate than using data from all the sensors. This suggests that the AM has a positive effect on identifying key sensors. Such a feature is not available in most other deep-learning models. However, it is worth noting that no attention allocated to a sensor does not necessarily mean that the sensor does not have enough information. It can only indicate that the sensor data with strong attention are easier to harness during the backpropagation process. In other tasks with accuracy above 95%, there was little difference between the results obtained by using the sensors with high attention and all the sensors.

- Enable the use of deep learning models with limited industrial data. Normally, transfer learning is a hot topic to solve the data shortage in industrial scenarios requiring that the two datasets have some kind of similarity [172] [239]. However, even collecting enough data on similar industrial processes is costly and finding similar industrial processes also has limitations. Our proposed method demonstrates that a deep model pre-trained in natural language can be transferred to industrial sensor fusion tasks. As natural language is a data-rich modality, deep learning models can be sufficiently trained from it. Therefore, the use of deep learning for sensor fusion tasks is less likely to be limited by the limited industrial data and the lack of similar industrial processes.

- Eliminate the need for artificial feature engineering: Manual feature extraction and selection are labour-intensive, difficult and have high uncertainty. However, in order to combine different sensors and reduce the complexity of the input space, it is usually necessary. In the proposed method, the sensor data with different sampling rates were combined and mapped to a unified embedding space, and a deep learning model was used to extract features from the embedding space automatically. Therefore no manual feature engineering was required.

However, while the model performs well on classification problems as shown in this paper, it shows less capacity on the regression tasks, such as remaining useful life prediction tasks. The reason for this may be that stacked attention layers are better at searching for key information rather than mapping features to a specific value. As mentioned in the previous Chapter, the output of AM is calculated based on Q, K, and V vectors, where Q and K vectors are used to search and match information and only the V vector is used to extract features. As the V vector is obtained only using a linear layer, therefore AM may be less capable of extracting abstract features from the data. In order to adapt the AM to the regression task, the computational mechanism of the attention layer may need to be adapted, which might be a possible future research direction. In addition, the large model size and high memory usage are other limitations of this approach. The computational complexity and memory usage of this method grows by the square of the length of the input sequence [240]. As a result, it may cause high occupancy of computing resources in industrial applications. How to minimise the model and find a balance between model performance and resource consumption may be another future research direction.

4.6 Summary

In conclusion, this work proposed a new deep transfer learning method to deal with industrial sensor fusion tasks. The results of condition monitoring of a hydraulic system show that the proposed method has achieved high accuracy without feature engineering in an extremely large input space. This proposed method allows industrial scenarios to use deep models with a relatively small amount of data. In its accumulator conditions classification task, the accuracy can be further improved from 91.4% to 96.4% if only the sensor data with high attention scores are used as input. This phenomenon suggests that AM has a positive effect on improving the interpretability of the deep learning model. As can be seen from the results of bearing condition classification, the proposed method achieves similar accuracy to the best results of previous studies in Tasks 1 to 5. However, it shows an unsatisfactory result in Task 6. The reason for the result in task 6 may be due to the fact that the frequency domain analysis used in [30] has an advantage in predicting high workloads conditions using low workloads data when compared to the time domain analysis used in this study. As for gearbox condition classification, the proposed method is slightly more accurate in comparison with the accuracy obtained in [30].

The results of this research show that the pre-trained NLP model GPT-2 based on the architecture of the Transformer also has the potential to handle multi-sensor fusion tasks. The GPT-2 acts as a feature extraction engine that could replace manual feature extraction thus eliminating the difficult choice of using a deep model or a shallow model with artificial feature extraction for industrial sensor fusion tasks. Moreover, the experimental results show that the deep learning model, in this case, GPT-2, trained from natural language can be transferred to industrial sensor data, which means it may not be necessary to collect data for specific kinds of machines or processes before using deep transfer learning. Hence, the cost and time required for industrial data collection can be significantly reduced. In addition, the AM allows the model to provide not only the prediction information but also the basis for the model´s decision-making which might be beneficial for industrial applications.

The main contributions of this work can be summarised as follows:

- A novel deep transfer learning solution that generalises the feature representation from a data-rich modality, in this case, NLP, to address challenges in sensor fusion. This work demonstrates that when using transfer learning in industrial scenarios, collecting data from similar industrial processes for pre-training may not be necessary. The proposed transfer learning method can effectively reduce the required amount of data when using deep learning for industrial sensor fusion tasks, thus benefiting from the learning ability of deep models and to some extent, eliminating the trade-off between deep and shallow models mentioned above. To the best of my knowledge, it is the first work that uses the model trained from language to solve the industrial sensor data processing problem.

- The problem of poor interpretability when using Deep Learning in sensor fusion tasks is alleviated. Based on the attention mechanism, the decision basis of the deep learning model can be inspected, thus the key sensors that are highly related to the final decisions can be identified. Instead of analysing data from all sensors, it allows for a narrower analysis during diagnostics.

- A novel deep learning solution to automatically establish a unified feature representation and association relationship for the sensor data at significantly different sampling rates from different modalities. For conventional sensor fusion methods, decision-level fusion is a better option if the types of sensors are significantly different [72]. However, the challenge for using decision-level fusion is that a large number of features which can be highly correlated have to be created. This could bias the final decision, and these features have to be processed properly [72]. In our proposed method, instead of utilising the raw data directly, the sensor data with different sampling rates will be combined and mapped to an embedding space, and the features among different sensors can be extracted from the embedding space by a deep neural network automatically. Hence, the challenge of decision-level fusion can be avoided, the need for artificial feature engineering can be eliminated, and the sensors with different sampling rates can also be combined easily.

5

Sensorformer: A Memory-efficient Transformer for Industrial Sensor Fusion

5.1 Introduction

As discussed in the previous chapters, the Transformer method can be an effective candidate for sensor fusion, offering the following advantages over similar techniques:

- Compared with the local pattern-matching process (fixed kernels) of CNN, the self-attention mechanism can adaptively discover the large-range dependencies by its attention mechanism, hence, it has stronger modelling capabilities and it may be a more efficient way to extract high-level information [163][164].

- Transformer has remarkable extensibility, which means significant performance improvements can be observed as the model gets larger, such as Turing Natural Language Generation (Turning-NLG)[241] and Generative Pre-trained Transformer (GPT)[165].

- Some studies have shown that it has the ability to combine different modalities, which is beneficial for making more comprehensive inferences[242].

However, since the computational complexity of the transformer is $O(N^2)$, and the memory usage is also the square of the sequence length, computing time and required resources increase with the number of sensors (length of sequence)[243], especially for processing the high sampling rate sensor data. This can result in a significant occupation of resources in industrial applications, which means less efficiency and more cost, especially in the case of edge computing. The commonly used Principal Components Analysis (PCA) and the removal of highly related sensor data can effectively reduce the sequence length of the input, however, they may not be the best choice for a multi-sensor system. In terms of PCA, the principal components could be less representative than the original data, as the components with small variances may represent influential information. As for removing the highly related sensor, for example, in[141], it may contradict the original purpose of designing a multi-sensing system. The purpose of introducing redundant sensors is to increase the robustness of the monitoring system or to detect the same physical quantity at different locations. These data can be highly related to each other. If these sensor data are removed in the pre-processing stage to reduce the length of the input sequence, then this can result in the loss of many advantages of a multi-sensor system.

In this work, a novel deep learning architecture, Sensorformer, was proposed to mitigate the problems mentioned above. An auto-decorrelation block based on the Fast Fourier Transform (FFT) was integrated with the self-attention block to reduce memory occupation and increase computing speed. Instead of removing the related data, this model automatically merges the related data during the processing of the data. In addition, due to the inherent $O(NlogN)$ computational complexity of FFT, the self-attention-based auto-decorrelation block can also reduce the computational complexity from $O(N^2)$ to $O(NlogN)$, hence the memory occupation and computing time can be reduced significantly.

5.2 Problem Statement

The Transformer and its variants have achieved impressive success across numerous fields, such as natural language processing, vision recognition, and multi-sensor industrial data processing. However, their application in industrial scenarios, especially in edge computing environments, can be often constrained by their high memory usage and computational complexity, which pose challenges in terms of cost, efficiency, and feasibility, particularly when dealing with large-scale or real-time tasks. Therefore, this research focuses on reducing the memory usage and computational complexity of the Transformer for sensor fusion tasks. As the sources of the high computational load of the Transformer are the self-attention mechanism and the large input space, the goals of this research are to reduce the computational complexity of the attention calculation and the size of the input sequence simultaneously in an end-to-end manner.

5.3 Methodology and Architecture

5.3.1 Overall architecture

Sensorformer is composed of two different blocks, the self-attention block and decorrelation block as shown in Fig.5.1 where N and M represent the number of blocks, Q, K, and V are Query, Key, and Value respectively similar to Transformer in[162]. The former block is the original transformer self-attention block which is used to establish the unified feature representation of multi-sensor data[162], and the latter block is used to reduce the sequence length before sending the data to the transformer block by merging the correlated channels automatically. Our Sensorformer harnesses the removal of correlated channels as an inner block of the deep learning model, which can progressively merge the correlated channels throughout the whole inference process. Note that the number of merged channels will increase with the number of decorrelation blocks. The feed-forward layer of the decorrelation block is used for non-linear projection

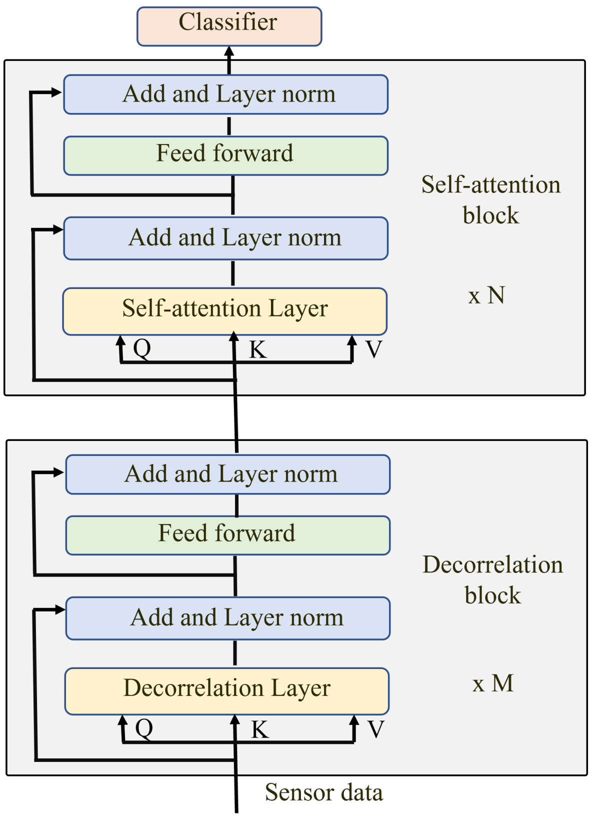


Figure 5.1.: Sensorformer Architecture

and its weights can be updated automatically during the backpropagation process, hence the calculation of the correlation coefficient will not be restricted to linear calculation.

The classifier used in our model is a simple one-layer neural network, and the feed-forward layer is a three-layer neural network with a $ReLU$ activation function.

5.3.2 Model input

The input of this model can be described by $\mathcal{X} \in \mathbb{R}^{d \times L}$, where L is the time window size of one sample, and d is the sequence length that is affected by the sampling rates of multiple sensors. For example, if we have a 1-second time window sample consisting of 2 sensors with the sampling rates of 10(Hz) and 50(Hz) respectively, L will be 10 and d will be $10(Hz)/10 + 50(Hz)/10 = 6$. $\mathcal{X}_i \in$

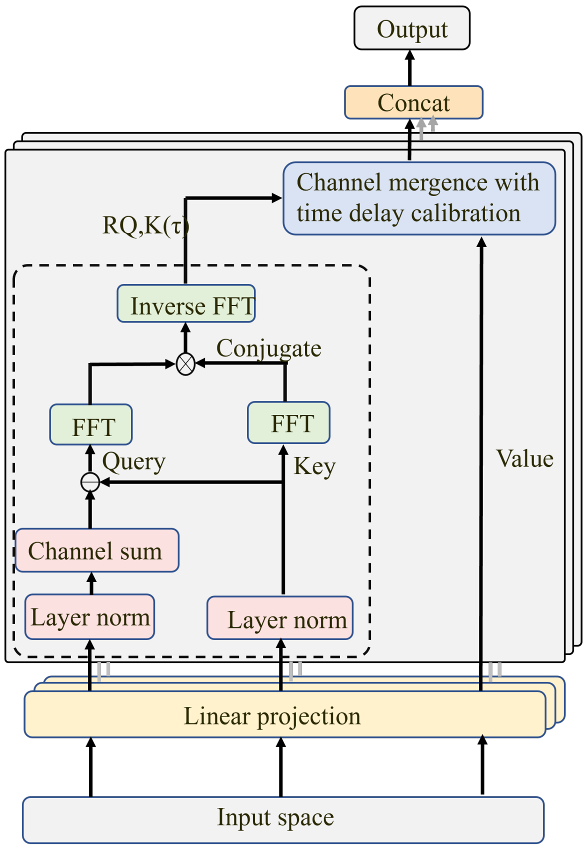


Figure 5.2.: Decorrelation layer

$\mathbb{R}^{1\times L}$ denotes the i-th element of the input sequence.

5.3.3 Decorrelation layer

The decorrelation layer shown in Fig.5.1 can be illustrated by Fig.5.2. We propose this mechanism with channel-wise data mergence to improve the efficiency of information utilisation and reduce computation load. The mechanism of this layer will be explained in the following subsections.

Pearson correlation coefficient calculation

In this research, Pearson's correlation coefficient is used to measure the correlation, which can be described by the following equation:

$$\rho_{\mathcal{X},\mathcal{Y}} = \frac{\mathrm{cov}(\mathcal{X},\mathcal{Y})}{\sigma_{\mathcal{X}}\sigma_{\mathcal{Y}}} \tag{5.1}$$

To simplify the calculation, based on stochastic process theory, the following equation is used in practice to estimate Pearson's correlation coefficient[244]:

$$\mathcal{R}_{\mathcal{X}_i\mathcal{Y}_i} = \frac{1}{L-1}\sum_{i=1}^{L} X_i Y \tag{5.2}$$

where

$$X_i = \left(\frac{\mathcal{X}_i - \bar{\mathcal{X}}}{\sigma_{\mathcal{X}_i}}\right), Y = \sum_{i=1}^{L} X_i - X_i \tag{5.3}$$

σ_{X_i} and $\bar{\mathcal{X}}$ are standard deviation and mean value of $\mathcal{X}_i$. Note that Y_i is the reference variable which means the correlation between $\mathcal{X}_i$ and the sum of the rest of the channels is evaluated here by $\mathcal{R}_{\mathcal{X}_i\mathcal{Y}_i}$. This operation is to avoid calculating C_L^2 times the correlation coefficient to identify the related channels. In this case, only an L-times calculation is needed. The following equation is used to describe the correlation between X_i and Y_i at the time delay τ:

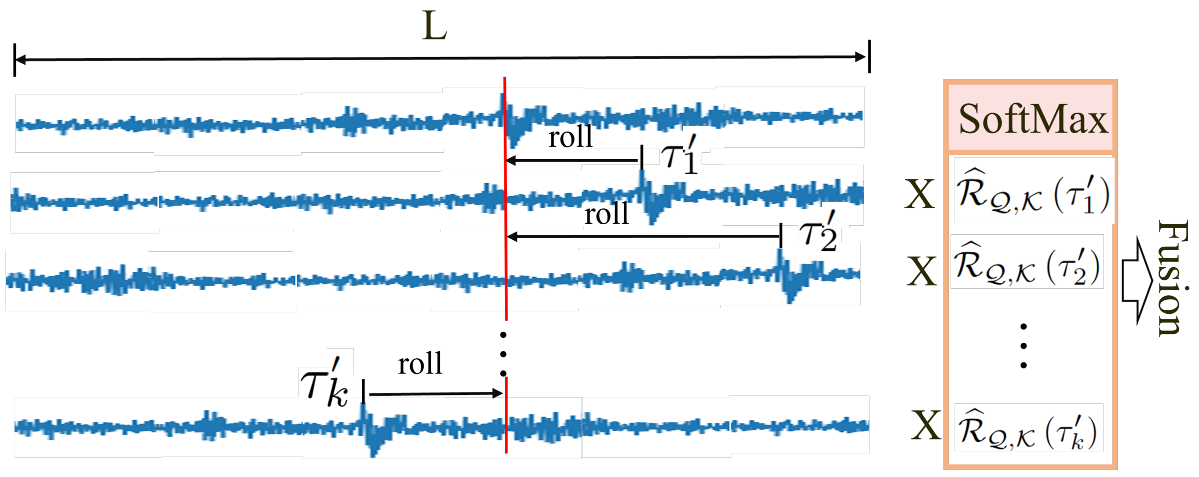
$$\mathcal{R}_{\mathcal{X}_i\mathcal{Y}_i}(\bar{\tau}) = \lim_{L\to\infty}\frac{1}{L}\sum_{t=1}^{L} X_i Y_{i\tau} \tag{5.4}$$

Based on Wiener-Khinchin theorem, for computational efficiency, $\mathcal{R}_{\mathcal{X}_i\mathcal{Y}_i}(\bar{\tau})$ can be derived by FFT[245]:

$$\begin{aligned}\mathcal{S}_{XY}(f) &= \mathcal{F}\left(X_t\right)\mathcal{F}^*\left(Y_t\right)\\ &= \int_{-\infty}^{\infty} X_t e^{-i2\pi t f}\,\mathrm{d}t \overline{\int_{-\infty}^{\infty} Y_t e^{-i2\pi t f}\,\mathrm{d}t}\end{aligned} \tag{5.5}$$

$$\mathcal{R}_{XY}(\tau) = \mathcal{F}^{-1}\left(\mathcal{S}_{XY}(f)\right) = \int_{-\infty}^{\infty} \mathcal{S}_{XY}(f) e^{i2\pi f\tau}\mathrm{d}f \tag{5.6}$$

where $\tau \in \{1, \cdots, L\}$, $\mathcal{F}$ and $\mathcal{F}^{-1}$ represent the FFT and its inverse, $\mathcal{F}^*$ means $\mathcal{F}$'s conjugate matrix. As $\mathcal{R}_{XY}(\tau)$ for $\tau \in \{1, \cdots, L\}$ can be obtained at the same time by FFT, the computational complexity is $O(LlogL)$. The entire process mentioned above can be described by the dashed box in Fig.5.2.



L
roll
τ_1'
roll
τ_2'
τ_k'
roll
SoftMax
X $\widehat{\mathcal{R}}_{\mathcal{Q},\mathcal{K}}(\tau_1')$
X $\widehat{\mathcal{R}}_{\mathcal{Q},\mathcal{K}}(\tau_2')$
X $\widehat{\mathcal{R}}_{\mathcal{Q},\mathcal{K}}(\tau_k')$
Fusion

Figure 5.3.: Channel mergence with time delay calibration

Channel mergence with time delay calibration

As shown in Fig.5.3, this layer is used to align the correlated channels at their maximum similarity based on the correlation coefficient matrix obtained in the last section. Then the correlated channels will be fused into a single channel, hence the sequence length can be reduced in a reasonable manner.

Firstly, the signal is rolled τ_i steps to make sure the channels can be fused at their maximum similarity. This is because the time delay between sensor data may vary due to factors such as the location of the sensors or the characteristics of the industrial process. Fusing the sensor data at their maximum similarity can mitigate the impact of this time delay effect on the fused data. τ_i is calculated by the following equation: τ_i is calculated by the following equation:

$$\tau_i = \underset{i \in \{1,\cdots,d\}}{\arg \mathrm{Max}} \left(\mathcal{R}_{X_i,Y_i}(\tau) \right) \tag{5.7}$$

Then, the k most relevant channels are fused into one channel based on the $SoftMax$ scores of their correlation coefficients. The channel mergence layer can be described by the following equations:

$$\{\tau_1', \cdots, \tau_k'\} = Topk\{\tau_1, \cdots, \tau_i\} \tag{5.8}$$

$$\widehat{\mathcal{R}}_{\mathcal{Q},\mathcal{K}}\left(\tau_1'\right),\cdots,\widehat{\mathcal{R}}_{\mathcal{Q},\mathcal{K}}\left(\tau_k'\right)=\mathrm{SoftMax}\left(1-\mathcal{R}_{\mathcal{Q},\mathcal{K}}\left(\tau_1'\right),\cdots,1-\mathcal{R}_{\mathcal{Q},\mathcal{K}}\left(\tau_k'\right)\right) \quad (5.9)$$

$$\text{ChannelMergence}\left(\mathcal{Q}_k,\mathcal{K}_k,\mathcal{V}_k\right)=\sum_{i=1}^{k}\mathrm{Roll}\left(\mathcal{V}_k,\tau_k'\right)\widehat{\mathcal{R}}_{\mathcal{Q},\mathcal{K}}\left(\tau_k'\right) \quad (5.10)$$

where $Topk\{*\}$ is the operation that takes the largest k values, and $Roll(*)$ is the rolling operation mentioned above. Note that the SoftMax is calculated based on $(1-\mathcal{R}_{\mathcal{Q},\mathcal{K}}\left(\tau_k'\right))$. This is because the channels that are highly correlated with other channels are expected to have smaller weights, thus making channels that are less correlated have large weights. $\mathcal{R}_{\mathcal{Q},\mathcal{K}}$ is the correlation coefficient between Q and K. Note that k is a hyper-parameter that controls how many channels will be merged at each of the decorrelation blocks. The output of ChannelMergence is a single channel, and this merged channel will be concatenated with the rest of the channels. Therefore, the channel number of the decorrelation layer output will be $d-k+1$.

The multi-head operation used in Sensorformer can be described in the following equations:

$$\mathrm{MultiHead}(\mathcal{Q},\mathcal{K},\mathcal{V})=\text{Concat}\left(\text{head}_1,\cdots,\text{head}_h\right) \quad (5.11)$$

where h is the number of heads. Each $head_h$ is the operation of the whole decorrelation layer, and the input of each head can be described by $\mathcal{X}_{head_h}\in\mathbb{R}^{d\times\frac{L}{h}}$.

5.3.4 Decorrelation block

Similar to the self-attention block of Transformer, as shown in Fig.5.1, the decorrelation layer is connected to a feed-forward layer via a shortcut connection[184] and layer normalisation[183]. The feed-forward layer also has its shortcut connection and layer normalisation.

Table 5.1.: Monitored Parameters of the Hydraulic System

Sensor	Physical quantity	Unit	Sampling rate
PS1-6	Pressure	bar	100Hz
EPS1	Motor power	W	100Hz
FS1-2	Volume flow	1/min	10Hz
TS1-4	Temperature	°C	1Hz
VS1	Vibration	mm/s	1Hz
CE	Cooling efficiency	%	1Hz
CP	Cooling power	kW	1Hz
SE	System efficiency factor	%	1Hz

5.4 Experiments and Results

This experiment compared the inference speed, memory usage, and prediction accuracy of the original Transformer and our proposed Sensorformer on the same dataset (details below). The size of the Transformer and Sensorformer were kept the same to make them comparable. The details of this experiment are explained in this section.

5.4.1 Experiment dataset

The proposed method was tested on a public industrial multi-sensor dataset: Condition Monitoring of a Hydraulic System, and it was created by Helwig et al. in the experiment described in [23]. This experiment used 17 sensors with different sampling rates to measure different physical quantities at different locations of a hydraulic system as shown in Table 5.1.

Based on these sensor data, the system conditions as shown in Table 5.2 were

Table 5.2.: Conditions of the Hydraulic System to be Predicted

Components	Component conditions
1 Cooler condition	(1) Close to failure (2) Reduced efficiency (3) Full efficiency
2 Valve condition	(1) Optimal switching behaviour (2) Small lag (3) Sever lag (4) Close to total failure
3 Internal pump	(1) No leakage (2) Weak leakage (3) Sever leakage
4 Hydraulic accumulator	(1) Optimal pressure (2) Slightly reduced pressure (3) Sever reduced pressure (4) Close to failure
5 Stable flag	(1) Condition were stable (2) Non-static conditions

expected to be identified. The total number of attributes for one data snapshot was $8(sensors) \times 60(s) \times 1(Hz) + 2(sensors) \times 60(s) \times 10(Hz) + 7(sensors) \times 60(s) \times 100(Hz) = 43680$, and the time window size was 60 seconds, hence, as described in the previous section, the input space can be described by:

$$\mathcal{X} \in \mathbb{R}^{728 \times 60} \tag{5.12}$$

The data were normalised to 0 to 1 based on the following equation:

$$X_{norm} = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{5.13}$$

5.4.2 Experiment environment and Sensorformer parameters

This experiment was conducted on Google Colab environment with NVIDIA Tesla P100 PCIe 16 GB, and PyTorch was used as the deep learning framework. The hyper-parameters of Sensorformer and Transformer are shown in Table 5.3.

Table 5.3.: Hyper-parameters of Transformer and Sensorformer

Hyper-parameters	Sensorformer	Transformer
Batch size	30	30
Learning rate	0.0001	0.0001
Optimiser	Adam	Adam
Loss	Cross Entropy	Cross Entropy
Self-attention blocks	8	12
Self-attention heads	6	6
Decorrelation blocks	4	0
Decorrelation heads	2	N/A
Topk	100	N/A
Initialisation	Orthogonal	Orthogonal

5.4.3 Experiment Results

In this experiment, we tested the performance of the Transformer and Sensorformer with 100% GPU occupancy to compare the memory efficiencies. As shown in Table 5.3, both models used the same parameters and they were both set to the same size (12 blocks). For the Transformer, the depth was 12 self-attention blocks. As for the Sensorformer, it was composed of 4 decorrelation blocks and 8 self-attention blocks. The results are shown in Fig. 5.4, the inference time and memory usage represent the average inference time and average memory usage of a single sample with 100% GPU memory usage. Based on the results, it can be found that with maximum GPU occupation, our proposed method takes only 50% of the inference time of the Transformer. In terms of memory usage, Transformer uses 4.6 times more GPU memory than Sensorformer. When deploying deep learning models for industrial multi-sensor data

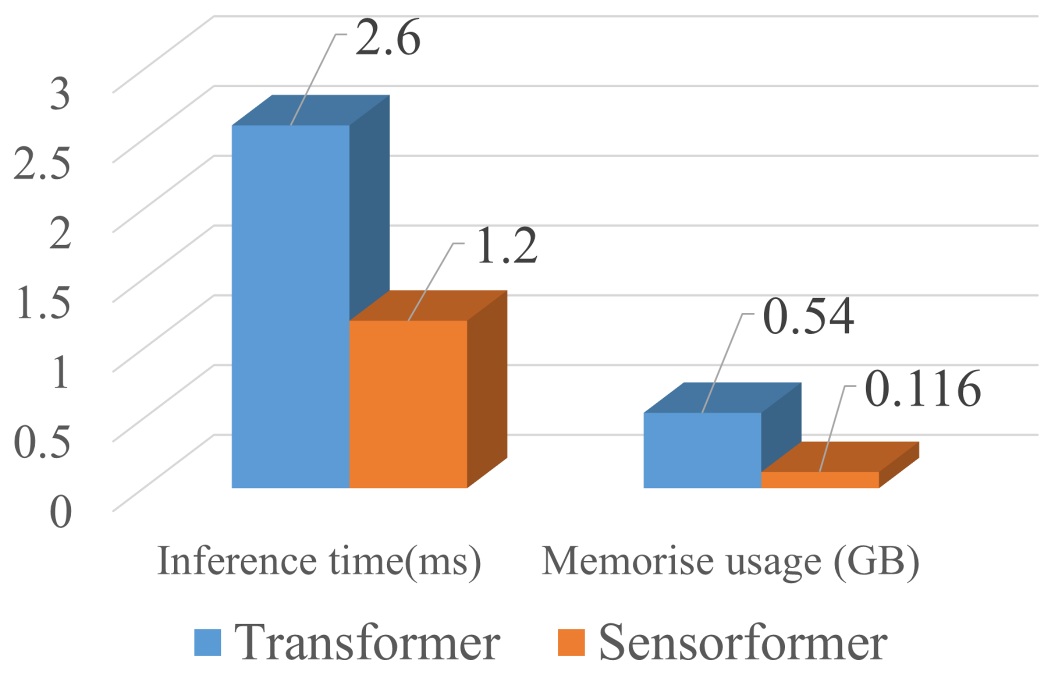


Figure 5.4.: Results of model speed and memory usage comparison

processing, lower memory usage and less inference time mean more data can be processed with the same computational resources. Our proposed method is therefore more efficient and less costly.

The result of the accuracy of our proposed method can be found in Table 5.4. The accuracy was evaluated by the following equation:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \tag{5.14}$$

Based on the results, Sensorformer achieved the top four accuracies. Compared with the 12-block Transformer, the accuracy of the Sensorformer only dropped by 0.3%, but the memory efficiency and inference speed improved significantly.

Table 5.4.: Performance Comparison

Research group	Method	Cooler	Valve	Pump	Accumulator	Mean
Helwig et al. [23] (2015)	LDA	100%	100%	98.0%	90.4%	97.1%
Berghout et al. [225] (2021)	Auto-NAHL	100%	100%	100%	96.4%	99.1%
Wu et al. [226] (2020)	EGMSVMs	100%	100%	100%	76.5%	94.1%
Gupta et al. [227] (2021)	SECM	N/A	N/A	N/A	N/A	92.3%
Lei et al. [228] (2019)	PCA+XGBoost	N/A	96.58%	N/A	N/A	N/A
Huang et al. [141] (2021)	Deep CNN	100%	100%	99.0%	99.4%	99.6%
Prakash et al. [130] (2020)	ANN+XGBoost	99.54%	N/A	N/A	N/A	N/A
Our results	**Sensorformer**	**99.57%**	**100%**	**95.52%**	**98.57%**	**98.2%**
Our results	**Transformer**	**99.57%**	**99.79%**	**97.76%**	**96.78%**	**98.5%**

5.5 Discussion

As described in [162], in the self-attention mechanism, the similarity calculation is based on the multiplication of two matrices of K(n,d) and Q(d,n), where n is the length of the input sequence and d is the embedding dimension. This results in its computational complexity being $O(n^2 \cdot d)$. In comparison, the similarity calculation of our proposed decorrelation layer is based on the Pearson correlation coefficient which can be converted to FFT-based calculation, hence the $O(nlogn)$ computational complexity is achieved. The calculated correlation coefficient can be a function of the time delay τ. By rolling the signal τ steps as shown in Figure 5.3, the signals can be synchronised before being merged, which reduces the input sequence length. This operation is performed in each of the proposed decorrelation blocks, hence the redundancy of the multi-sensor signal can be reduced progressively. Therefore, compared to Transformer, our proposed model has more advantages in terms of resource occupancy. In addition, compared with the conventional redundancy removal method that removes collected sensors in the data pre-processing stage, the proposed method merges the correlated channel, resulting in less information loss.

5.6 Summary

The deep learning model, Transformer, has achieved state-of-the-art performance in many domains, including the industrial sensor data processing domain. However, due to the high memory usage and $O(N^2)$ computational complexity, it can be resource-consuming, especially for processing industrial multi-sensor data with high sampling rates. In this paper, we proposed Sensorformer with a novel decorrelation block based on Fast Fourier Transform and the self-attention architecture to reduce memory usage and inference time. This method does not require the removal of highly relevant sensors during the data pre-processing phase, hence the robustness and comprehensiveness introduced by redundant sensors can be kept. The experiment results show that compared with the original Transformer architecture, Sensorformer takes only half of the inference time, and uses one-fifth of the GPU memory of the Transformer with remarkable accuracy.

6

Discussion and Conclusions

6.1 Research Contributions

The thesis outlines significant contributions towards enhancing sensor data processing and anomaly detection in industrial settings, leveraging advanced deep learning techniques. These contributions, aligned with the thesis objectives, are summarized as follows:

Contribution 1

- Novel Model for Sensor Data Anomaly Detection: Introduced a novel anomaly detection model combining self-attention mechanisms with CNN, focusing on sensor data reliability and quality for industrial process monitoring.
- Dual-Channel Self-Attention Structure: Developed a unique dual-channel self-attention framework, integrating sensor-wise and time-wise attention channels, to capture spatiotemporal features effectively.

- Improved Sensitivity and Performance: Demonstrated significant improvement over state-of-the-art methods on a public CAV dataset, indicating fewer undetected anomalies and enhanced anomaly detection capabilities.

Contribution 2

- Deep Transfer Learning for Limited Data Scenarios: Proposed a novel deep transfer learning approach to utilize Transformers trained on natural language for sensor fusion tasks in industrial process monitoring, addressing the challenge of limited data availability.
- First to Transfer Pre-trained Language Model for Industrial Sensor Fusion: Pioneered the application of a pre-trained large language model to address industrial sensor fusion problems, tested across multiple datasets with high accuracy and reduced need for manual feature engineering.
- Enhanced Interpretability and Feature Representation: Enhanced interpretability through the self-attention mechanism's ability to identify critical sensors and automatic feature extraction, offering a novel approach to multi-sensor representation without manual feature engineering.

Contribution 3

- Innovative Auto-Decorrelation Block: Introduced an auto-decorrelation block combining Fast Fourier Transform with self-attention to lower memory usage and computational complexity, making the Transformer more suitable for industrial applications.
- Resource Efficiency and Speed: Demonstrated significant reductions in memory usage and inference time on a hydraulic system monitoring dataset, enabling the efficient application of Transformers in resource-constrained

industrial environments.

These contributions collectively advance the field of industrial process monitoring by integrating cutting-edge deep learning techniques to address practical challenges such as limited data availability, the need for efficient computational resource usage, and the improvement of sensor data processing and anomaly detection accuracy.

6.2 Research Limitations

The thesis acknowledges several limitations across its objectives, which are critical for interpreting its results and findings:

Limitations of the work in Chapter 3

- The model's real-system performance presented in Chapter 3 remains unverified as sensor anomalies were manually injected based on common patterns, not actual anomalies.
- The supervised learning approach limits the model's ability to handle unknown and complex anomalous patterns, posing potential risks in safety-critical systems due to its limited generalisation capability.

Limitations of the work in Chapter 4

- The data-driven method for sensor feature representation requires labeled data, restricting its use in non-repeatable industrial processes.
- High-frequency sensors may dominate the embedding space due to their proportional representation, making the method unsuitable for integrating vastly different sensor frequencies, such as vision and numerical data, thereby limiting its applicability in certain monitoring tasks.

- High computational resources are demanded by the pre-trained Transformer model, especially for large sensor arrays or high-frequency sensors, due to the self-attention mechanism's memory footprint.
- The model's large size necessitates considerable storage space, potentially limiting widespread deployment in industrial settings where computing performance and storage are constrained.

Limitations of the work in Chapter 5

- Modifications to the Transformer architecture prevent the use of the transfer learning method from Chapter 4, increasing the model's training data requirements.
- Hyperparameter selection, specifically the number of channels to merge based on sensor correlation, is challenging and requires extensive experimentation to optimize, with no guaranteed optimal solution.

6.3 Future Work

In this section, future works are summarised for each objective of this thesis.

Future Work of Objection 1

In sensor anomaly detection, the performance of supervised learning on unknown anomaly patterns can be usually questioned. A common deep learning-based approach in the literature for handling unknown pattern anomalies can be the unsupervised learning-based autoencoder architecture, which means reconstructing the original normal signal using a deep learning model and then using one of the latent spaces as a feature representation. By doing this, the distance between the latent space of the anomalous data and the latent space of the normal data can be calculated as a criterion for the occurrence of anomalies [149] [158]. Since the proposed DA-CNN architecture has advantages in spatiotemporal features extraction, this model also has the potential to be organised into autoencoder architecture to test the performance of DA-CNN in this type of methodology in the future. This enables the model to have the potential to detect any anomalous behaviour.

Future Work of Objection 2

The current establishment of multi-sensor feature representation is limited to the fusion of numerical sensor data. However, it is frequently the case that many industrial processes generate data in many significantly different modalities, such as sensors, images, text, audio, or even human input, providing a more comprehensive understanding of a given process. For example, the welding process, visual, acoustic, as well as vibration and current parameters etc. can provide important information [246]. As Transformer can be relatively easy to model the dependencies between different parts of the input, it can be a very promising possibility for multi-modal fusion if a proper method can be developed to create a unified feature representation for data from different modalities. Transformer has achieved remarkable performance on image-text and image-LiDar fusion [247] [248] [249], but research on industrial sensor fusion remains limited.

Future Work of Objection 3

Transfer learning based on pre-trained models can significantly reduce the data requirements for training deep learning models, which can be very effective in natural language [165] as well as image processing [250]. This is because the learning of many basic features does not require task-specific data to be captured, for example in image recognition, points, lines, colours, geometries, textures and other basic features are required by almost all image recognition tasks, so a large amount of feature learning can be done with related data. In the field of industrial sensor data, different sensor data may also have many similar underlying feature learning processes, such as the acquisition of statistical features, and features in the time domain frequency domain. Similar to visual features such as points, lines, textures, etc., the specific representation of these sensor data features may be different, but their computational processes can be the same, offering the possibility of pre-training models in the field of industrial sensors. The current transfer learning method employs the model pre-trained from natural languages, whether the industrial version of a pre-trained deep learning model can be more problem-specific and hence improves the performance can be worth being investigated.

Future Work of Objection 4

In the work presented in Chapter 5, the correlations of the different sensors are not calculated using their raw data, but using their non-linearly mapped expressions. This non-linear mapping is learned by a neural network, which means that the sensor correlation is calculated in its feature space. This gives us the possibility to calculate correlations for data of different modalities such as visual, numerical signals, etc. Data from different modalities can first be mapped in a data-driven manner to a unified representation from which correlations can then be calculated, thereby reducing redundancy in a multi-modal system to reduce computational load. Industrial process monitoring is moving in the direction of being more intelligent, integrated and comprehensive, so how to reduce the computational burden for multi-modal monitoring systems may become a promising direction for research.

6.4 Conclusions

This thesis presents research on deep learning-based multiple-sensor fusion for industrial process monitoring applications. Industrial process monitoring can be an important guarantee for the safety, efficiency and economy of industrial production. Due to the high complexity of industrial processes, traditional sensor fusion algorithms and manual feature engineering can be sometimes difficult and time-consuming. As a result, end-to-end approaches to automatic feature learning based on deep learning have received a high level of attention. In addition, due to the development of IIoT and communication technologies, the availability of industrial data is rapidly increasing, laying the foundation for data-driven approaches. Therefore, this thesis focuses on how deep learning-based methods can be applied and adapted to industrial sensor fusion tasks, contributing to the process monitoring domain.

Based on the research presented in this thesis, it can be found that the Transformer architecture and the self-attention mechanism in deep learning algorithms can be effective not only in multi-sensor anomaly detection tasks but also in complex industrial process condition analysis. This is due to the fact that the self-attention mechanism can simultaneously take into account the internal features of single-sensor data as well as the global dependencies among multiple sensors, which can be very important when dealing with multi-sensor tasks. Furthermore, as the nature of the self-attention mechanism is to calculate the extent to which each feature will be used for model decision-making, it can mitigate to some extent the drawbacks of deep learning methods in terms of their lack of interpretability by visualising the attention weights. This advantage can be used to identify key sensors within a multi-sensor system, which can be useful for industrial process analysis, and to reduce the complexity of monitoring systems.

As with other deep learning algorithms, a large amount of training data is required when using Transformer, which is not friendly to industrial applications. The research in this thesis found that using pre-trained large models from the field of natural language processing can effectively alleviate this problem. Multi-sensor data can be embedded into the embedding space in a similar way to natural language embedding, and then interpreted for meaning using the pre-trained model by fine-tuning it with industrial data. As for the high computational complexity introduced by the Transformer, this can be mitigated by modifying its self-attention mechanism to a Fast Fourier Transform-based calculation of the correlation coefficient. This modification is specifically for multi-sensor data processing and this work can contribute to the use of Transformer in industrial scenarios.

References

[1] Andrew KS Jardine, Daming Lin and Dragan Banjevic. 'A review on machinery diagnostics and prognostics implementing condition-based maintenance'. In: *Mechanical systems and signal processing* 20.7 (2006), pp. 1483–1510 (cit. on p. 1).

[2] Maurizio Bevilacqua and Marcello Braglia. 'The analytic hierarchy process applied to maintenance strategy selection'. In: *Reliability Engineering & System Safety* 70.1 (2000), pp. 71–83 (cit. on p. 1).

[3] Mohammad A AlKazimi, Hanan Altabbakh, Susan Murray and Katie Grantham. 'Evaluating generated risk event effect neutralization as a new mitigation strategy tool in the upstream industry'. In: *Procedia Manufacturing* 3 (2015), pp. 1374–1378 (cit. on p. 2).

[4] Elizabeth B Kujawinski, Christopher M Reddy, Ryan P Rodgers et al. 'The first decade of scientific insights from the Deepwater Horizon oil release'. In: *Nature Reviews Earth & Environment* 1.5 (2020), pp. 237–250 (cit. on p. 2).

[5] KX Peng, L Ma and K Zhang. 'Review of quality-related fault detection and diagnosis techniques for complex industrial processes'. In: *Acta Automatica Sinica* 43.3 (2017), pp. 349–365 (cit. on pp. 2, 30, 59).

[6] Mustafa Kuntoğlu, Abdullah Aslan, Hacı Sağlam et al. 'Optimization and analysis of surface roughness, flank wear and 5 different sensorial data via tool condition monitoring system in turning of AISI 5140'. In: *Sensors* 20.16 (2020), p. 4377 (cit. on p. 2).

[7] C Scheffer, H Kratz, PS Heyns and F Klocke. 'Development of a tool wear-monitoring system for hard turning'. In: *International Journal of Machine Tools and Manufacture* 43.10 (2003), pp. 973–985 (cit. on p. 2).

[8] Karali Patra, Surjya K Pal and Kingshook Bhattacharyya. 'Artificial neural network based prediction of drill flank wear from motor current signals'. In: *Applied Soft Computing* 7.3 (2007), pp. 929–935 (cit. on p. 2).

[9] Mehrdad Nouri Khajavi, Ebrahim Nasernia and Mostafa Rostaghi. 'Milling tool wear diagnosis by feed motor current signal using an artificial neural network'. In: *Journal of Mechanical Science and Technology* 30.11 (2016), pp. 4869–4875 (cit. on p. 2).

[10] Mustafa Kuntoğlu and Hacı Sağlam. 'Investigation of progressive tool wear for determining of optimized machining parameters in turning'. In: *Measurement* 140 (2019), pp. 427–436 (cit. on p. 2).

[11] Onur Özbek and Hamit Saruhan. 'The effect of vibration and cutting zone temperature on surface roughness and tool wear in eco-friendly MQL turning of AISI D2'. In: *Journal of Materials Research and Technology* 9.3 (2020), pp. 2762–2772 (cit. on p. 2).

[12] Shuang Yi, Jinjin Li, Jiahua Zhu et al. 'Investigation of machining Ti-6Al-4V with graphene oxide nanofluids: tool wear, cutting forces and cutting vibration'. In: *Journal of Manufacturing Processes* 49 (2020), pp. 35–49 (cit. on p. 2).

[13] Harvey B Mitchell. *Multi-sensor data fusion: an introduction*. Springer Science & Business Media, 2007 (cit. on p. 3).

[14] Harvey B Mitchell. *Data fusion: concepts and ideas*. Springer Science & Business Media, 2012 (cit. on p. 3).

[15] Bahador Khaleghi, Alaa Khamis, Fakhreddine O Karray and Saiedeh N Razavi. 'Multisensor data fusion: A review of the state-of-the-art'. In: *Information fusion* 14.1 (2013), pp. 28–44 (cit. on pp. 3, 4).

[16] Federico Castanedo. 'A review of data fusion techniques'. In: *The scientific world journal* 2013 (2013) (cit. on pp. 3, 34).

[17] Wilfried Elmenreich. 'A review on system architectures for sensor fusion applications'. In: *IFIP International Workshop on Software Technolgies for Embedded and Ubiquitous Systems*. Springer. 2007, pp. 547–559 (cit. on p. 3).

[18] Po-Yu Chen, Shusen Yang and Julie A McCann. 'Distributed real-time anomaly detection in networked industrial sensing systems'. In: *IEEE Transactions on Industrial Electronics* 62.6 (2014), pp. 3832–3842 (cit. on p. 4).

[19] Man Lok Fung, Michael ZQ Chen and Yong Hua Chen. 'Sensor fusion: A review of methods and applications'. In: *2017 29th Chinese Control And Decision Conference (CCDC)*. IEEE. 2017, pp. 3853–3860 (cit. on p. 6).

[20] Zhiqiang Ge, Zhihuan Song and Furong Gao. 'Review of recent research on data-based process monitoring'. In: *Industrial & Engineering Chemistry Research* 52.10 (2013), pp. 3543–3562 (cit. on p. 6).

[21] Shen Yin and Okyay Kaynak. 'Big data for modern industry: challenges and trends [point of view]'. In: *Proceedings of the IEEE* 103.2 (2015), pp. 143–146 (cit. on p. 6).

[22] US Department of Transportation. *Safety Pilot Model Deployment Data*. Website. https://catalog.data.gov/dataset/safety-pilot-model-deployment-data. January 24, 2022 (cit. on pp. 8, 70).

[23] Nikolai Helwig, Eliseo Pignanelli and Andreas Schütze. 'Condition monitoring of a complex hydraulic system using multivariate statistics'. In: *2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings*. IEEE. 2015, pp. 210–215 (cit. on pp. 10, 26, 106, 108, 115, 117, 135, 139).

[24] Rolf Isermann. *Fault-diagnosis applications: model-based condition monitoring: actuators, drives, machinery, plants, sensors, and fault-tolerant systems*. Springer Science & Business Media, 2011 (cit. on p. 16).

[25] Florian Sell-Le Blanc, Janna Hofmann, Rico Simmler and Juergen Fleischer. 'Coil winding process modelling with deformation based wire tension analysis'. In: *CIRP Annals* 65.1 (2016), pp. 65–68 (cit. on p. 16).

[26] SA Tsirkas, Paraskevas Papanikos and Th Kermanidis. 'Numerical simulation of the laser welding process in butt-joint specimens'. In: *Journal of materials processing technology* 134.1 (2003), pp. 59–69 (cit. on p. 16).

[27] Quentin Sirvin, Vincent Velay, Rébecca Bonnaire and Luc Penazzi. 'Mechanical behaviour modelling and finite element simulation of simple part of Ti-6Al-4V sheet under hot/warm stamping conditions'. In: *Journal of Manufacturing Processes* 38 (2019), pp. 472–482 (cit. on p. 16).

[28] Fouzi Harrou, Ying Sun, Amanda S Hering, Muddu Madakyaru et al. *Statistical process monitoring using advanced data-driven and deep learning approaches: theory and practical applications*. Elsevier, 2020 (cit. on pp. 16, 17, 31).

[29] Jie Liu, Youmin Hu, Yan Wang et al. 'An integrated multi-sensor fusion-based deep feature learning approach for rotating machinery diagnosis'. In: *Measurement Science and Technology* 29.5 (2018), p. 055103 (cit. on pp. 17, 95).

[30] Siyu Shao, Stephen McAleer, Ruqiang Yan and Pierre Baldi. 'Highly accurate machine fault diagnosis using deep transfer learning'. In: *IEEE Transactions on Industrial Informatics* 15.4 (2018), pp. 2446–2455 (cit. on pp. 17, 61, 118–121, 124).

[31] Sikai Zhang and Zi-Qiang Lang. 'Orthogonal least squares based fast feature selection for linear classification'. In: *Pattern Recognition* 123 (2022), p. 108419 (cit. on p. 17).

[32] Siheung Kim, Jinwok Chung, Ilsung Hwang et al. 'A rule based approach to network fault and security diagnosis with agent collaboration'. In: *International Conference on AI, Simulation, and Planning in High Autonomy Systems*. Springer. 2004, pp. 597–606 (cit. on p. 17).

[33] Barera Sarwar, Imran Sarwar Bajwa, Shabana Ramzan, Bushra Ramzan and Mubeen Kausar. 'Design and application of fuzzy logic based fire monitoring and warning systems for smart buildings'. In: *Symmetry* 10.11 (2018), p. 615 (cit. on p. 17).

[34] Sylvain Verron, Teodor Tiplica and Abdessamad Kobi. *Fault detection with bayesian network*. 2008 (cit. on p. 17).

[35] Venkat Venkatasubramanian, Raghunathan Rengaswamy, Surya N Kavuri and Kewen Yin. 'A review of process fault detection and diagnosis: Part III: Process history based methods'. In: *Computers & chemical engineering* 27.3 (2003), pp. 327–346 (cit. on p. 18).

[36] Dražen Slišković, Ratko Grbić and Željko Hocenski. 'Multivariate statistical process monitoring'. In: *Tehnički vjesnik* 19.1 (2012), pp. 33–41 (cit. on p. 19).

[37] Peihua Qiu. *Introduction to statistical process control*. CRC press, 2013 (cit. on p. 19).

[38] Cynthia A Lowry and Douglas C Montgomery. 'A review of multivariate control charts'. In: *IIE transactions* 27.6 (1995), pp. 800–810 (cit. on p. 19).

[39] Ioannis Ch Paschalidis and Yin Chen. 'Statistical anomaly detection with sensor networks'. In: *ACM Transactions on Sensor Networks* (*TOSN*) 7.2 (2010), pp. 1–23 (cit. on p. 20).

[40] Rui Zhang, Ping Ji, Dinkar Mylaraswamy, Mani Srivastava and Sadaf Zahedi. 'Cooperative sensor anomaly detection using global information'. In: *Tsinghua Science and Technology* 18.3 (2013), pp. 209–219 (cit. on p. 20).

[41] Miao Xie, Jiankun Hu and Biming Tian. 'Histogram-based online anomaly detection in hierarchical wireless sensor networks'. In: *2012 IEEE 11th international conference on trust, security and privacy in computing and communications*. IEEE. 2012, pp. 751–759 (cit. on p. 20).

[42] Kobi Cohen and Qing Zhao. 'Active hypothesis testing for anomaly detection'. In: *IEEE Transactions on Information Theory* 61.3 (2015), pp. 1432–1450 (cit. on p. 20).

[43] Laura Erhan, M Ndubuaku, Mario Di Mauro et al. 'Smart anomaly detection in sensor systems: A multi-perspective review'. In: *Information Fusion* 67 (2021), pp. 64–79 (cit. on pp. 20, 21).

[44] Venkatesh Rajagopalan and Asok Ray. 'Symbolic time series analysis via wavelet-based partitioning'. In: *Signal processing* 86.11 (2006), pp. 3309–3320 (cit. on p. 20).

[45] Nawaz Mohamudally and Mahejabeen Peermamode-Mohaboob. 'Building an anomaly detection engine (ADE) for Iot smart applications'. In: *Procedia computer science* 134 (2018), pp. 10–17 (cit. on p. 21).

[46] Franco Van Wyk, Yiyang Wang, Anahita Khojandi and Neda Masoud. 'Real-time sensor anomaly detection and identification in automated vehicles'. In: *IEEE Transactions on Intelligent Transportation Systems* 21.3 (2019), pp. 1264–1276 (cit. on pp. 21, 69–71, 84, 86–88, 91, 92).

[47] Karthick Thiyagarajan, Sarath Kodagoda and Linh Van Nguyen. 'Predictive analytics for detecting sensor failure using autoregressive integrated moving average model'. In: *2017 12th IEEE conference on industrial electronics and applications (ICIEA)*. IEEE. 2017, pp. 1926–1931 (cit. on p. 21).

[48] Nawaz Mohamudally. *Introductory Chapter: Time Series Analysis (TSA) for Anomaly Detection in IoT*. IntechOpen, 2018 (cit. on p. 21).

[49] Yuan Luo, Ya Xiao, Long Cheng, Guojun Peng and Danfeng Yao. 'Deep learning-based anomaly detection in cyber-physical systems: Progress

and opportunities'. In: *ACM Computing Surveys (CSUR)* 54.5 (2021), pp. 1–36 (cit. on p. 21).

[50] Olivier Janssens, Viktor Slavkovikj, Bram Vervisch et al. 'Convolutional neural network based fault detection for rotating machinery'. In: *Journal of Sound and Vibration* 377 (2016), pp. 331–345 (cit. on p. 21).

[51] Fatma Ozge Ozkok and Mete Celik. 'Convolutional neural network analysis of recurrence plots for high resolution melting classification'. In: *Computer methods and programs in biomedicine* 207 (2021), p. 106139 (cit. on p. 21).

[52] Abdul Rehman Javed, Muhammad Usman, Saif Ur Rehman, Mohib Ullah Khan and Mohammad Sayad Haghighi. 'Anomaly detection in automated vehicles using multistage attention-based convolutional neural network'. In: *IEEE Transactions on Intelligent Transportation Systems* 22.7 (2020), pp. 4291–4300 (cit. on pp. 21, 69–71, 84, 86–88, 92, 93).

[53] Tiago Zonta, Cristiano André da Costa, Felipe A Zeiser et al. 'A predictive maintenance model for optimizing production schedule using deep neural networks'. In: *Journal of Manufacturing Systems* 62 (2022), pp. 450–462 (cit. on p. 22).

[54] Mohsen Ferdowsi, Andrea Benigni, Antonello Monti and Ferdinanda Ponci. 'Measurement Selection for Data-Driven Monitoring of Distribution Systems'. In: *IEEE Systems Journal* 13.4 (2019), pp. 4260–4268 (cit. on p. 22).

[55] FE White Jr. 'Joint directors of laboratories data fusion subpanel report'. In: *Proceedings of the joint service data fusion symposium, DFS–90*. 1990, pp. 496–484 (cit. on p. 22).

[56] Wilfried Elmenreich. 'An introduction to sensor fusion'. In: *Vienna University of Technology, Austria* 502 (2002), pp. 1–28 (cit. on p. 22).

[57] De Jong Yeong, Gustavo Velasco-Hernandez, John Barry and Joseph Walsh. 'Sensor and sensor fusion technology in autonomous vehicles: A review'. In: *Sensors* 21.6 (2021), p. 2140 (cit. on p. 23).

[58] Seungeun Chung, Jiyoun Lim, Kyoung Ju Noh, Gague Kim and Hyuntae Jeong. 'Sensor data acquisition and multimodal sensor fusion for human activity recognition using deep learning'. In: *Sensors* 19.7 (2019), p. 1716 (cit. on p. 23).

[59] Mary B Alatise and Gerhard P Hancke. 'A review on challenges of autonomous mobile robot and sensor fusion methods'. In: *IEEE Access* 8 (2020), pp. 39830–39846 (cit. on p. 23).

[60] Mustafa Kuntoğlu and Hacı Sağlam. 'Investigation of signal behaviors for sensor fusion with tool condition monitoring system in turning'. In: *Measurement* 173 (2021), p. 108582 (cit. on p. 23).

[61] Hugh F Durrant-Whyte. 'Sensor models and multisensor integration'. In: *Autonomous robot vehicles*. Springer, 1990, pp. 73–89 (cit. on p. 23).

[62] Pietro Maris Ferreira, Martin Schaeffer, Adel Mezaour et al. 'A- 40 to 250° C Triple Modular Redundancy Temperature Sensor for Turbofan Engines'. In: *2018 31st Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE. 2018, pp. 1–6 (cit. on p. 23).

[63] Joel R. Sklaroff. 'Redundancy management technique for space shuttle computers'. In: *IBM Journal of Research and Development* 20.1 (1976), pp. 20–28 (cit. on p. 23).

[64] Ihn-Sik Weon, Soon-Geul Lee and Jae-Kwan Ryu. 'Object Recognition based interpolation with 3d lidar and vision for autonomous driving of an intelligent vehicle'. In: *IEEE Access* 8 (2020), pp. 65599–65608 (cit. on p. 24).

[65] César Debeunne and Damien Vivet. 'A review of visual-LiDAR fusion based simultaneous localization and mapping'. In: *Sensors* 20.7 (2020), p. 2068 (cit. on p. 24).

[66] Belur V Dasarathy. 'Sensor fusion potential exploitation-innovative architectures and illustrative applications'. In: *Proceedings of the IEEE* 85.1 (1997), pp. 24–38 (cit. on pp. 25–27).

[67] Billy Pik Lik Lau, Sumudu Hasala Marakkalage, Yuren Zhou et al. 'A survey of data fusion in smart city applications'. In: *Information Fusion* 52 (2019), pp. 357–374 (cit. on p. 26).

[68] Yu Zheng, Furui Liu and Hsun-Ping Hsieh. 'U-air: When urban air quality inference meets big data'. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013, pp. 1436–1444 (cit. on p. 26).

[69] Farman Ali, Shaker El-Sappagh, SM Riazul Islam et al. 'A smart healthcare monitoring system for heart disease prediction based on ensemble deep learning and feature fusion'. In: *Information Fusion* 63 (2020), pp. 208–222 (cit. on p. 26).

[70] Ugochukwu Ejike Akpudo and Hur Jang-Wook. 'An Automated Sensor Fusion Approach for the RUL Prediction of Electromagnetic Pumps'. In: *IEEE Access* 9 (2021), pp. 38920–38933 (cit. on p. 26).

[71] Jun Wu, Yongheng Su, Yiwei Cheng et al. 'Multi-sensor information fusion for remaining useful life prediction of machining tools by adaptive network based fuzzy inference system'. In: *Applied Soft Computing* 68 (2018), pp. 13–23 (cit. on p. 26).

[72] Anna Stief, James R Ottewill, Jerzy Baranowski and Michal Orkisz. 'A PCA and two-stage Bayesian sensor fusion approach for diagnosing electrical and mechanical faults in induction motors'. In: *IEEE Transactions on Industrial Electronics* 66.12 (2019), pp. 9510–9520 (cit. on pp. 26, 125).

[73] Essam Debie, Raul Fernandez Rojas, Justin Fidock et al. 'Multimodal fusion for objective assessment of cognitive workload: a review'. In: *IEEE transactions on cybernetics* 51.3 (2019), pp. 1542–1555 (cit. on p. 27).

[74] Rustem Dautov, Salvatore Distefano and Rajkumaar Buyya. 'Hierarchical data fusion for smart healthcare'. In: *Journal of Big Data* 6.1 (2019), pp. 1–23 (cit. on p. 27).

[75] Athanasios C Rakitzis, Subhabrata Chakraborti, Sandile C Shongwe, Marien A Graham and Michael Boon Chong Khoo. 'An overview of synthetic-type control charts: techniques and methodology'. In: *Quality and Reliability Engineering International* 35.7 (2019), pp. 2081–2096 (cit. on p. 28).

[76] Ewan S Page. 'Continuous inspection schemes'. In: *Biometrika* 41.1/2 (1954), pp. 100–115 (cit. on p. 29).

[77] SW Roberts. 'Control chart tests based on geometric moving averages'. In: *Technometrics* 42.1 (2000), pp. 97–101 (cit. on p. 29).

[78] Margarita Beneke, Lawrence M Leemis, Robert E Schlegel and Bobbie L Foote. 'Spectral analysis in quality control: a control chart based on the periodogram'. In: *Technometrics* 30.1 (1988), pp. 63–70 (cit. on p. 30).

[79] Teodor Tiplica, Abdessamad Kobi and Alain Barreau. 'Spectral control chart'. In: *Quality Engineering* 17.4 (2005), pp. 695–702 (cit. on p. 30).

[80] Rajesh Ganesan, Tapas K Das and Vivekanand Venkataraman. 'Wavelet-based multiscale statistical process monitoring: A literature review'. In: *IIE transactions* 36.9 (2004), pp. 787–806 (cit. on p. 30).

[81] Reza Baradaran Kazemzadeh, Mahdi Karbasian and Mohammad Ali Babakhani. 'An EWMA t chart with variable sampling intervals for monitoring the process mean'. In: *The International Journal of Advanced Manufacturing Technology* 66.1 (2013), pp. 125–139 (cit. on p. 30).

[82] Yan Su, Lianjie Shu and Kwok-Leung Tsui. 'Adaptive EWMA procedures for monitoring processes subject to linear drifts'. In: *Computational statistics & data analysis* 55.10 (2011), pp. 2819–2829 (cit. on p. 30).

[83] Abdul Haq and Michael BC Khoo. 'An adaptive multivariate EWMA chart'. In: *Computers & Industrial Engineering* 127 (2019), pp. 549–557 (cit. on p. 30).

[84] Abdul Haq, Tahir Munir and Michael BC Khoo. 'Dual multivariate CUSUM mean charts'. In: *Computers & Industrial Engineering* 137 (2019), p. 106028 (cit. on p. 30).

[85] Vahid Nasir and Farrokh Sassani. 'A review on deep learning in machining and tool monitoring: methods, opportunities, and challenges'. In: *The International Journal of Advanced Manufacturing Technology* 115.9 (2021), pp. 2683–2709 (cit. on p. 30).

[86] Jinjiang Wang, Yulin Ma, Laibin Zhang, Robert X Gao and Dazhong Wu. 'Deep learning for smart manufacturing: Methods and applications'. In: *Journal of manufacturing systems* 48 (2018), pp. 144–156 (cit. on p. 30).

[87] Thanasis Kotsiopoulos, Panagiotis Sarigiannidis, Dimosthenis Ioannidis and Dimitrios Tzovaras. 'Machine learning and deep learning in smart manufacturing: The smart grid paradigm'. In: *Computer Science Review* 40 (2021), p. 100341 (cit. on p. 30).

[88] Juan Jose Saucedo-Dorantes, Miguel Delgado-Prieto, Roque Alfredo Osornio-Rios and Rene de Jesus Romero-Troncoso. 'Multifault diagnosis method applied to an electric machine based on high-dimensional feature reduction'. In: *IEEE Transactions on industry applications* 53.3 (2016), pp. 3086–3097 (cit. on pp. 31, 32).

[89] Rodrigo Henrique Cunha Palácios, Alessandro Goedtel, Wagner Fontes Godoy and José Augusto Fabri. 'Fault identification in the stator winding of induction motors using PCA with artificial neural networks'. In: *Journal of Control, Automation and Electrical Systems* 27.4 (2016), pp. 406–418 (cit. on p. 32).

[90] Heng Wang, Guangxian Ni, Jinhai Chen and Jiangming Qu. 'Research on rolling bearing state health monitoring and life prediction based on PCA and Internet of things with multi-sensor'. In: *Measurement* 157 (2020), p. 107657 (cit. on p. 32).

[91] Theodoros Loutas, Nick Eleftheroglou, George Georgoulas et al. 'Valve failure prognostics in reciprocating compressors utilizing temperature measurements, PCA-based data fusion, and probabilistic algorithms'. In: *IEEE Transactions on Industrial Electronics* 67.6 (2019), pp. 5022–5029 (cit. on p. 32).

[92] Anna Stief, James R Ottewill, Michal Orkisz and Jerzy Baranowski. `Two stage data fusion of acoustic, electric and vibration signals for diagnosing faults in induction motors'. In: *Elektronika ir Elektrotechnika* 23.6 (2017), pp. 19–24 (cit. on pp. 32, 34).

[93] Jinjiang Wang, Junyao Xie, Rui Zhao, Laibin Zhang and Lixiang Duan. `Multisensory fusion based virtual tool wear sensing for ubiquitous manufacturing'. In: *Robotics and computer-integrated manufacturing* 45 (2017), pp. 47–58 (cit. on p. 32).

[94] Wei Li, Minjun Peng and Qingzhong Wang. `Fault detectability analysis in PCA method during condition monitoring of sensors in a nuclear power plant'. In: *Annals of Nuclear Energy* 119 (2018), pp. 342–351 (cit. on p. 32).

[95] Lijia Luo, Shiyi Bao and Chudong Tong. `Sparse robust principal component analysis with applications to fault detection and diagnosis'. In: *Industrial & Engineering Chemistry Research* 58.3 (2019), pp. 1300–1309 (cit. on p. 32).

[96] Jinjiang Wang, Junyao Xie, Rui Zhao, Laibin Zhang and Lixiang Duan. `Multisensory fusion based virtual tool wear sensing for ubiquitous manufacturing'. In: *Robotics and computer-integrated manufacturing* 45 (2017), pp. 47–58 (cit. on p. 32).

[97] Dongsoon Kim and In-Beum Lee. `Process monitoring based on probabilistic PCA'. In: *Chemometrics and intelligent laboratory systems* 67.2 (2003), pp. 109–123 (cit. on p. 32).

[98] Jyh-Cheng Jeng. `Adaptive process monitoring using efficient recursive PCA and moving window PCA algorithms'. In: *Journal of the Taiwan Institute of Chemical Engineers* 41.4 (2010), pp. 475–481 (cit. on p. 32).

[99] Tauheed Mian, Anurag Choudhary and Shahab Fatima. `A sensor fusion based approach for bearing fault diagnosis of rotating machine'. In: *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability* 236.5 (2022), pp. 661–675 (cit. on p. 32).

[100] U Kruger, Q Chen, DJ Sandoz and RC McFarlane. 'Extended PLS approach for enhanced condition monitoring of industrial processes'. In: *AIChE journal* 47.9 (2001), pp. 2076–2091 (cit. on p. 32).

[101] Chudong Tong, Ahmet Palazoglu and Xuefeng Yan. 'Improved ICA for process monitoring based on ensemble learning and Bayesian inference'. In: *Chemometrics and intelligent laboratory systems* 135 (2014), pp. 141–149 (cit. on p. 32).

[102] Víctor H Jaramillo, James R Ottewill, Rafał Dudek, Dariusz Lepiarczyk and Paweł Pawlik. 'Condition monitoring of distributed systems using two-stage Bayesian inference data fusion'. In: *Mechanical Systems and Signal Processing* 87 (2017), pp. 91–110 (cit. on pp. 33, 34).

[103] Anna Stief, James R Ottewill, Jerzy Baranowski and Michal Orkisz. 'A PCA and two-stage Bayesian sensor fusion approach for diagnosing electrical and mechanical faults in induction motors'. In: *IEEE Transactions on Industrial Electronics* 66.12 (2019), pp. 9510–9520 (cit. on p. 33).

[104] Ferat Sahin, M Çetin Yavuz, Ziya Arnavut and Önder Uluyol. 'Fault diagnosis for airplane engines using Bayesian networks and distributed particle swarm optimization'. In: *Parallel Computing* 33.2 (2007), pp. 124–143 (cit. on p. 34).

[105] Kari Sentz and Scott Ferson. 'Combination of evidence in Dempster-Shafer theory'. In: (2002) (cit. on p. 34).

[106] Arthur P Dempster. 'A generalization of Bayesian inference'. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 30.2 (1968), pp. 205–232 (cit. on p. 34).

[107] Glenn Shafer. *A mathematical theory of evidence*. Vol. 42. Princeton university press, 1976 (cit. on p. 34).

[108] Wentao Zhao, Tao Fang and Yan Jiang. 'Data fusion using improved Dempster Shafer evidence theory for vehicle detection'. In: *Fourth Inter-*

national Conference on Fuzzy Systems and Knowledge Discovery (*FSKD 2007*). Vol. 1. IEEE. 2007, pp. 487–491 (cit. on p. 35).

[109] Arnaud Martin and Isabelle Quidu. 'Decision support with belief functions theory for seabed characterization'. In: *2008 11th International Conference on Information Fusion*. IEEE. 2008, pp. 1–8 (cit. on p. 36).

[110] Yuwei Liu, Yuqiang Cheng, Zhenzhen Zhang and Jianjun Wu. 'Multi information fusion fault diagnosis based on KNN and improved evidence theory'. In: *Journal of Vibration Engineering & Technologies* 10.3 (2022), pp. 841–852 (cit. on p. 37).

[111] Yingjie Zhang, Wentao Yan, Geok Soon Hong et al. 'Data fusion analysis in the powder-bed fusion AM process monitoring by Dempster-Shafer evidence theory'. In: *Rapid Prototyping Journal* (2021) (cit. on p. 37).

[112] Hepeng Zhang and Yong Deng. 'Engine fault diagnosis based on sensor data fusion considering information quality and evidence theory'. In: *Advances in Mechanical Engineering* 10.11 (2018), p. 1687814018809184 (cit. on p. 37).

[113] Xianghong Tang, Xin Gu, Lei Rao and Jianguang Lu. 'A single fault detection method of gearbox based on random forest hybrid classifier and improved Dempster-Shafer information fusion'. In: *Computers & Electrical Engineering* 92 (2021), p. 107101 (cit. on p. 37).

[114] Wen Jiang and Jun Zhan. 'A modified combination rule in generalized evidence theory'. In: *Applied Intelligence* 46.3 (2017), pp. 630–640 (cit. on p. 37).

[115] Lotfi A Zadeh. 'Fuzzy sets'. In: *Information and control* 8.3 (1965), pp. 338–353 (cit. on p. 37).

[116] Zhiqiang Ge and Yue Liu. 'Analytic hierarchy process based fuzzy decision fusion system for model prioritization and process monitoring application'. In: *IEEE Transactions on Industrial Informatics* 15.1 (2018), pp. 357–365 (cit. on p. 38).

[117] Mustapha Ammiche, Abdelmalek Kouadri and Abderazak Bensmail. 'A modified moving window dynamic PCA with fuzzy logic filter and application to fault detection'. In: *Chemometrics and Intelligent Laboratory Systems* 177 (2018), pp. 100–113 (cit. on p. 38).

[118] MRH Mohd Adnan, Arezoo Sarkheyli, Azlan Mohd Zain and Habibollah Haron. 'Fuzzy logic for modeling machining process: a review'. In: *Artificial Intelligence Review* 43.3 (2015), pp. 345–379 (cit. on pp. 38, 39).

[119] Mustafa Kuntoğlu and Hacı Sağlam. 'Investigation of signal behaviors for sensor fusion with tool condition monitoring system in turning'. In: *Measurement* 173 (2021), p. 108582 (cit. on pp. 38, 39).

[120] Mohammad Amin Ahmad Akhoundi and Ehsan Valavi. 'Multi-sensor fuzzy data fusion using sensors with different characteristics'. In: *arXiv preprint arXiv:1010.6096* (2010) (cit. on p. 39).

[121] Warren S McCulloch and Walter Pitts. 'A logical calculus of the ideas immanent in nervous activity'. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133 (cit. on p. 41).

[122] Sagar Sharma, Simone Sharma and Anidhya Athaiya. 'Activation functions in neural networks'. In: *towards data science* 6.12 (2017), pp. 310–316 (cit. on p. 42).

[123] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep learning*. MIT press, 2016 (cit. on pp. 42, 43, 45, 58, 74).

[124] Kurt Hornik, Maxwell Stinchcombe and Halbert White. 'Multilayer feedforward networks are universal approximators'. In: *Neural networks* 2.5 (1989), pp. 359–366 (cit. on p. 42).

[125] J Rafiee, F Arvani, A Harifi and MH Sadeghi. 'Intelligent condition monitoring of a gearbox using artificial neural network'. In: *Mechanical systems and signal processing* 21.4 (2007), pp. 1746–1754 (cit. on p. 43).

[126] Aydin Salimiasl and Ahmet Özdemir. `Analyzing the performance of artificial neural network (ANN)-, fuzzy logic (FL)-, and least square (LS)-based models for online tool condition monitoring'. In: *The International Journal of Advanced Manufacturing Technology* 87.1 (2016), pp. 1145–1158 (cit. on p. 43).

[127] Zhigang Tian. `An artificial neural network method for remaining useful life prediction of equipment subject to condition monitoring'. In: *Journal of intelligent Manufacturing* 23.2 (2012), pp. 227–237 (cit. on p. 44).

[128] K Patra, AK Jha, Tibor Szalay, J Ranjan and László Monostori. `Artificial neural network based tool condition monitoring in micro mechanical peck drilling using thrust force signals'. In: *Precision Engineering* 48 (2017), pp. 279–291 (cit. on p. 44).

[129] Abhinav Saxena and Ashraf Saad. `Evolving an artificial neural network classifier for condition monitoring of rotating mechanical systems'. In: *Applied Soft Computing* 7.1 (2007), pp. 441–454 (cit. on p. 44).

[130] Jatin Prakash and Pavan Kumar Kankar. `Health prediction of hydraulic cooling circuit using deep neural network with ensemble feature ranking technique'. In: *Measurement* 151 (2020), p. 107225 (cit. on pp. 44, 59, 108, 115, 117, 139).

[131] Yann LeCun et al. `Generalization and network design strategies'. In: *Connectionism in perspective* 19.143-155 (1989), p. 18 (cit. on p. 44).

[132] Dulari Bhatt, Chirag Patel, Hardik Talsania et al. `CNN variants for computer vision: History, architecture, application, challenges and future scope'. In: *Electronics* 10.20 (2021), p. 2470 (cit. on p. 46).

[133] Tanvir Alam Shifat and Jang Wook Hur. `An effective stator fault diagnosis framework of BLDC motor based on vibration and current signals'. In: *IEEE Access* 8 (2020), pp. 106968–106981 (cit. on p. 46).

[134] Chandrabhanu Malla and Isham Panigrahi. ‘Review of condition monitoring of rolling element bearing using vibration analysis and other techniques’. In: *Journal of Vibration Engineering & Technologies* 7.4 (2019), pp. 407–414 (cit. on p. 46).

[135] Ke Feng, JC Ji, Qing Ni and Michael Beer. ‘A review of vibration-based gear wear monitoring and prediction techniques’. In: *Mechanical Systems and Signal Processing* 182 (2023), p. 109605 (cit. on p. 46).

[136] Gokberk Serin, Batihan Sener, A Murat Ozbayoglu and Hakki Ozgur Unver. ‘Review of tool condition monitoring in machining and opportunities for deep learning’. In: *The International Journal of Advanced Manufacturing Technology* 109.3 (2020), pp. 953–974 (cit. on p. 46).

[137] Galipothu Dheeraj Simon and R Deivanathan. ‘Early detection of drilling tool wear by vibration data acquisition and classification’. In: *Manufacturing Letters* 21 (2019), pp. 60–65 (cit. on p. 46).

[138] K Balachandar, R Jegadeeshwaran, J Lakshmipathi and D Saravanakumar. ‘Friction Stir Welding Tool Condition Prediction Using Vibrational Analysis Through Machine Learning–A Review’. In: *Journal of Physics: Conference Series*. Vol. 1969. 1. IOP Publishing. 2021, p. 012051 (cit. on p. 46).

[139] Luyang Jing, Ming Zhao, Pin Li and Xiaoqiang Xu. ‘A convolutional neural network based feature learning and fault diagnosis method for the condition monitoring of gearbox’. In: *Measurement* 111 (2017), pp. 1–10 (cit. on p. 46).

[140] Minghang Zhao, Shisheng Zhong, Xuyun Fu, Baoping Tang and Michael Pecht. ‘Deep residual shrinkage networks for fault diagnosis’. In: *IEEE Transactions on Industrial Informatics* 16.7 (2019), pp. 4681–4690 (cit. on p. 46).

[141] Keke Huang, Shujie Wu, Fanbiao Li, Chunhua Yang and Weihua Gui. ‘Fault diagnosis of hydraulic systems based on deep learning model with multirate data samples’. In: *IEEE Transactions on neural networks and learning systems* (2021) (cit. on pp. 46, 47, 50, 95, 108, 115–117, 128, 139).

[142] Can Cheng, Jianyong Li, Yueming Liu, Meng Nie and Wenxi Wang. 'Deep convolutional neural network-based in-process tool condition monitoring in abrasive belt grinding'. In: *Computers in Industry* 106 (2019), pp. 1–13 (cit. on p. 46).

[143] Yu Liang, Binbin Li and Bin Jiao. 'A deep learning method for motor fault diagnosis based on a capsule network with gate-structure dilated convolutions'. In: *Neural Computing and Applications* 33.5 (2021), pp. 1401–1418 (cit. on p. 46).

[144] David E Rumelhart, Geoffrey E Hinton and Ronald J Williams. 'Learning representations by back-propagating errors'. In: *nature* 323.6088 (1986), pp. 533–536 (cit. on p. 47).

[145] Yoshua Bengio, Patrice Simard and Paolo Frasconi. 'Learning long-term dependencies with gradient descent is difficult'. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166 (cit. on p. 49).

[146] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre et al. 'Learning phrase representations using RNN encoder-decoder for statistical machine translation'. In: *arXiv preprint arXiv:1406.1078* (2014) (cit. on p. 49).

[147] Sepp Hochreiter and Jürgen Schmidhuber. 'Long short-term memory'. In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 49).

[148] Felix O Heimes. 'Recurrent neural networks for remaining useful life estimation'. In: *2008 international conference on prognostics and health management*. IEEE. 2008, pp. 1–6 (cit. on p. 49).

[149] Wennian Yu, Il Yong Kim and Chris Mechefske. 'An improved similarity-based prognostic algorithm for RUL estimation using an RNN autoencoder scheme'. In: *Reliability Engineering & System Safety* 199 (2020), p. 106926 (cit. on pp. 50, 51, 93, 145).

[150] Yingchao Liu, Xiaofeng Hu and Wenjuan Zhang. 'Remaining useful life prediction based on health index similarity'. In: *Reliability Engineering & System Safety* 185 (2019), pp. 502–510 (cit. on p. 50).

[151] Tarak Benkedjouh, Kamal Medjaher, Noureddine Zerhouni and Said Rechak. ‘Remaining useful life estimation based on nonlinear feature reduction and support vector regression’. In: *Engineering Applications of Artificial Intelligence* 26.7 (2013), pp. 1751–1760 (cit. on p. 50).

[152] Jihong Yan, Muammer Koc and Jay Lee. ‘A prognostic algorithm for machine performance assessment and its application’. In: *Production Planning & Control* 15.8 (2004), pp. 796–801 (cit. on p. 50).

[153] Tianyi Wang, Jianbo Yu, David Siegel and Jay Lee. ‘A similarity-based prognostics approach for remaining useful life estimation of engineered systems’. In: *2008 international conference on prognostics and health management*. IEEE. 2008, pp. 1–6 (cit. on p. 50).

[154] Racha Khelif, Simon Malinowski, Brigitte Chebel-Morello and Noureddine Zerhouni. ‘RUL prediction based on a new similarity-instance based approach’. In: *2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE)*. IEEE. 2014, pp. 2463–2468 (cit. on p. 50).

[155] Narendhar Gugulothu, Vishnu Tv, Pankaj Malhotra et al. ‘Predicting remaining useful life using time series embeddings based on recurrent neural networks’. In: *arXiv preprint arXiv:1709.01073* (2017) (cit. on p. 50).

[156] Ugochukwu Ejike Akpudo and Hur Jang-Wook. ‘An Automated Sensor Fusion Approach for the RUL Prediction of Electromagnetic Pumps’. In: *IEEE Access* 9 (2021), pp. 38920–38933 (cit. on p. 50).

[157] Kwok Tai Chui, Brij B Gupta and Pandian Vasant. ‘A genetic algorithm optimized RNN-LSTM model for remaining useful life prediction of turbofan engine’. In: *Electronics* 10.3 (2021), p. 285 (cit. on p. 50).

[158] Wennian Yu, Il Yong Kim and Chris Mechefske. ‘Analysis of different RNN autoencoder variants for time series classification and machine prognostics’. In: *Mechanical Systems and Signal Processing* 149 (2021), p. 107322 (cit. on pp. 51, 93, 145).

[159] Volodymyr Mnih, Nicolas Heess, Alex Graves and Koray Kavukcuoglu. 'Recurrent models of visual attention'. In: *arXiv preprint arXiv:1406.6247* (2014) (cit. on p. 51).

[160] Zeyu Cheng, Yi Zhang and Chengkai Tang. 'Swin-Depth: Using Transformers and Multi-Scale Fusion for Monocular-Based Depth Estimation'. In: *IEEE Sensors Journal* 21.23 (2021), pp. 26912–26920 (cit. on p. 51).

[161] Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio. 'Neural machine translation by jointly learning to align and translate'. In: *arXiv preprint arXiv:1409.0473* (2014) (cit. on p. 51).

[162] Ashish Vaswani, Noam Shazeer, Niki Parmar et al. 'Attention is all you need'. In: *Advances in neural information processing systems* 30 (2017) (cit. on pp. 51, 53, 68, 77, 79, 80, 96, 98, 129, 139).

[163] Han Hu, Zheng Zhang, Zhenda Xie and Stephen Lin. 'Local relation networks for image recognition'. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 3464–3473 (cit. on pp. 53, 127).

[164] Prajit Ramachandran, Niki Parmar, Ashish Vaswani et al. 'Stand-alone self-attention in vision models'. In: *Advances in neural information processing systems* 32 (2019) (cit. on pp. 53, 127).

[165] OpenAI. *GPT-3*. https://openai.com/blog/gpt-3-apps/ (cit. on pp. 53, 127, 145).

[166] OpenAI. *ChatGPT*. https://openai.com/blog/chatgpt/ (cit. on p. 53).

[167] Anurag Arnab, Mostafa Dehghani, Georg Heigold et al. 'Vivit: A video vision transformer'. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 6836–6846 (cit. on p. 53).

[168] DeepMind. *Alphafold*. https://www.deepmind.com/research/highlighted-research/alphafold (cit. on p. 53).

[169] Yuhong Jin, Lei Hou and Yushu Chen. 'A new rotating machinery fault diagnosis method based on the Time Series Transformer'. In: *arXiv preprint arXiv:2108.12562* (2021) (cit. on p. 53).

[170] Case Western Reserve University. *Bearing Fault dataset*. https://engineering.case.edu/bearingdatacenter (cit. on p. 54).

[171] Bingjie Wu, Wenjian Cai, Fanyong Cheng and Haoran Chen. 'Simultaneous-fault diagnosis considering time series with a deep learning transformer architecture for air handling units'. In: *Energy and Buildings* 257 (2022), p. 111608 (cit. on p. 54).

[172] Xinglong Pei, Xiaoyang Zheng and Jinliang Wu. 'Rotating machinery fault diagnosis through a transformer convolution network subjected to transfer learning'. In: *IEEE Transactions on Instrumentation and Measurement* 70 (2021), pp. 1–11 (cit. on pp. 54, 55, 115, 122).

[173] Hui Wang, Jiawen Xu, Ruqiang Yan, Chuang Sun and Xuefeng Chen. 'Intelligent bearing fault diagnosis using multi-head attention-based CNN'. In: *Procedia Manufacturing* 49 (2020), pp. 112–118 (cit. on p. 54).

[174] Hairui Fang, Jin Deng, Bo Zhao et al. 'LEFE-Net: A lightweight efficient feature extraction network with strong robustness for bearing fault diagnosis'. In: *IEEE Transactions on Instrumentation and Measurement* 70 (2021), pp. 1–11 (cit. on p. 54).

[175] Zhiwen Chen, Ketian Liang, Steven X Ding et al. 'A comparative study of deep neural network-aided canonical correlation analysis-based process monitoring and fault detection methods'. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.11 (2021), pp. 6158–6172 (cit. on p. 55).

[176] Aneesh G Nath, Sandeep S Udmale, Divyanshu Raghuwanshi and Sanjay Kumar Singh. 'Structural rotor fault diagnosis using attention-based sensor fusion and transformers'. In: *IEEE Sensors Journal* 22.1 (2021), pp. 707–719 (cit. on p. 55).

[177] Xavier Glorot and Yoshua Bengio. 'Understanding the difficulty of training deep feedforward neural networks'. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256 (cit. on p. 56).

[178] Yoshua Bengio, Patrice Simard and Paolo Frasconi. 'Learning long-term dependencies with gradient descent is difficult'. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166 (cit. on p. 56).

[179] Matías Roodschild, Jorge Gotay Sardiñas and Adrián Will. 'A new approach for the vanishing gradient problem on sigmoid activation'. In: *Progress in Artificial Intelligence* 9.4 (2020), pp. 351–360 (cit. on p. 56).

[180] Meenal V Narkhede, Prashant P Bartakke and Mukul S Sutaone. 'A review on weight initialization strategies for neural networks'. In: *Artificial intelligence review* 55.1 (2022), pp. 291–322 (cit. on p. 56).

[181] Jingzhao Zhang, Tianxing He, Suvrit Sra and Ali Jadbabaie. 'Why gradient clipping accelerates training: A theoretical justification for adaptivity'. In: *arXiv preprint arXiv:1905.11881* (2019) (cit. on p. 56).

[182] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas and Aleksander Madry. 'How does batch normalization help optimization?' In: *Advances in neural information processing systems* 31 (2018) (cit. on pp. 57, 76).

[183] Jimmy Lei Ba, Jamie Ryan Kiros and Geoffrey E Hinton. 'Layer normalization'. In: *arXiv preprint arXiv:1607.06450* (2016) (cit. on pp. 57, 76, 134).

[184] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. 'Deep residual learning for image recognition'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cit. on pp. 57, 58, 134).

[185] Imanol Bilbao and Javier Bilbao. 'Overfitting problem and the over-training in the era of data: Particularly for Artificial Neural Networks'. In: *2017 eighth international conference on intelligent computing and information systems (ICICIS)*. IEEE. 2017, pp. 173–177 (cit. on p. 57).

[186] Pierre Baldi and Peter J Sadowski. ‘Understanding dropout’. In: *Advances in neural information processing systems* 26 (2013) (cit. on p. 57).

[187] Guodong Zhang, Chaoqi Wang, Bowen Xu and Roger Grosse. ‘Three mechanisms of weight decay regularization’. In: *arXiv preprint arXiv: 1810. 12281* (2018) (cit. on p. 57).

[188] Lutz Prechelt. ‘Early stopping—but when?’ In: *Neural networks: tricks of the trade: second edition* (2012), pp. 53–67 (cit. on p. 57).

[189] Kaiming He and Jian Sun. ‘Convolutional neural networks at constrained time cost’. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 5353–5360 (cit. on p. 58).

[190] Rupesh Kumar Srivastava, Klaus Greff and Jürgen Schmidhuber. ‘Highway networks’. In: *arXiv preprint arXiv:1505.00387* (2015) (cit. on p. 58).

[191] The UniProt Consortium. *UniProt: the Universal Protein Knowledgebase in 2023*. https://www.uniprot.org/ (cit. on p. 59).

[192] Kevin Lu, Aditya Grover, Pieter Abbeel and Igor Mordatch. ‘Pretrained transformers as universal computation engines’. In: *arXiv preprint arXiv: 2103. 05247* 1 (2021) (cit. on pp. 59, 83, 96, 103).

[193] Shizhao Sun, Wei Chen, Liwei Wang, Xiaoguang Liu and Tie-Yan Liu. ‘On the depth of deep neural networks: A theoretical view’. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016 (cit. on p. 59).

[194] Zhe Li, Jingyue Li, Yi Wang and Kesheng Wang. ‘A deep learning approach for anomaly detection based on SAE and LSTM in mechanical equipment’. In: *The International Journal of Advanced Manufacturing Technology* 103 (2019), pp. 499–510 (cit. on p. 59).

[195] Caroline König and Ahmed Mohamed Helmi. ‘Sensitivity analysis of sensors in a hydraulic condition monitoring system using CNN models’. In: *Sensors* 20.11 (2020), p. 3307 (cit. on p. 59).

[196] Chuang Sun, Meng Ma, Zhibin Zhao et al. 'Deep transfer learning based on sparse autoencoder for remaining useful life prediction of tool in manufacturing'. In: *IEEE transactions on industrial informatics* 15.4 (2018), 2416–2425 (cit. on p. 60).

[197] Wentao Mao, Jianliang He and Ming J Zuo. 'Predicting remaining useful life of rolling bearings based on deep feature representation and transfer learning'. In: *IEEE Transactions on Instrumentation and Measurement* 69.4 (2019), pp. 1594–1608 (cit. on p. 60).

[198] Princeton University Stanford Vision Lab Stanford University. *ImageNet*. `https://www.image-net.org/index.php` (cit. on p. 61).

[199] Pei Cao, Shengli Zhang and Jiong Tang. 'Preprocessing-free gear fault diagnosis using small datasets with deep convolutional neural network-based transfer learning'. In: *Ieee Access* 6 (2018), pp. 26241–26253 (cit. on p. 61).

[200] Chunlei Chen, Peng Zhang, Huixiang Zhang et al. 'Deep learning on computational-resource-limited platforms: a survey'. In: *Mobile Information Systems* 2020 (2020), pp. 1–19 (cit. on p. 62).

[201] Siemens. *SIMATIC IPC*. `https://new.siemens.com/global/en/products/automation/pc-based/simatic-rack-ipc.html` (cit. on p. 62).

[202] Manabu Kano and Yoshiaki Nakagawa. 'Data-based process monitoring, process control, and quality improvement: Recent developments and applications in steel industry'. In: *Computers & Chemical Engineering* 32.1-2 (2008), pp. 12–24 (cit. on p. 62).

[203] Yatao Yang, Runze Yang, Longhui Pan et al. 'A lightweight deep learning algorithm for inspection of laser welding defects on safety vent of power battery'. In: *Computers in industry* 123 (2020), p. 103306 (cit. on p. 62).

[204] Edward A Lee. 'The past, present and future of cyber-physical systems: A focus on models'. In: *Sensors* 15.3 (2015), pp. 4837–4869 (cit. on p. 67).

[205] Sutharshan Rajasegarar, Christopher Leckie and Marimuthu Palaniswami. 'Anomaly detection in wireless sensor networks'. In: *IEEE Wireless Communications* 15.4 (2008), pp. 34–40 (cit. on p. 68).

[206] Abhishek B Sharma, Leana Golubchik and Ramesh Govindan. 'Sensor faults: Detection methods and prevalence in real-world datasets'. In: *ACM Transactions on Sensor Networks* (*TOSN*) 6.3 (2010), pp. 1–39 (cit. on p. 68).

[207] Monowar H Bhuyan, Dhruba Kumar Bhattacharyya and Jugal K Kalita. 'Network anomaly detection: methods, systems and tools'. In: *Ieee communications surveys & tutorials* 16.1 (2013), pp. 303–336 (cit. on p. 68).

[208] Debby Bezzina and James Sayer. 'Safety pilot model deployment: Test conductor team report'. In: *Report No. DOT HS* 812.171 (2014), p. 18 (cit. on p. 70).

[209] Jonathan Petit and Steven E Shladover. 'Potential cyberattacks on automated vehicles'. In: *IEEE Transactions on Intelligent transportation systems* 16.2 (2014), pp. 546–556 (cit. on pp. 71, 93).

[210] Yilin Mo, Emanuele Garone, Alessandro Casavola and Bruno Sinopoli. 'False data injection attacks against state estimation in wireless sensor networks'. In: *49th IEEE Conference on Decision and Control* (*CDC*). IEEE. 2010, pp. 5967–5972 (cit. on pp. 71, 93).

[211] Zhiguang Wang, Weizhong Yan and Tim Oates. 'Time series classification from scratch with deep neural networks: A strong baseline'. In: *2017 International joint conference on neural networks* (*IJCNN*). IEEE. 2017, pp. 1578–1585 (cit. on p. 74).

[212] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar and Pierre-Alain Muller. 'Deep learning for time series classification: a review'. In: *Data mining and knowledge discovery* 33.4 (2019), pp. 917–963 (cit. on p. 74).

[213] Wensi Tang, Guodong Long, Lu Liu et al. `Rethinking 1d-cnn for time series classification: A stronger baseline'. In: *arXiv preprint arXiv:2002.10061* (2020), pp. 1–7 (cit. on p. 74).

[214] Abien Fred Agarap. `Deep learning using rectified linear units (relu)'. In: *arXiv preprint arXiv:1803.08375* (2018) (cit. on p. 75).

[215] Andrew M Saxe, James L McClelland and Surya Ganguli. `Exact solutions to the nonlinear dynamics of learning in deep linear neural networks'. In: *arXiv preprint arXiv:1312.6120* (2013) (cit. on p. 83).

[216] Yanhong Fei, Yingjie Liu, Xian Wei and Mingsong Chen. `O-vit: Orthogonal vision transformer'. In: *arXiv preprint arXiv:2201.12133* (2022) (cit. on p. 83).

[217] Linzhen Nie, Jiayi Guan, Chihua Lu, Hao Zheng and Zhishuai Yin. `Longitudinal speed control of autonomous vehicle based on a self-adaptive PID of radial basis function neural network'. In: *IET Intelligent Transport Systems* 12.6 (2018), pp. 485–494 (cit. on p. 91).

[218] Qiang Li, Ranyang Li, Kaifan Ji and Wei Dai. `Kalman filter and its application'. In: *2015 8th International Conference on Intelligent Networks and Intelligent Systems* (*ICINIS*). IEEE. 2015, pp. 74–77 (cit. on p. 91).

[219] Lin Jiang, Hang Xu, Jinhai Liu et al. `Anomaly detection of industrial multi-sensor signals based on enhanced spatiotemporal features'. In: *Neural Computing and Applications* 34.11 (2022), pp. 8465–8477 (cit. on p. 94).

[220] Nadav Cohen, Or Sharir and Amnon Shashua. `On the expressive power of deep learning: A tensor analysis'. In: *Conference on learning theory*. PMLR. 2016, pp. 698–728 (cit. on pp. 95, 116).

[221] Hai Qiu, Jay Lee, Jing Lin and Gang Yu. `Wavelet filter-based weak signature detection method and its application on rolling element bearing prognostics'. In: *Journal of sound and vibration* 289.4-5 (2006), pp. 1066–1090 (cit. on pp. 95, 120).

[222] Minghui Cheng, Li Jiao, Pei Yan et al. 'Intelligent tool wear monitoring and multi-step prediction based on deep learning model'. In: *Journal of Manufacturing Systems* 62 (2022), pp. 286–300 (cit. on p. 95).

[223] Alec Radford, Wu Jeffrey, Dario Amodei et al. *Better Language Models and Their Implications.* Website. https://openai.com/blog/better-language-models/ Last access: Nov 2021. 2019 (cit. on pp. 102, 104).

[224] Alec Radford, Jeffrey Wu, Rewon Child et al. 'Language models are unsupervised multitask learners'. In: *OpenAI blog* 1.8 (2019), p. 9 (cit. on p. 102).

[225] Tarek Berghout, Mohamed Benbouzid, SM Muyeen, Toufik Bentrcia and Leila-Hayet Mouss. 'Auto-NAHL: A neural network approach for condition-based maintenance of complex industrial systems'. In: *IEEE Access* 9 (2021), pp. 152829–152840 (cit. on pp. 108, 115, 117, 139).

[226] Jun Wu, Pengfei Guo, Yiwei Cheng et al. 'Ensemble generalized multi-class support-vector-machine-based health evaluation of complex degradation systems'. In: *IEEE/ASME Transactions on Mechatronics* 25.5 (2020), pp. 2230–2240 (cit. on pp. 108, 115, 117, 139).

[227] Ashish Gupta, Hari Prabhat Gupta, Bhaskar Biswas and Tanima Dutta. 'An unseen fault classification approach for smart appliances using ongoing multivariate time series'. In: *IEEE Transactions on Industrial Informatics* 17.6 (2020), pp. 3731–3738 (cit. on pp. 108, 139).

[228] Yafei Lei, Wanlu Jiang, Anqi Jiang et al. 'Fault diagnosis method for hydraulic directional valves integrating PCA and XGBoost'. In: *Processes* 7.9 (2019), p. 589 (cit. on pp. 108, 115, 117, 139).

[229] Zachary C Lipton. 'The Mythos of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery.' In: *Queue* 16.3 (2018), pp. 31–57 (cit. on p. 109).

[230] Supriyo Chakraborty, Richard Tomsett, Ramya Raghavendra et al. 'Interpretability of deep learning models: A survey of results'. In: *2017 IEEE smartworld, ubiquitous intelligence & computing, advanced & trusted computed, scalable computing & communications, cloud & big data computing, Internet of people and smart city innovation*. IEEE. 2017, pp. 1–6 (cit. on p. 109).

[231] Sarah Wiegreffe and Yuval Pinter. 'Attention is not not explanation'. In: *arXiv preprint arXiv:1908.04626* (2019) (cit. on p. 109).

[232] Yafei Deng, Delin Huang, Shichang Du et al. 'A double-layer attention based adversarial network for partial transfer learning in machinery fault diagnosis'. In: *Computers in Industry* 127 (2021), p. 103399 (cit. on p. 115).

[233] *Case Western Reserve University Bearing Data Center*. Website. `https://engineering.case.edu/bearingdatacenter`. Last access: Sep 2021 (cit. on p. 118).

[234] Xiaoxi Ding and Qingbo He. 'Energy-fluctuated multiscale feature learning with deep convnet for intelligent spindle bearing fault diagnosis'. In: *IEEE Trans on Instrumentation and Measurement* 66.8 (2017), pp. 1926–1935 (cit. on pp. 118, 119).

[235] Wenliao Du, Jianfeng Tao, Yanming Li and Chengliang Liu. 'Wavelet leaders multifractal features based fault diagnosis of rotating mechanism'. In: *Mechanical Systems and Signal Processing* 43.1-2 (2014), pp. 57–75 (cit. on p. 119).

[236] Xiaohang Jin, Mingbo Zhao, Tommy WS Chow and Michael Pecht. 'Motor bearing fault diagnosis using trace ratio linear discriminant analysis'. In: *IEEE Transactions on Industrial Electronics* 61.5 (2013), pp. 2441–2451 (cit. on p. 119).

[237] PD McFadden and JD Smith. 'Model for the vibration produced by a single point defect in a rolling element bearing'. In: *Journal of sound and vibration* 96.1 (1984), pp. 69–82 (cit. on p. 120).

[238] Rui Zhao, Dongzhe Wang, Ruqiang Yan et al. 'Machine health monitoring using local feature-based gated recurrent unit networks'. In: *IEEE Transactions on Industrial Electronics* 65.2 (2017), pp. 1539–1548 (cit. on pp. 120, 121).

[239] Xinglong Pei, Xiaoyang Zheng and Jinliang Wu. 'Rotating Machinery Fault Diagnosis Through a Transformer Convolution Network Subjected to Transfer Learning'. In: *IEEE Transactions on Instrumentation and Measurement* 70 (2021), pp. 1–11 (cit. on p. 122).

[240] Jiehui Xu, Jianmin Wang, Mingsheng Long et al. 'Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting'. In: *Advances in Neural Information Processing Systems* 34 (2021) (cit. on p. 123).

[241] *Microsoft Turing*. Website. `https://turing.microsoft.com/`. Last access: Sep 2022 (cit. on p. 127).

[242] Alec Radford, Jong Wook Kim, Chris Hallacy et al. 'Learning transferable visual models from natural language supervision'. In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763 (cit. on p. 128).

[243] Jiachen Lu, Jinghan Yao, Junge Zhang et al. 'Soft: Softmax-free transformer with linear complexity'. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 21297–21309 (cit. on p. 128).

[244] Chris Chatfield. *The analysis of time series: an introduction*. Chapman and hall/CRC, 2003 (cit. on p. 132).

[245] Norbert Wiener. 'Generalized harmonic analysis'. In: *Acta mathematica* 55.1 (1930), pp. 117–258 (cit. on p. 132).

[246] DY You, XD Gao and S Katayama. 'Review of laser welding monitoring'. In: *Science and technology of welding and joining* 19.3 (2014), pp. 181–201 (cit. on p. 145).

[247] Valentin Gabeur, Chen Sun, Karteek Alahari and Cordelia Schmid. 'Multi-modal transformer for video retrieval'. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*. Springer. 2020, pp. 214–229 (cit. on p. 145).

[248] Jun Yu, Jing Li, Zhou Yu and Qingming Huang. 'Multimodal transformer with multi-view visual representation for image captioning'. In: *IEEE transactions on circuits and systems for video technology* 30.12 (2019), pp. 4467–4480 (cit. on p. 145).

[249] Aditya Prakash, Kashyap Chitta and Andreas Geiger. 'Multi-modal fusion transformer for end-to-end autonomous driving'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 7077–7087 (cit. on p. 145).

[250] Kaiming He, Xinlei Chen, Saining Xie et al. 'Masked Autoencoders Are Scalable Vision Learners'. In: *arXiv:2111.06377* (2021) (cit. on p. 145).

A

Discussion on Application and Implementation in Practice

This thesis proposed deep learning methods, developed with PyTorch, for sensor anomaly detection and process monitoring based on multi-sensor data streams. The practical application of the proposed method, including the deployment strategies and maintenance considerations, will be discussed in the appendix. Examples of implementation codes are also provided in the following sections, including the model architecture and the corresponding training process. Depending on the dataset, the hyperparameters of the models can be tuned to find the best performance on the target dataset. It is important to note that the methodology presented in this thesis has only been validated on publicly available datasets and has not been deployed in real industrial systems. The deployment methods discussed in this section are general application flows.

A.0.1 Integration into Existing Systems

The integration process begins with the collection of multi-sensor data streams. Our methods are designed to be adaptable to various sensor types that collect numerical data. Data preprocessing steps, including normalisation, noise reduction, and feature extraction, were not conducted before training the models, which means the models received raw sensor data for analysis.

Deployment of the PyTorch-based models into production environments can be divided into a two-step process. First, the trained model, saved as a .pth file, can be converted to an ONNX format to ensure compatibility across different platforms. Second, the ONNX model can be deployed to either a server or an edge device, which provides the computation and data processing capacity. The deep learning models presented in the thesis support batch processing for historical data analysis. This feature can be used for retrospective analysis, and model retraining purposes.

A.0.2 Maintenance and Updating

Maintaining the accuracy and relevance of the deployed model over time can be essential. A regular retraining schedule, leveraging newly gathered data to continuously refine the model. By fine-tuning the model to update its parameters in response to new data, the model can remain effective, as the characteristics of the data stream may evolve.

B Implementation codes - Chapter 3

B.1 Sensor Anomaly Injection

```
from google.colab import drive
drive.mount('/content/gdrive')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm

# Anomaly injection
import random

df_instance = pd.read_csv('/content/gdrive/MyDrive/Colab Notebooks/Sensor
    Anomaly Detection/Data/gps_das_acce_10107.csv', sep='\t')

def anomaly_gen(values, prob):
  x = random.uniform(0,1)
  cumulative_prob = 0.0
  for item, item_prob in zip(values, prob):
    cumulative_prob += item_prob
    if x < cumulative_prob:
      break

  return item

# Instant
i = 0
while(i<43599):
  values = [0, 1]
  prob = [0.95, 0.05]
  res = anomaly_gen(values, prob)
  if res == 0:
    df_instance.iloc[i,1] += 0
    df_instance.iloc[i,0] += 0
    i += 1
  else:
    ad = np.random.normal(loc=0.0 , scale=0.1)
    column_idx = np.random.randint(low=2,high=5)
    df_instance.iloc[i,1] = 1
    df_instance.iloc[i,0] = column_idx - 1
    df_instance.iloc[i,column_idx] = df_instance.iloc[i,column_idx] + ad *
   500 # Instant
    i += 1

'''
# Constant
duration = 3
i = 0
while(i<26000):
  a = np.random.uniform(0.00,5)


  values = [0, a]
  prob = [0.95, 0.05]
  res = anomaly_gen(values, prob)


  if res == 0:
    df_instance.iloc[i,1] = 0
    df_instance.iloc[i,0] = 0
    i += 1

  else:
    column_idx = np.random.randint(low=2,high=5)
    df_instance.iloc[i:i+duration,column_idx] = res
    df_instance.iloc[i:i+duration,1] = 1
    df_instance.iloc[i:i+duration,0] = column_idx - 1
    i += duration

# Bias
duration = 10
i = 0
# while(i<43600):
while(i<26000):
  values = [0, 1]
  prob = [0.95, 0.05]
  res = anomaly_gen(values, prob)

  if res == 0:
    df_instance.iloc[i,1] += 0
    df_instance.iloc[i,0] += 0
    i += 1

  else:
    column_idx = np.random.randint(low=2,high=5) # inject anomaly to random sensor

    ad = np.random.uniform(0,1)

    df_instance.iloc[i:i+duration,column_idx] = df_instance.iloc[i:i+duration,column_idx] + ad
    df_instance.iloc[i:i+duration,1] = 1
    df_instance.iloc[i:i+duration,0] = column_idx - 1
    i += duration
```

```

# Drift
duration = 20
i = 0
while(i<26000):
  values = [0, 1]
  prob = [0.95, 0.05]
  res = anomaly_gen(values, prob)

  if res == 0:
    df_instance.iloc[i,1] = 0
    df_instance.iloc[i,0] = 0
    i += 1

  else:
    column_idx = np.random.randint(low=2,high=5)
    ad = np.linspace(0, 2, duration)

    df_instance.iloc[i:i+duration,column_idx] = df_instance.iloc[i:i+
   duration,column_idx] + ad
    df_instance.iloc[i:i+duration,1] = 1
    df_instance.iloc[i,1] = 0  # anomaly starts from 0, 0 is not an anomaly
    df_instance.iloc[i:i+duration,0] = column_idx
    df_instance.iloc[i,0] = 0
    i += duration

# Mixex
duration = 20
i = 0
while(i<26000):
  values = [0, 1]
  prob = [0.95, 0.05]
  res = anomaly_gen(values, prob)


  if res == 0:
    df_instance.iloc[i,1] += 0
```

```
            df_instance.iloc[i,0] += 0
            i += 1

    else:
        column_idx = np.random.randint(low=2,high=5) # inject anomaly to random sensor
        ad_type_idx = np.random.randint(low=1, high=5) # randomly selected anomaly type

        if ad_type_idx == 1: # Instant
            ad = np.random.normal(loc=0.0 , scale=0.1) * 1000
            df_instance.iloc[i,column_idx] = df_instance.iloc[i,column_idx] + ad
            df_instance.iloc[i,0] = column_idx - 1 # faulty sensor
            df_instance.iloc[i,1] = 1 # label
            i += 1

        elif ad_type_idx == 2:  # Constant
            duration = 10
            ad = np.random.uniform(0.00,5)
            df_instance.iloc[i:i+duration,column_idx] = ad
            df_instance.iloc[i:i+duration,1] = 1
            df_instance.iloc[i:i+duration,0] = column_idx - 1
            i += duration
            # i += 1

        elif ad_type_idx == 3: # Drift
            duration = 20
            ad = np.linspace(0, 4, duration)
            df_instance.iloc[i:i+duration,column_idx] = df_instance.iloc[i:i+duration,column_idx] + ad
            df_instance.iloc[i:i+duration,1] = 1
            df_instance.iloc[i,1] = 0
            # df_instance.iloc[i+1,1] = 0
            df_instance.iloc[i:i+duration,0] = column_idx - 1
            df_instance.iloc[i,0] = 0
            # df_instance.iloc[i+1,0] = 0 # Uncomment: ingnore the initial small value of anomaly
```

```
      i += duration
      # i += 1

    elif ad_type_idx == 4: # Bias
      duration = 10
      ad = np.random.uniform(0,5)
      df_instance.iloc[i:i+duration,column_idx] = df_instance.iloc[i:i+
   duration,column_idx] + ad
      df_instance.iloc[i:i+duration,1] = 1
      df_instance.iloc[i:i+duration,0] = column_idx - 1
      i += duration
      # i += 1
'''


# Data visualisation
lower_limit = 0
upper_limit = 43599
value_gpsdas = df_instance.iloc[lower_limit:upper_limit,2].values
value_das = df_instance.iloc[lower_limit:upper_limit,3].values
value_das_acce = df_instance.iloc[lower_limit:upper_limit,4].values
label = df_instance.iloc[lower_limit:upper_limit,1].values
fault_sensor = df_instance.iloc[lower_limit:upper_limit,0].values

fig, ax = plt.subplots(figsize=(24,8))
plt.figure(figsize=(24,8))
ax.plot(value_das, label='Das')
ax.plot(value_gpsdas, label='GpsDas')
ax.plot(value_das_acce, label='Acc')
ax.plot(label, label='label')
ax.plot(fault_sensor, label='Fault_sensor')
ax.legend()

print(df_instance)
```

Listing B.1: Sensor Anomaly Injection Algorithm

B.2 DA-CNN model Implementation and Training Process

```python
from google.colab import drive
drive.mount('/content/gdrive')

import os
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
import numpy as np
from numpy import *
import math
import time
import torch.nn.functional as F
from tqdm import tqdm

use_gpu = torch.cuda.is_available()
print(use_gpu)
print(torch.cuda.device_count())
torch.cuda.current_device()
device = torch.device('cuda' if torch.cuda.is_available else 'cpu')
print(device)


data_10107_path = '/content/gdrive/MyDrive/Colab_Notebooks/Sensor Anomaly
    Detection/Data/Instant/instance_25_full.csv'

# Here we create the training dataset with anomaly injected.
# rate: Training dataset size / Total number
# stride: How many data point per step
# num_fold: How many vertors will be used for 1 sensor in a time window

class ad_training_instant(Dataset):
  def __init__(self, num_total=2590, window_size=15, stride=15, num_fold=1,
      rate=1):
```

```

    self.num_fold = num_fold
    self.window_size = window_size
    self.stride = stride

    self.num_data = int(num_total*rate)
    self.d_model = int(window_size/num_fold)

    self.y = torch.zeros(self.num_data).type(torch.long).to(device)
    self.x = torch.zeros(self.num_data, int(3*self.num_fold), self.d_model)
   .to(device)

    data = pd.read_csv(data_10107_path, sep='\t')

    # MinMax
    # data.iloc[:, [3,4]] = (data.iloc[:,[3,4]] - data.iloc[:,[3,4]].min())
     / (data.iloc[:,[3,4]].max() - data.iloc[:,[3,4]].min())
    # data.iloc[:, 5] = (data.iloc[:,5] - data.iloc[:,5].min()) / (data.
   iloc[:,5].max() - data.iloc[:,5].min())




    for i in tqdm(range(self.num_data)):
      self.x[i][0:self.num_fold, :] = torch.tensor(data.iloc[(self.stride*i
   ):(self.window_size+self.stride*i), 3].values.reshape(self.num_fold,
   self.d_model))
      self.x[i][self.num_fold:self.num_fold*2, :] = torch.tensor(data.iloc
   [(self.stride*i):(self.window_size+self.stride*i), 4].values.reshape(
   self.num_fold, self.d_model))
      self.x[i][self.num_fold*2:self.num_fold*3, :] = torch.tensor(data.
   iloc[(self.stride*i):(self.window_size+self.stride*i), 5].values.reshape
   (self.num_fold, self.d_model))

      if sum(data.iloc[(self.stride*i):(self.window_size+self.stride*i),
   2].values) == 0:
        self.y[i] = torch.tensor(0, dtype=torch.long)
      else:
        self.y[i] = torch.tensor(1, dtype=torch.long)


    self.len = self.x.shape[0]

  # Getter
  def __getitem__(self, index):
    return self.x[index], self.y[index]

  # Get length
  def __len__(self):
    return self.len

data = pd.read_csv(data_10107_path, sep='\t')

lower_limit = 0
upper_limit = 26000
value_gpsdas = data.iloc[lower_limit:upper_limit,3].values
value_das = data.iloc[lower_limit:upper_limit,4].values
value_das_acce = data.iloc[lower_limit:upper_limit,5].values


fig, ax = plt.subplots(figsize=(24,8))
plt.figure(figsize=(24,8))
ax.plot(value_das, label='Das')
ax.plot(value_gpsdas, label='GpsDas')

ax.plot(value_das_acce, label='Acc')
ax.legend()

dataset = ad_training_instant()

# Count the number of different classes
dataset_size = len(dataset)
classes = ['Normal', 'Faulty']
num_classes = len(classes)
```

```
img_dict = {}
for i in range(num_classes):
    img_dict[classes[i]] = 0

for i in range(dataset_size):
    img, label = dataset[i]
    img_dict[classes[label]] += 1

img_dict

train_size = 2340
test_size = 2590 - train_size
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [
    train_size, test_size], generator=torch.Generator().manual_seed(12))

class PositionwiseFeedForward(nn.Module):
  '''
  2 feed forward layer module
  '''

  def __init__(self, d_in, d_hid, dropout=0.1):
    super().__init__()
    self.w_1 = nn.Linear(d_in, d_hid)
    self.w_2 = nn.Linear(d_hid, d_in)
    self.layer_norm = nn.LayerNorm(d_in, eps=1e-6)
    self.dropout = nn.Dropout(dropout)

  def forward(self, x):

    residual = x

    x = self.w_2(F.relu(self.w_1(x)))
    x = self.dropout(x)
    x += residual

    x = self.layer_norm(x)

    return x

class PositionwiseFeedForward_CNN(nn.Module):
  '''
  2 feed forward layer module
  '''

  def __init__(self, d_in, d_hid, d_model, dropout=0.2):
    super().__init__()
    self.w_1 = nn.Conv1d(in_channels=d_in, out_channels=d_hid, kernel_size
   =1, bias=False)
    self.w_2 = nn.Conv1d(in_channels=d_hid, out_channels=d_in, kernel_size
   =1, bias=False)
    self.layer_norm = nn.LayerNorm([d_in], eps=1e-6)
    self.dropout = nn.Dropout(dropout)

  def forward(self, x):

    residual = x
    x = x.permute(0, 2, 1)
    x = self.w_2(F.relu(self.w_1(x)))
    x = self.dropout(x)
    x = x.permute(0, 2, 1)
    x += residual

    x = self.layer_norm(x)

    return x


class Attention(nn.Module):
  def __init__(self, d_model, num_heads, qkv_bias=True, qk_scale=None,
   attn_drop=0., proj_drop=0.1, position_bias=True):
    super().__init__()
    self.d_model = d_model
    self.num_heads = num_heads
    head_dim = d_model // num_heads
    self.scale = qk_scale or head_dim ** -0.5
    self.position_bias = position_bias

    self.qkv = nn.Linear(d_model, d_model *3, bias = qkv_bias)
    self.attn_drop = nn.Dropout(attn_drop)
    self.proj = nn.Linear(d_model, d_model)
    self.proj_drop = nn.Dropout(proj_drop)

    self.softmax = nn.Softmax(dim=-1)
    self.norm = nn.LayerNorm(self.d_model)
    # self.norm1 = nn.LayerNorm(self.d_model)

    # self.mlp = PositionwiseFeedForward(d_in=self.d_model, d_hid=self.d_model * 4)

  def forward(self, x, mask=None):
    B_, N, C = x.shape

    qkv = self.qkv(x).reshape(B_, N, 3, self.num_heads, C // self.num_heads).permute(2, 0, 3, 1, 4)
    q, k, v = qkv[0], qkv[1], qkv[2]  # make torchscript happy (cannot use tensor as tuple)
    q = q * self.scale
    attn = (q @ k.transpose(-2, -1))
    attn = self.softmax(attn)

    res = x
    x = (attn @ v).transpose(1, 2).reshape(B_, N, C)
    x = self.proj(x)
    x = self.proj_drop(x)
    x = self.norm(x + res)

    # res_mlp = x
    # x = self.mlp(x)
    # x = self.norm1(x + res_mlp)
    return x
```

```python

class Attention_layer_for_dual(nn.Module):
  def __init__(self, d_model, num_heads, qkv_bias=True, qk_scale=None,
   attn_drop=0., proj_drop=0.1, position_bias=True):
    super().__init__()
    self.d_model = d_model
    self.num_heads = num_heads
    head_dim = d_model // num_heads
    self.scale = qk_scale or head_dim ** -0.5
    self.position_bias = position_bias

    self.qkv = nn.Linear(d_model, d_model *3, bias = qkv_bias)
    self.attn_drop = nn.Dropout(attn_drop)
    self.proj = nn.Linear(d_model, d_model)
    self.proj_drop = nn.Dropout(proj_drop)

    self.softmax = nn.Softmax(dim=-1)
    self.norm = nn.LayerNorm(self.d_model)
    # self.norm1 = nn.LayerNorm(self.d_model)

    # self.mlp = PositionwiseFeedForward(d_in=self.d_model, d_hid=self.
   d_model * 4)

  def forward(self, x, mask=None):
    B_, N, C = x.shape

    qkv = self.qkv(x).reshape(B_, N, 3, self.num_heads, C // self.num_heads
   ).permute(2, 0, 3, 1, 4)
    q, k, v = qkv[0], qkv[1], qkv[2]  # make torchscript happy (cannot use
   tensor as tuple)
    q = q * self.scale
    attn = (q @ k.transpose(-2, -1))
    attn = self.softmax(attn)

    res = x
    x = (attn @ v).transpose(1, 2).reshape(B_, N, C)
    x = self.proj(x)
    x = self.proj_drop(x)
    x = self.norm(x + res)

    # res_mlp = x
    # x = self.mlp(x)
    # x = self.norm1(x + res_mlp)
    return x, attn


# sensor-wise and time wise attention, initialise the feature representation for the downstream
class Attention_dual(nn.Module):
  def __init__(self, d_model, num_heads, length_model, attn):
    super().__init__()
    self.d_model = d_model
    self.num_heads = num_heads
    self.length_model = length_model
    self.attn = attn

    self.norm = nn.LayerNorm(self.d_model)

    self.att_sensor_wise = Attention_layer_for_dual(d_model=self.d_model, num_heads=self.num_heads)
    self.att_time_wise = Attention_layer_for_dual(d_model=self.length_model, num_heads=self.num_heads)

  def forward(self, x, mask=None):
    res = x
    x1, attn_sensor = self.att_sensor_wise(x)

    # x1, attn_sensor = self.att_time_wise(x.permute(0,2,1))
    # x1 = x1.permute(0,2,1)

    x2, attn_time = self.att_time_wise(x.permute(0,2,1))
    x2 = x2.permute(0,2,1)
    # x2, attn_time = self.att_sensor_wise(x)


    x = x1 + x2
    x = self.norm(x + res)

    if self.attn == False:
      return x
    else:
      return x, attn_sensor, attn_time


class AD_model(nn.Module):
  def __init__(self, attn_depth, num_class, d_model, num_heads_attn,
   orth_gain, length_model):
    super().__init__()

    self.d_model = d_model
    self.num_heads_attn = num_heads_attn
    self.attn_depth = attn_depth
    self.num_class = num_class
    self.length_model = length_model
    self.cls_token = torch.randn(1,d_model).to(device)

    self.att_dual = Attention_dual(d_model=self.d_model, num_heads=self.
   num_heads_attn, length_model=self.length_model, attn=True)

    self.attn = nn.ModuleList()
    for i in range(self.attn_depth):
      # self.attn.append(Attention(d_model=self.d_model, num_heads=self.
   num_heads_attn))

      self.attn.append(Attention_dual(d_model=self.d_model, num_heads=self.
   num_heads_attn, length_model=self.length_model+1, attn = False))

      # self.attn.append(PositionwiseFeedForward_CNN(d_in=(self.
   length_model+1), d_hid=4 * (self.length_model+1), d_model=self.d_model))
    # cls token added, hence + 1
      self.attn.append(PositionwiseFeedForward_CNN(d_in=(self.d_model),
    d_hid=4 * (self.d_model), d_model=self.d_model)) # cls token added,
    hence + 1
      # self.attn.append(PositionwiseFeedForward(d_in=self.d_model, d_hid=4
     * self.d_model))

    self.classifier = nn.Linear(self.d_model, self.num_class)

    # initialisation
    for p in self.parameters():
      if p.dim() > 1:
        nn.init.xavier_uniform_(p)

    if orth_gain is not None:
      torch.nn.init.orthogonal_(self.classifier.weight, gain = orth_gain)
    self.classifier.bias.data.zero_()


  def forward(self, x):
    B, _, _ = x.shape


    #print(x.shape)

    x, attn_sensor, attn_time = self.att_dual(x)
    x = torch.cat((self.cls_token.unsqueeze(0).repeat(B,1,1), x), dim=-2)

    for i in range(self.attn_depth):
      x = self.attn[i](x)

    x = x[:, 0, :]
    x = self.classifier(x)

    return x, attn_sensor, attn_time


model = AD_model(
    attn_depth=7,
```

```python
    num_class=2,
    d_model=15,
    num_heads_attn=1,
    orth_gain=1.41,
    length_model=3
    )
model.to(device)
# model.load_state_dict(torch.load("/content/gdrive/MyDrive/Colab_Notebooks
    /Sensor Anomaly Detection/Instant/25/Instant25.pth"))
def get_p(net):
  t = sum(p.numel() for p in net.parameters())
  trainable = sum(p.numel() for p in net.parameters() if p.requires_grad)
  return t, trainable
print(get_p(model))


weights = [1.5 if label == 1 else 1 for data, label in train_dataset]

sampler = torch.utils.data.sampler.WeightedRandomSampler(weights, len(
    train_dataset), replacement=True)

batch_size = 16
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size,
    shuffle=False, drop_last=True, sampler=sampler)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size,
    shuffle=True, drop_last=True)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(), lr=0.0001)

def train(epoch):

    correct1 = 0
    total1 = 0
    running_loss = 0

    epoch_running_loss = 0
    epoch_correct1 = 0
    epoch_total1 = 0

    for batch_idx, data in enumerate(train_loader):
      images, labels = data
      optimizer.zero_grad()
      outputs, _, _ = model(images)
      outputs = outputs.squeeze(1)
      loss = criterion(outputs, labels)
      loss.backward()
      #torch.nn.utils.clip_grad_norm_(model.parameters(), args.clip)
      optimizer.step()

      running_loss += loss.detach().cpu().item()
      epoch_running_loss += loss.detach().cpu().item()

      idx_control = (train_size // batch_size) // 4

      if (batch_idx + 1) % idx_control == 0:
        train_loss_list.append(running_loss / batch_size / idx_control)
        #print('[%d, %d], training loss is %.3f' % (epoch, batch_idx, running_loss / batch_size / idx_control))
        running_loss = 0

      if (batch_idx + 1) % (train_size // batch_size) == 0:
        epoch_train_loss_list.append(epoch_running_loss / (train_size // batch_size * batch_size))


      _, predict1 = torch.max(outputs, dim=1)
      predict1 = predict1.detach().cpu()
      labels = labels.detach().cpu()
      correct1 += (labels == predict1).sum().item()
      total1 += labels.size(0)

      epoch_correct1 += (labels == predict1).sum().item()
      epoch_total1 += labels.size(0)

```

```python
        if (batch_idx + 1) % idx_control == 0:
          train_accuracy_list.append(100 * (correct1 / total1))
          #print('Train acc: ', 100 * (correct1 / total1))
          correct1 = 0
          total1 = 0

        if (batch_idx + 1) % (train_size // batch_size) == 0:
          epoch_train_accuracy_list.append(100 * (epoch_correct1 / epoch_total1))
          print('Epoch train acc: ', 100 * (epoch_correct1 / epoch_total1))



def test():
    correct = 0
    total = 0
    test_loss = 0

    epoch_correct = 0
    epoch_total = 0
    epoch_test_loss = 0
    epoch_TP = 0
    epoch_TN = 0
    epoch_FP = 0
    epoch_FN = 0

    with torch.no_grad():
        for batch_idx, data in enumerate(test_loader):
            images, labels = data
            outputs, _, _ = model(images)
            outputs = outputs.squeeze(1)
            _, predict = torch.max(outputs, dim=1)
            predict = predict.detach().cpu()
            labels = labels.detach().cpu()
            correct += (labels == predict).sum().item()
            total += labels.size(0)

            epoch_correct += (labels == predict).sum().item()
            epoch_total += labels.size(0)

            epoch_TP += ((predict == 1) & (labels.data == 1)).sum().item()
            epoch_TN += ((predict == 0) & (labels.data == 0)).sum().item()
            epoch_FN += ((predict == 0) & (labels.data == 1)).sum().item()
            epoch_FP += ((predict == 1) & (labels.data == 0)).sum().item()

            idx_control = (test_size // batch_size) // 4

            if (batch_idx + 1) % idx_control == 0:
              test_accuracy_list.append(100 * (correct / total))
              #print('correct/total:%d/%d, Test Accuracy:%.2f%%' % (correct
, total, 100 * (correct / total)))
              correct = 0
              total = 0

            if (batch_idx + 1) % (test_size // batch_size) == 0:
              epoch_test_accuracy_list.append(100 * (epoch_correct /
epoch_total))

              p = epoch_TP/(epoch_TP + epoch_FP)
              s = epoch_TP/(epoch_TP + epoch_FN)

              print('Epoch test acc: ', 100 * (epoch_correct / epoch_total)
)
              print('Precision: ', 100 * p)
              print('Sensitivity: ', 100 * s)
              print('F1 Score: ', 100 * 2 * s * p / (s + p))

              specificity = epoch_TN/(epoch_TN + epoch_FP)
              print('Specificity: ', 100 * specificity)


            outputs = outputs.detach().cpu()
            loss = criterion(outputs, labels)

            test_loss += loss.detach().cpu().item()
            epoch_test_loss += loss.detach().cpu().item()

            if (batch_idx + 1) % idx_control == 0:
              test_loss_list.append(test_loss / batch_size / idx_control)
              #print('[%d, %d], test loss is %.3f' % (epoch, batch_idx, test_loss / batch_size / idx_control))
              test_loss = 0

            if (batch_idx + 1) % (test_size // batch_size) == 0:
              epoch_test_loss_list.append(epoch_test_loss / batch_size / idx_control)

            #print('epoch', epoch)
            # if (epoch+1) % 150 == 0:
            #     _, attention_map = model(images)
            #     # print('attention_map', attention_map[0])
            #     attention_map = attention_map[0][0][0]
            #     attention_map = np.array(attention_map.detach(), dtype=float)
            #     # print(attention_map.shape)
            #     fig = plt.figure()
            #     ax = fig.add_subplot(111)
            #     ax.set_yticks(range(120))
            #     ax.set_xticks(range(120))
            #     im = ax.imshow(attention_map, cmap=plt.cm.viridis)
            #     plt.colorbar(im)
            #     plt.show()





if __name__ == '__main__':
    train_loss_list = []
    test_accuracy_list = []
    train_accuracy_list = []
```

```
test_loss_list = []
train_epoch = 300
train_time = []

epoch_train_loss_list = []
epoch_train_accuracy_list = []
epoch_test_loss_list = []
epoch_test_accuracy_list = []



for epoch in range(train_epoch):
    print('epoch: ', epoch)
    model.train()

    start_time = time.time()
    train(epoch)
    end_time = time.time()
    epoch_time = (end_time - start_time)
    train_time.append(epoch_time)


    model.eval()
    test()



print('trian time per epoch: ', mean(train_time))


y1 = test_accuracy_list
y2 = train_loss_list
y3 = train_accuracy_list
y4 = test_loss_list

y11 = epoch_test_accuracy_list
y22 = epoch_train_loss_list
```

```
y33 = epoch_train_accuracy_list
y44 = epoch_test_loss_list

plt.subplot(4, 1, 1)
plt.plot(y1, 'o-')
plt.xlabel('Test acc vs. epochs')
plt.ylabel('Test accuracy')

plt.subplot(4, 1, 2)
plt.plot(y3, '.-')
plt.xlabel('Train acc vs. epochs')
plt.ylabel('Train acc')

plt.subplot(4, 1, 3)
plt.plot(y4, '.-')
plt.xlabel('Test loss vs. epochs')
plt.ylabel('Test loss')

plt.subplot(4, 1, 4)
plt.plot(y2, 'o-')
plt.xlabel('Train loss vs. epochs')
plt.ylabel('Train loss')

plt.show()




plt.subplot(4, 1, 1)
plt.plot(y11, 'o-')
plt.xlabel('Test acc vs. epochs')
plt.ylabel('Test accuracy')

plt.subplot(4, 1, 2)
plt.plot(y33, '.-')
plt.xlabel('Train acc vs. epochs')
plt.ylabel('Train acc')
```

```

    plt.subplot(4, 1, 3)
    plt.plot(y44, '.-')
    plt.xlabel('Test loss vs. epochs')
    plt.ylabel('Test loss')

    plt.subplot(4, 1, 4)
    plt.plot(y22, 'o-')
    plt.xlabel('Train loss vs. epochs')
    plt.ylabel('Train loss')

    plt.show()


torch.save(model.state_dict(), "/content/gdrive/MyDrive/Colab_Notebooks/
    Sensor Anomaly Detection/Instant/25/Instant25.pth")

test_loader = DataLoader(dataset=test_dataset, batch_size=16, shuffle=False
    , drop_last=True)

# ROC curve

model_test = model
model_test.to(device)

from sklearn.metrics import roc_curve, auc
# After training, get predictions for the ROC curve
model_test.eval()  # set the model to evaluation mode
y_true = []
y_score = []

with torch.no_grad():
    for inputs, labels in test_loader:
        outputs, _, _ = model_test(inputs)
        # Store true labels and predicted probabilities for class 1
        y_true.extend(labels.cpu().numpy())

        y_score.extend(outputs[:, 1].cpu().numpy())


# print(y_true)
# print(y_score)

# Compute ROC curve and ROC area
fpr, tpr, _ = roc_curve(y_true, y_score)
roc_auc = auc(fpr, tpr)
print(roc_auc)

print(f"FPR at 95% TPR: {fpr[np.where(tpr >= 0.95)[0][0]]}")

# Plot the ROC curve
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange', lw=lw, label='ROC curve (area = %0.2
    f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

Listing B.2: DA-CNN model Implementation and Training Process

C

Implementation codes - Chapter 4

```
!pip install transformers
from google.colab import drive
drive.mount('/content/gdrive')

import os
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
import numpy as np
from transformers import GPT2Model

data = pd.read_csv('/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
    Monitoring of Hydraulic Systems/profile.txt', sep='\t')
#data.columns = ['1', '2', '3', '4', '5']
# data = data.values
# 1: Cooler 2: Valve 3: Internal pump leakage 4: Hydraulic accumulator 5:
    Stable flag
# print(data.shape[1])
# print(data.shape[0])
# print(data)
# count1, division = np.histogram(data[:, 0])
# count2, division = np.histogram(data[:, 1])
# count3, division = np.histogram(data[:, 2])
# count4, division = np.histogram(data[:, 3])
# count5, division = np.histogram(data[:, 4])
# print(count1)
# print(count2)
# print(count3)
# print(count4)
# print(count5)

use_gpu = torch.cuda.is_available()
print(use_gpu)
print(torch.cuda.device_count())
torch.cuda.current_device()
device = torch.device('cuda' if torch.cuda.is_available else 'cpu')
print(device)

class Hydraulic_data(Dataset):
  def __init__(self):
    self.y = torch.zeros(2204).type(torch.long).to(device)
    self.x = torch.zeros(2204, 728, 60).to(device)

    EPS1_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
   Monitoring of Hydraulic Systems/EPS1.txt'
    PS1_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
   Monitoring of Hydraulic Systems/PS1.txt'
    PS2_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
   Monitoring of Hydraulic Systems/PS2.txt'
    PS3_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
   Monitoring of Hydraulic Systems/PS3.txt'
    PS4_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
   Monitoring of Hydraulic Systems/PS4.txt'
    PS5_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
   Monitoring of Hydraulic Systems/PS5.txt'
```

```
PS6_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/PS6.txt'

FS1_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/FS1.txt'
FS2_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/FS2.txt'

CE_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/CE.txt'
CP_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/CP.txt'
SE_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/SE.txt'
TS1_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/TS1.txt'
TS2_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/TS2.txt'
TS3_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/TS3.txt'
TS4_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/TS4.txt'
VS1_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/VS1.txt'

Label_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/profile.txt'

EPS1 = pd.read_csv(EPS1_path, sep='\ ')

PS1 = pd.read_csv(PS1_path, sep='\ ')
PS2 = pd.read_csv(PS2_path, sep='\ ')
PS3 = pd.read_csv(PS3_path, sep='\ ')
PS4 = pd.read_csv(PS4_path, sep='\ ')
PS5 = pd.read_csv(PS5_path, sep='\ ')
PS6 = pd.read_csv(PS6_path, sep='\ ')

```

```
    FS1 = pd.read_csv(FS1_path, sep='\ ')
    FS2 = pd.read_csv(FS2_path, sep='\ ')

    CE = pd.read_csv(CE_path, sep='\ ')
    CP = pd.read_csv(CP_path, sep='\ ')
    SE = pd.read_csv(SE_path, sep='\ ')
    TS1 = pd.read_csv(TS1_path, sep='\ ')
    TS2 = pd.read_csv(TS2_path, sep='\ ')
    TS3 = pd.read_csv(TS3_path, sep='\ ')
    TS4 = pd.read_csv(TS4_path, sep='\ ')
    VS1 = pd.read_csv(VS1_path, sep='\ ')

    Label = pd.read_csv(Label_path, sep='\t')
    Label = Label.replace([3, 20, 100], [0, 1, 2])

    for i in range(2204):
      #print(torch.tensor(EPS1.iloc[[i]].values).shape)
      self.x[i][0:100, :] = torch.tensor(EPS1.iloc[[i]].values.reshape(100,
 60))
      self.x[i][100:200, :] = torch.tensor(PS1.iloc[[i]].values.reshape
(100, 60))
      self.x[i][200:300, :] = torch.tensor(PS2.iloc[[i]].values.reshape
(100, 60))
      self.x[i][300:400, :] = torch.tensor(PS3.iloc[[i]].values.reshape
(100, 60))
      self.x[i][400:500, :] = torch.tensor(PS4.iloc[[i]].values.reshape
(100, 60))
      self.x[i][500:600, :] = torch.tensor(PS5.iloc[[i]].values.reshape
(100, 60))
      self.x[i][600:700, :] = torch.tensor(PS6.iloc[[i]].values.reshape
(100, 60))

      self.x[i][700:710, :] = torch.tensor(FS1.iloc[[i]].values.reshape(10,
 60))
      self.x[i][710:720, :] = torch.tensor(FS2.iloc[[i]].values.reshape(10,
 60))

```

```
      self.x[i][720, :] = torch.tensor(CE.iloc[[i]].values)
      self.x[i][721, :] = torch.tensor(CP.iloc[[i]].values)
      self.x[i][722, :] = torch.tensor(SE.iloc[[i]].values)
      self.x[i][723, :] = torch.tensor(TS1.iloc[[i]].values)
      self.x[i][724, :] = torch.tensor(TS2.iloc[[i]].values)
      self.x[i][725, :] = torch.tensor(TS3.iloc[[i]].values)
      self.x[i][726, :] = torch.tensor(TS4.iloc[[i]].values)
      self.x[i][727, :] = torch.tensor(VS1.iloc[[i]].values)

      self.y[i] = torch.tensor(Label.iloc[i,0])
      self.len = self.x.shape[0]

  # Getter
  def __getitem__(self, index):
    return self.x[index], self.y[index]

  # Get length
  def __len__(self):
    return self.len


dataset = Hydraulic_data()
train_size = 1704
test_size = 2204 - train_size
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [
    train_size, test_size])

input_dim = 60
output_dim = 3

class FPT(nn.Module):

    def __init__(
            self,
            input_dim,
            output_dim,
            model_name='gpt2',
            pretrained=False,
            return_last_only=True,
            use_embeddings_for_in=False,
            in_layer_sizes=None,
            out_layer_sizes=None,
            freeze_trans=True,
            freeze_in=False,
            freeze_pos=False,
            freeze_ln=False,
            freeze_attn=True,
            freeze_ff=True,
            freeze_out=False,
            dropout=0.1,
            orth_gain=1.41,
    ):
        super().__init__()

        self.input_dim = input_dim
        self.output_dim = output_dim
        self.model_name = model_name
        self.return_last_only = return_last_only
        self.use_embeddings_for_in = use_embeddings_for_in

        self.in_layer_sizes = [] if in_layer_sizes is None else
in_layer_sizes
        self.out_layer_sizes = [] if out_layer_sizes is None else
out_layer_sizes
        self.dropout = dropout

        if 'gpt' in model_name:
            assert model_name in ['gpt2', 'gpt2-medium', 'gpt2-large', '
gpt2-xl']

            from transformers import GPT2Model

            pretrained_transformer = GPT2Model.from_pretrained(model_name)
            if pretrained:
```

```python
                self.transformer = pretrained_transformer
            else:
                self.transformer = GPT2Model(pretrained_transformer.config)

            if model_name == 'gpt2':
                embedding_size = 768
            elif model_name == 'gpt2-medium':
                embedding_size = 1024
            elif model_name == 'gpt2-large':
                embedding_size = 1280
            elif model_name == 'gpt2-xl':
                embedding_size = 1600

        else:
            raise NotImplementedError('model_name not implemented')

        if use_embeddings_for_in:
            self.in_net = nn.Embedding(input_dim, embedding_size)
        else:
            in_layers = []
            last_output_size = input_dim
            for size in self.in_layer_sizes:
                layer = nn.Linear(last_output_size, size)
                if orth_gain is not None:
                    torch.nn.init.orthogonal_(layer.weight, gain=orth_gain)
                layer.bias.data.zero_()

                in_layers.append(layer)
                in_layers.append(nn.ReLU())
                in_layers.append(nn.Dropout(dropout))
                last_output_size = size

            final_linear = nn.Linear(last_output_size, embedding_size)
            if orth_gain is not None:
                torch.nn.init.orthogonal_(final_linear.weight, gain=
orth_gain)
            final_linear.bias.data.zero_()

            in_layers.append(final_linear)
            in_layers.append(nn.Dropout(dropout))

            self.in_net = nn.Sequential(*in_layers)

        out_layers = []
        last_output_size = embedding_size
        for size in self.out_layer_sizes:
            out_layers.append(nn.Linear(last_output_size, size))
            out_layers.append(nn.ReLU())
            out_layers.append(nn.Dropout(dropout))
            last_output_size = size
        out_layers.append(nn.Linear(last_output_size, output_dim))
        self.out_net = nn.Sequential(*out_layers)

        if freeze_trans:
            for name, p in self.transformer.named_parameters():
                name = name.lower()
                if 'ln' in name:
                    p.requires_grad = not freeze_ln
                elif 'wpe' in name:
                    p.requires_grad = not freeze_pos
                elif 'mlp' in name:
                    p.requires_grad = not freeze_ff
                elif 'attn' in name:
                    p.requires_grad = not freeze_attn
                else:
                    p.requires_grad = False
        if freeze_in:
            for p in self.in_net.parameters():
                p.requires_grad = False
        if freeze_out:
            for p in self.out_net.parameters():
                p.requires_grad = False

    def forward(self, x, output_attentions=True):
```

```python

        orig_dim = x.shape[-1]
        if orig_dim != self.input_dim and not self.use_embeddings_for_in:
            if orig_dim % self.input_dim != 0:
                raise ValueError('dimension of x must be divisible by patch size')
            ratio = orig_dim // self.input_dim
            x = x.reshape(x.shape[0], x.shape[1] * ratio, self.input_dim)
        else:
            ratio = 1

        x = self.in_net(x)

        transformer_outputs = self.transformer(
            inputs_embeds=x,
            return_dict=True,
            output_attentions=output_attentions,
        )
        x = transformer_outputs.last_hidden_state

        if self.return_last_only:
            x = x[:, -ratio:]

        x = self.out_net(x)
        if self.return_last_only and ratio > 1:
            x = x.reshape(x.shape[0], x.shape[1] // ratio, ratio * self.output_dim)

        if output_attentions:
            return x, transformer_outputs.attentions
        else:
            return x

model = FPT(
    input_dim=input_dim,
    output_dim=3,
    model_name='gpt2',
    pretrained=True,
    return_last_only=True,
    use_embeddings_for_in=False,
    in_layer_sizes=None,
    out_layer_sizes=None,
    freeze_trans=True,
    freeze_in=False,
    freeze_pos=False,
    freeze_ln=False,
    freeze_attn=True,
    freeze_ff=True,
    freeze_out=False,
    dropout=0.1,
    orth_gain=1.41,
)
model.to(device)
print(device)

batch_size = 8
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size,
    shuffle=True, drop_last=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size,
    shuffle=True, drop_last=True)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(), lr=0.001)

def train(epoch):
    running_loss = 0
    correct1 = 0
    total1 = 0
    for batch_idx, data in enumerate(train_loader):
        images, labels = data
        optimizer.zero_grad()
        outputs, _ = model(images)
        #print(outputs)
        outputs = outputs.squeeze(1)
        #print(outputs.shape)
        #print(labels.shape)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.detach().cpu().item()
        if (batch_idx + 1) % 100 == 0:
            loss_list.append(running_loss / batch_size)
        if (batch_idx + 1) % 206 == 0:
            print('[%d, %d], training loss is %.2f' % (epoch, batch_idx,
    running_loss / 206))

        _, predict1 = torch.max(outputs, dim=1)
        predict1 = predict1.detach().cpu()
        labels = labels.detach().cpu()
        correct1 += (labels == predict1).sum().item()
        total1 += labels.size(0)

        if (batch_idx + 1) % 100 == 0:
          accuracy_list1.append(100 * (correct1 / total1))

        if (batch_idx + 1) % 206 == 0:
          print('Train acc: ', 100 * (correct1 / total1))



def test():
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, data in enumerate(test_loader):
            images, labels = data
            outputs, _ = model(images)
            outputs = outputs.squeeze(1)
            _, predict = torch.max(outputs, dim=1)
            predict = predict.detach().cpu()
            labels = labels.detach().cpu()
            #print(outputs)
```

```
                #print(predict)
                #print(labels)
                correct += (labels == predict).sum().item()
                total += labels.size(0)
                if (batch_idx + 1) % 62 == 0:
                  accuracy_list.append(100 * (correct / total))

        print('correct/total:%d/%d, Test Accuracy:%.2f%%' % (correct, total
    , 100 * (correct / total)))


if __name__ == '__main__':
    loss_list = []
    accuracy_list = []
    accuracy_list1 = []
    for epoch in range(10):
        print('epoch: ', epoch)
        model.train()
        train(epoch)
        model.eval()
        test()

    y1 = accuracy_list
    y2 = loss_list
    y3 = accuracy_list1
    #print('list: ', y3)

    plt.subplot(3, 1, 1)
    plt.plot(y1, 'o-')
    plt.title('test')
    plt.ylabel('Test accuracy')
    plt.subplot(3, 1, 2)
    plt.plot(y2, '.-')
    plt.xlabel('Train loss vs. epochs')
    plt.ylabel('Train loss')
    plt.subplot(3, 1, 3)
    plt.plot(y3, '.-')
    plt.xlabel('Train acc vs. epochs')
    plt.ylabel('Train acc')
    plt.show()

    # Attention map plot

    torch.save(model.state_dict(), '/content/gdrive/MyDrive/Colab Notebooks
   /Saved_models/Hydraulic_Task1_1.pth')
```

Listing C.1: Deep Transfer Learning with Self-attention for Industry Sensor Fusion Tasks

D

Implementation codes - Chapter 5

```
from google.colab import drive
drive.mount('/content/gdrive')

import numpy as np
import torch
import torch.nn as nn
import math
import time
import torch.nn.functional as F

from numpy import *
import matplotlib.pyplot as plt
import os
import pandas as pd
from torch.utils.data import Dataset
from torch.utils.data import DataLoader

torch.manual_seed(0)

use_gpu = torch.cuda.is_available()
print(use_gpu)
print(torch.cuda.device_count())
torch.cuda.current_device()
device = torch.device('cuda' if torch.cuda.is_available else 'cpu')
print(device)

class Hydraulic_data(Dataset):
  def __init__(self):
    self.y = torch.zeros(2204).type(torch.long).to(device)
    self.x = torch.zeros(2204, 728, 60).to(device)

    EPS1_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition Monitoring of Hydraulic Systems/EPS1.txt'
    PS1_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition Monitoring of Hydraulic Systems/PS1.txt'
    PS2_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition Monitoring of Hydraulic Systems/PS2.txt'
    PS3_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition Monitoring of Hydraulic Systems/PS3.txt'
    PS4_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition Monitoring of Hydraulic Systems/PS4.txt'
    PS5_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition Monitoring of Hydraulic Systems/PS5.txt'
    PS6_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition Monitoring of Hydraulic Systems/PS6.txt'

    FS1_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition Monitoring of Hydraulic Systems/FS1.txt'
    FS2_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition Monitoring of Hydraulic Systems/FS2.txt'

    CE_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition Monitoring of Hydraulic Systems/CE.txt'
    CP_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition Monitoring of Hydraulic Systems/CP.txt'
    SE_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition Monitoring of Hydraulic Systems/SE.txt'
    TS1_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
```

```
Monitoring of Hydraulic Systems/TS1.txt'
 TS2_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/TS2.txt'
 TS3_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/TS3.txt'
 TS4_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/TS4.txt'
 VS1_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/VS1.txt'

 Label_path = '/content/gdrive/MyDrive/Colab Notebooks/Data/Condition
Monitoring of Hydraulic Systems/profile.txt'

 EPS1 = pd.read_csv(EPS1_path, sep='\ ')

 PS1 = pd.read_csv(PS1_path, sep='\ ')
 PS2 = pd.read_csv(PS2_path, sep='\ ')
 PS3 = pd.read_csv(PS3_path, sep='\ ')
 PS4 = pd.read_csv(PS4_path, sep='\ ')
 PS5 = pd.read_csv(PS5_path, sep='\ ')
 PS6 = pd.read_csv(PS6_path, sep='\ ')

 FS1 = pd.read_csv(FS1_path, sep='\ ')
 FS2 = pd.read_csv(FS2_path, sep='\ ')

 CE = pd.read_csv(CE_path, sep='\ ')
 CP = pd.read_csv(CP_path, sep='\ ')
 SE = pd.read_csv(SE_path, sep='\ ')
 TS1 = pd.read_csv(TS1_path, sep='\ ')
 TS2 = pd.read_csv(TS2_path, sep='\ ')
 TS3 = pd.read_csv(TS3_path, sep='\ ')
 TS4 = pd.read_csv(TS4_path, sep='\ ')
 VS1 = pd.read_csv(VS1_path, sep='\ ')

 Label = pd.read_csv(Label_path, sep='\t')
 Label = Label.replace([3, 20, 100], [0, 1, 2])

```

```
  for i in range(2204):
    #print(torch.tensor(EPS1.iloc[[i]].values).shape)
    self.x[i][0:100, :] = torch.tensor(EPS1.iloc[[i]].values.reshape(100, 60))
    self.x[i][100:200, :] = torch.tensor(PS1.iloc[[i]].values.reshape(100, 60))
    self.x[i][200:300, :] = torch.tensor(PS2.iloc[[i]].values.reshape(100, 60))
    self.x[i][300:400, :] = torch.tensor(PS3.iloc[[i]].values.reshape(100, 60))
    self.x[i][400:500, :] = torch.tensor(PS4.iloc[[i]].values.reshape(100, 60))
    self.x[i][500:600, :] = torch.tensor(PS5.iloc[[i]].values.reshape(100, 60))
    self.x[i][600:700, :] = torch.tensor(PS6.iloc[[i]].values.reshape(100, 60))

    self.x[i][700:710, :] = torch.tensor(FS1.iloc[[i]].values.reshape(10, 60))
    self.x[i][710:720, :] = torch.tensor(FS2.iloc[[i]].values.reshape(10, 60))

    self.x[i][720, :] = torch.tensor(CE.iloc[[i]].values)
    self.x[i][721, :] = torch.tensor(CP.iloc[[i]].values)
    self.x[i][722, :] = torch.tensor(SE.iloc[[i]].values)
    self.x[i][723, :] = torch.tensor(TS1.iloc[[i]].values)
    self.x[i][724, :] = torch.tensor(TS2.iloc[[i]].values)
    self.x[i][725, :] = torch.tensor(TS3.iloc[[i]].values)
    self.x[i][726, :] = torch.tensor(TS4.iloc[[i]].values)
    self.x[i][727, :] = torch.tensor(VS1.iloc[[i]].values)

    self.y[i] = torch.tensor(Label.iloc[i,0])
    self.len = self.x.shape[0]

# Getter
def __getitem__(self, index):
  return self.x[index], self.y[index]
```

```python

  # Get length
  def __len__(self):
    return self.len


dataset = Hydraulic_data()


train_size = 1704
test_size = 2204 - train_size
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size, test_size], generator=torch.Generator().manual_seed(12))


# When the correlated channels have been merged, it can return a new tensor with top_k channels being removed
def index_delete(tensor, index):
  mask = torch.ones(tensor.numel(), dtype=torch.bool)
  mask[index] = False
  return tensor[mask]

class CrossCorrelation(nn.Module):
    """
    CrossCorrelation Mechanism with the following two phases:
    (1) channel-based (sensor wise) dependencies discovery
    (2) aggregation of highly correlated channel (sensor wise)
    This block can replace the self-attention family mechanism seamlessly.
    This block can shrunk the input data space by nonelinear dependencis discorvery with O(nlogn) computation complexity
    """
    def __init__(self, num_topk, d_model, num_heads, compression_scale=0.7, attention_dropout=0.1, output_attention=True):
      super(CrossCorrelation, self).__init__()
      self.compression_scale = compression_scale
      self.output_attention = output_attention
      self.dropout = nn.Dropout(attention_dropout)
```

```
        self.top_k = num_topk

        self.crosscorr_dim = d_model // num_heads

        self.norm = nn.LayerNorm(self.crosscorr_dim, elementwise_affine=False)

    def correlation_removal_training(self, values, corr):
        """
        Speed up version of Crosscorrelation (a batch-normalisation style design: corr will be calculated based on all the samples inside one batch).
        This is for training phase.
        """
        head = values.shape[1]
        channel = values.shape[2]
        length = values.shape[3]

        # Find top k most corelated channels
        top_k = self.top_k

        # Use absolute value to evaluate the corr (B, H, C, L)
        avg_corr = torch.mean(torch.mean(torch.abs_(corr), dim=1), dim=0) # Mean of Batch and attention heads (C, L)
        max_corr, time_delay_index = torch.max(avg_corr, dim=-1) # max(R(t+n) for n belong to (0,L)) for all channel (C)
        max_corr = max_corr/(length*(channel-1)) # Pearson correlation coefficient can be estimated by 1/n * sum(y1(t)*y2(t+n)) n belong to (0,t)
                                 # as corr is calculated between y and y1+y2+...+yn(channel), the coeff should be devided by num_channel
                                 # hence, max_corr belong to (0,1)

        # find the most correlated cluster
        weights, index = torch.topk(max_corr, top_k, dim=-1) # channel index

        # update corr
        tmp_corr = torch.softmax((1-weights), dim=-1)

        # cluster merge with time delay at max corr
        time_delay = time_delay_index.index_select(-1, index)
        tmp = torch.mul(values.index_select(-2, index), tmp_corr.reshape(top_k,1))

        for i in range(top_k): # Align the most relevant time_delay
          tmp[:,:,i,:] = torch.roll(tmp[:,:,i,:], int(time_delay[i]), -1)

        tmp_sum = torch.sum(tmp, dim=-2) # (B,H,L) unsq --> (B,H,C,L)
        merged_channel = tmp_sum.unsqueeze(2)

        # obtain the index of the rest channels
        new_index = index_delete(torch.arange(channel), index.to(device))
        tmp_values = values.index_select(-2, new_index.to(device))

        new_values = torch.cat((merged_channel, tmp_values), dim=-2)

        return new_values, index.to(device), new_index.to(device)


      def forward(self, queries, keys, values):
        _,C,_,_ = queries.shape # B,C,H,L

        # Channel based dependencies
        queries = self.norm(queries.permute(0,2,1,3).contiguous())
        queries = torch.sum(queries, dim=-2).unsqueeze(-2).repeat(1,1,C,1) - queries
        keys = self.norm(keys.permute(0,2,1,3).contiguous())

        q_fft = torch.fft.rfft(queries, dim=-1)
        k_fft = torch.fft.rfft(keys, dim=-1)
        res = q_fft * torch.conj(k_fft)
        corr = torch.fft.irfft(res, dim=-1)

        # Channel agg
```

```
        V, index, new_index = self.correlation_removal_training(values.
      permute(0,2,1,3).contiguous(), corr)

        return V.permute(0,2,1,3).contiguous(), index, new_index


class CrossCorrelation_layer(nn.Module):
  def __init__(self, correlation, d_model, num_heads):
    super(CrossCorrelation_layer, self).__init__()

    d_keys = d_model // num_heads
    d_values = d_model // num_heads

    self.correlation = correlation
    self.query_projection = nn.Linear(d_model, d_keys * num_heads)
    self.key_projection = nn.Linear(d_model, d_keys * num_heads)
    self.value_projection = nn.Linear(d_model, d_keys * num_heads)
    self.out_projection = nn.Linear(d_keys * num_heads, d_model)
    self.num_heads = num_heads
    self.norm = nn.LayerNorm(d_values * num_heads)

  def forward(self, queries, keys, values):
    B, C, L = queries.shape
    H = self.num_heads

    values_res = values
    queries = self.query_projection(queries).view(B, C, H, -1)
    keys = self.key_projection(keys).view(B, C, H, -1)
    values = self.value_projection(values).view(B, C, H, -1)

    out, index, new_index = self.correlation(
        queries,
        keys,
        values
    )

    out = out.view(B, -1, L)
```

```
        out = self.out_projection(out)
        res = torch.cat((torch.zeros(B,1,L).to(device), values_res.index_select
        (-2, new_index)), dim=-2)
        out = self.norm(out + res)

        return out



class PositionwiseFeedForward(nn.Module):
    '''
    2 feed forward layer module
    '''

    def __init__(self, d_in, d_hid, dropout=0.05):
        super().__init__()
        self.w_1 = nn.Linear(d_in, d_hid)
        self.w_2 = nn.Linear(d_hid, d_in)
        self.layer_norm = nn.LayerNorm(d_in, eps=1e-6)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):

        residual = x

        x = self.w_2(F.relu(self.w_1(x)))
        x = self.dropout(x)
        x += residual

        x = self.layer_norm(x)

        return x


class CrossCorrelationBlock(nn.Module):
    def __init__(self, num_topk, d_model, num_heads, attention_dropout=0.1):
        super(CrossCorrelationBlock, self).__init__()
        self.num_topk = num_topk
        self.d_model = d_model
        self.num_heads = num_heads
        self.attention_dropout = attention_dropout

        self.cross_correlation = CrossCorrelation(
            num_topk=self.num_topk,
            d_model=self.d_model,
            num_heads=self.num_heads
            )

        # CrossCorralation layer
        self.cross_corr_layer = CrossCorrelation_layer(self.cross_correlation, d_model=self.d_model, num_heads=self.num_heads)
        self.feedforward = PositionwiseFeedForward(d_in=self.d_model, d_hid=4*self.d_model)

    def forward(self, x):
        x = self.cross_corr_layer(x,x,x)
        x = self.feedforward(x)
        return x


class Attention(nn.Module):
    def __init__(self, d_model, num_heads, qkv_bias=True, qk_scale=None, attn_drop=0., proj_drop=0., position_bias=True):
        super().__init__()
        self.d_model = d_model
        self.num_heads = num_heads
        head_dim = d_model // num_heads
        self.scale = qk_scale or head_dim ** -0.5
        self.position_bias = position_bias

        self.qkv = nn.Linear(d_model, d_model *3, bias = qkv_bias)
        self.attn_drop = nn.Dropout(attn_drop)
        self.proj = nn.Linear(d_model, d_model)
        self.proj_drop = nn.Dropout(proj_drop)
```

```python

    self.softmax = nn.Softmax(dim=-1)
    self.norm = nn.LayerNorm(self.d_model)
    self.norm1 = nn.LayerNorm(self.d_model)

    self.mlp = PositionwiseFeedForward(d_in=self.d_model, d_hid=self.d_model * 4)

  def forward(self, x, mask=None):
    B_, N, C = x.shape
    qkv = self.qkv(x).reshape(B_, N, 3, self.num_heads, C // self.num_heads).permute(2, 0, 3, 1, 4)
    q, k, v = qkv[0], qkv[1], qkv[2]  # make torchscript happy (cannot use tensor as tuple)
    q = q * self.scale
    attn = (q @ k.transpose(-2, -1))
    attn = self.softmax(attn)


    #print(attn.shape)
    res = x
    x = (attn @ v).transpose(1, 2).reshape(B_, N, C)
    x = self.proj(x)
    x = self.proj_drop(x)
    x = self.norm(x + res)

    res_mlp = x
    x = self.mlp(x)
    x = self.norm1(x + res_mlp)
    return x


class Sensorformer_FFT(nn.Module):
  def __init__(self, decorr_depth, attn_depth, num_class, d_model, num_heads, num_heads_attn, num_topk, orth_gain, attention_dropout=0.1):
    super().__init__()

    self.d_model = d_model
    self.num_heads = num_heads
    self.num_heads_attn = num_heads_attn
    self.num_topk = num_topk
    self.decorr_depth = decorr_depth
    self.attn_depth = attn_depth
    self.num_class = num_class
    self.cls_token = torch.randn(1,d_model).to(device)

    self.decorr = nn.ModuleList()

    for i in range(self.decorr_depth):
      self.decorr.append(CrossCorrelationBlock(num_topk=self.num_topk,
   d_model=self.d_model, num_heads=self.num_heads))

    self.attn = nn.ModuleList()

    for i in range(self.attn_depth):
      self.attn.append(Attention(d_model=self.d_model, num_heads=self.
   num_heads_attn))

    self.classifier = nn.Linear(self.d_model, self.num_class)

    # initialisation
    for p in self.parameters():
      if p.dim() > 1:
        nn.init.xavier_uniform_(p)

    if orth_gain is not None:
      torch.nn.init.orthogonal_(self.classifier.weight, gain = orth_gain)
    self.classifier.bias.data.zero_()


  def forward(self, x):
    B, _, _ = x.shape
    for i in range(self.decorr_depth):
      x = self.decorr[i](x)

    #print(x.shape)

    x = torch.cat((self.cls_token.unsqueeze(0).repeat(B,1,1), x), dim=-2)
    for i in range(self.attn_depth):
      x = self.attn[i](x)

    x = x[:, 0, :]
    x = self.classifier(x)

    return x


model = Sensorformer_FFT(
    decorr_depth=4,
    attn_depth=8,
    num_class=3,
    d_model=60,
    num_heads=2,
    num_heads_attn=6,
    num_topk=101,
    orth_gain=1.41)
model.to(device)
def get_p(net):
  t = sum(p.numel() for p in net.parameters())
  trainable = sum(p.numel() for p in net.parameters() if p.requires_grad)
  return t, trainable
print(get_p(model))
# for p in model.parameters():
#     p.register_hook(lambda grad: torch.clamp(grad, -clip_value, clip_value)
    )



batch_size = 30
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size,
    shuffle=True, drop_last=True)
```

```python
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size,
    shuffle=True, drop_last=True)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(), lr=0.0001)

def train(epoch):

    correct1 = 0
    total1 = 0
    running_loss = 0

    epoch_running_loss = 0
    epoch_correct1 = 0
    epoch_total1 = 0

    for batch_idx, data in enumerate(train_loader):
        images, labels = data
        optimizer.zero_grad()
        outputs = model(images)
        outputs = outputs.squeeze(1)
        loss = criterion(outputs, labels)
        loss.backward()
        #torch.nn.utils.clip_grad_norm_(model.parameters(), args.clip)
        optimizer.step()

        running_loss += loss.detach().cpu().item()
        epoch_running_loss += loss.detach().cpu().item()

        idx_control = (train_size // batch_size) // 4

        if (batch_idx + 1) % idx_control == 0:
          train_loss_list.append(running_loss / batch_size / idx_control)
          #print('[%d, %d], training loss is %.3f' % (epoch, batch_idx,
    running_loss / batch_size / idx_control))
          running_loss = 0

        if (batch_idx + 1) % (train_size // batch_size) == 0:
          epoch_train_loss_list.append(epoch_running_loss / (train_size // batch_size * batch_size))


        _, predict1 = torch.max(outputs, dim=1)
        predict1 = predict1.detach().cpu()
        labels = labels.detach().cpu()
        correct1 += (labels == predict1).sum().item()
        total1 += labels.size(0)

        epoch_correct1 += (labels == predict1).sum().item()
        epoch_total1 += labels.size(0)

        if (batch_idx + 1) % idx_control == 0:
          train_accuracy_list.append(100 * (correct1 / total1))
          #print('Train acc: ', 100 * (correct1 / total1))
          correct1 = 0
          total1 = 0

        if (batch_idx + 1) % (train_size // batch_size) == 0:
          epoch_train_accuracy_list.append(100 * (epoch_correct1 / epoch_total1))
          print('Epoch train acc: ', 100 * (epoch_correct1 / epoch_total1))



def test():
    correct = 0
    total = 0
    test_loss = 0

    epoch_correct = 0
    epoch_total = 0
    epoch_test_loss = 0

    with torch.no_grad():
        for batch_idx, data in enumerate(test_loader):
```

```
        images, labels = data
        outputs = model(images)
        outputs = outputs.squeeze(1)
        _, predict = torch.max(outputs, dim=1)
        predict = predict.detach().cpu()
        labels = labels.detach().cpu()
        correct += (labels == predict).sum().item()
        total += labels.size(0)

        epoch_correct += (labels == predict).sum().item()
        epoch_total += labels.size(0)

        idx_control = (test_size // batch_size) // 4

        if (batch_idx + 1) % idx_control == 0:
          test_accuracy_list.append(100 * (correct / total))
          #print(’correct/total:%d/%d, Test Accuracy:%.2f%%’ % (correct, total, 100 * (correct / total)))
          correct = 0
          total = 0

        if (batch_idx + 1) % (test_size // batch_size) == 0:
          epoch_test_accuracy_list.append(100 * (epoch_correct / epoch_total))
          print(’Epoch test acc: ’, 100 * (epoch_correct / epoch_total))


        outputs = outputs.detach().cpu()
        loss = criterion(outputs, labels)

        test_loss += loss.detach().cpu().item()
        epoch_test_loss += loss.detach().cpu().item()

        if (batch_idx + 1) % idx_control == 0:
          test_loss_list.append(test_loss / batch_size / idx_control)
          #print(’[%d, %d], test loss is %.3f’ % (epoch, batch_idx,
test_loss / batch_size / idx_control))
                test_loss = 0

            if (batch_idx + 1) % (test_size // batch_size) == 0:
                epoch_test_loss_list.append(epoch_test_loss / batch_size /
idx_control)


if __name__ == '__main__':
    train_loss_list = []
    test_accuracy_list = []
    train_accuracy_list = []
    test_loss_list = []
    train_epoch = 20
    train_time = []

    epoch_train_loss_list = []
    epoch_train_accuracy_list = []
    epoch_test_loss_list = []
    epoch_test_accuracy_list = []



    for epoch in range(train_epoch):
        print('epoch: ', epoch)
        model.train()

        start_time = time.time()
        train(epoch)
        end_time = time.time()
        epoch_time = (end_time - start_time)
        train_time.append(epoch_time)


        model.eval()
        test()

```

```python


print('trian time per epoch: ', mean(train_time))


y1 = test_accuracy_list
y2 = train_loss_list
y3 = train_accuracy_list
y4 = test_loss_list

y11 = epoch_test_accuracy_list
y22 = epoch_train_loss_list
y33 = epoch_train_accuracy_list
y44 = epoch_test_loss_list

plt.subplot(4, 1, 1)
plt.plot(y1, 'o-')
plt.xlabel('Test acc vs. epochs')
plt.ylabel('Test accuracy')

plt.subplot(4, 1, 2)
plt.plot(y3, '.-')
plt.xlabel('Train acc vs. epochs')
plt.ylabel('Train acc')

plt.subplot(4, 1, 3)
plt.plot(y4, '.-')
plt.xlabel('Test loss vs. epochs')
plt.ylabel('Test loss')

plt.subplot(4, 1, 4)
plt.plot(y2, 'o-')
plt.xlabel('Train loss vs. epochs')
plt.ylabel('Train loss')

plt.show()

```

```



    plt.subplot(4, 1, 1)
    plt.plot(y11, 'o-')
    plt.xlabel('Test acc vs. epochs')
    plt.ylabel('Test accuracy')

    plt.subplot(4, 1, 2)
    plt.plot(y33, '.-')
    plt.xlabel('Train acc vs. epochs')
    plt.ylabel('Train acc')

    plt.subplot(4, 1, 3)
    plt.plot(y44, '.-')
    plt.xlabel('Test loss vs. epochs')
    plt.ylabel('Test loss')

    plt.subplot(4, 1, 4)
    plt.plot(y22, 'o-')
    plt.xlabel('Train loss vs. epochs')
    plt.ylabel('Train loss')

    plt.show()
```

Listing D.1: Sensorformer: A Memory-efficient Transformer for Industrial Sensor Fusion