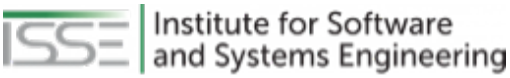
TU Clausthal
Clausthal University of Technology



Master Thesis

Deep Learning-based Simultaneous Fault Detection and Classification for Recordings Analysis of Hardware-in-the-Loop Tests

VISHWANATHA REDDY, HARSHINI

Matriculation Number: 525712

MSc Informatik

Erstgutachter: Prof. Dr. Andreas Rausch

Zweitgutachter: Dr. Christoph Knieke

Betreuer: MSc. Mohammad Abboush

Institute for Software and Systems Engineering

Technische Universität Clausthal - Clausthal University of Technology, Germany

July 19, 2023

Declaration of Authorship

I have read and understood the guidelines of the Clausthal University of Technology. I confirm that I have prepared this master thesis independently by myself. Any information taken from other sources and being reproduced in this thesis is clearly referenced.

In terms of the general examination regulations, this work has not yet been submitted to any other examination division.

I hereby agree that my master thesis may be exhibited in the institute's or university library and kept for inspection.

Clausthal-Zellerfeld, July 19, 2023	Vishwanatha Reddy, Harshini
Location and Date	Last Name, First Name

Abstract

Functional validation of complex Automotive Software Systems (ASSs) is important to avoid any upcoming dangerous consequences. Hardware-in-the-Loop (HIL) is recommended By ISO 26262 as a test bench to ensure the safety aspects of such complex systems. Failure is common occurrence in any system which are hard to analyze and more often is treated manually. Most of the failures occur due to associated faults. Faults can be either single or simultaneous fault. Simultaneous fault is the scenario where two or more faults raise during the same time but in different location. In this study, simultaneous fault with two different kinds of the faults is considered for detection and classification. Data driven method is associated with many machines learning techniques which enable to take decision based on data generated from system rather than based in intuition. An intelligent novel technique for Simultaneous Fault Detection and Classification (SFDC) is proposed based on hybrid machine learning algorithms.

Once the imbalanced data is collected from HIL through the fault injection method, data undergoes preprocessing. Before data preprocessing auto balanced technique called random undersampling is applied to avoid training the model in biased way. SFDC model is designed such a way that it can also deal with noise in the data. Denoising Autoencoder (DAE) model is used to remove the noise from the data. Firstly, long short-term memory (LSTM) model alone is used for both feature extraction and classification. For second model, LSTM is used for feature extraction and random forest (RF) is used for classification. Since both models mentioned before are well-known for their performance as classifiers, voter is used to combine predictions from parallel models, LSTM (feature extraction + classification) and LSTM (feature extraction) + RF (classification). High performance of 92% F1 score for single faults and 85.6% F1 score for simultaneous faults are achieved. Good balanced accuracy of highest 95.5% is obtained by LSTM+ RF+ voter. The SFDC model represents a significant advancement in addressing the challenges of simultaneous fault analysis, showcasing its potential to contribute to safer and more robust automotive systems.

Zusammenfassung

Die funktionale Validierung komplexer Automotive Software Systems (ASSs) ist wichtig, um gefährliche Folgen zu vermeiden. Hardware-in-the-Loop (HIL) wird von ISO 26262 als Prüfstand empfohlen, um die Sicherheitsaspekte solcher komplexen Systeme zu gewährleisten. In jedem System treten häufig Fehler auf, die schwer zu analysieren sind und häufig manuell behandelt werden. Die meisten Ausfälle treten aufgrund von Fehlern auf. Bei den Fehlern kann es sich entweder um einzelne oder gleichzeitige Fehler handeln. Ein gleichzeitiger Fehler ist ein Szenario, bei dem zwei oder mehr Fehler zur gleichen Zeit, aber an unterschiedlichen Orten auftreten. In dieser Studie wird ein gleichzeitiger Fehler mit zwei verschiedenen Arten von Fehlern für die Erkennung und Klassifizierung berücksichtigt. Die datengesteuerte Methode wird mit vielen maschinellen Lerntechniken in Verbindung gebracht, die es ermöglichen, Entscheidungen auf der Grundlage von Daten zu treffen, die vom System generiert werden, anstatt auf der Grundlage von Intuition. Es wird eine intelligente neue Technik zur gleichzeitigen Fehlererkennung und Klassifizierung (SFDC) vorgeschlagen, die auf hybriden maschinellen Lernalgorithmen basiert.

Sobald die unausgewogenen Daten von der HIL durch die Fehlerinjektionsmethode gesammelt wurden, werden die Daten vorverarbeitet. Vor der Datenvorverarbeitung wird die Technik des automatischen Ausgleichs (random under-sampling) angewandt, um ein verzerrtes Training des Modells zu vermeiden. Das SFDC-Modell ist so konzipiert, dass es auch mit Rauschen in den Daten umgehen kann. Das Modell Denoising Autoencoder (DAE) wird verwendet, um das Rauschen aus den Daten zu entfernen. Erstens wird das LSTM-Modell (Long Short Memory) allein für die Merkmalsextraktion und die Klassifizierung verwendet. Beim zweiten Modell wird LSTM für die Merkmalsextraktion und Random Forest (RF) für die Klassifizierung verwendet. Da beide vorgenannten Modelle für ihre Leistung als Klassifikatoren bekannt sind, werden die Vorhersagen von parallelen Modellen, LSTM (Merkmalsextraktion + Klassifikation) und LSTM (Merkmalsextraktion) + RF (Klassifikation), kombiniert. Es wird eine hohe Leistung von 92% F1-Score für einzelne Fehler und 85,6% F1-Score für gleichzeitige Fehler erreicht. Eine ausgewogene Genauigkeit von 95,5%

wird durch LSTM+ RF+ Wähler erreicht. Das SFDC-Modell stellt einen bedeutenden Fortschritt bei der Bewältigung der Herausforderungen der simultanen Fehleranalyse dar und zeigt, dass es zu sichereren und robusteren Automobilsystemen beitragen kann.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context and Motivation

With regards to the Advanced Driver Assistance Systems (ADAS) approving the right activity of situation level capabilities of the system like advanced braking system, carto-X (C2X) communication, and gaze detection has become a challenge. To be specific increasing automotive software systems (ASSs) feature dependencies has made check and approval concepts complex [Lotz et al., 2019].

Hardware-in-the-Loop (HIL) is a test bench which is employed for system integration level validation to conduct a comprehensive evaluation of a system under controlled conditions that closely mimic its real-world usage [Vishnukumar et al., 2017]. This procedure seeks to uncover any imperfections or unusual behaviour within the system that could potentially result in unintended or dangerous consequences. HIL is not the only test bench there are other platforms like SIL (Software-in-the-Loop), PIL (Processor-in-the-Loop), MIL (Model-in-the-Loop) and VIL (Vehicle-in-the-Loop) [Drenth et al., 2014]. There are different test methods designed by engineers to test the system within HIL. Test cases are executed and validated based on the large data that is generated by HIL. The data not only possesses a substantial volume but also exhibits a high degree of complexity, thereby representing diverse system behaviours. It is worth noting that data-driven methods come up with innovative approaches for ASSs. Data-driven methods leverage on abilities of machine learning and data analysis techniques. Tasks like fault detection, diagnosis and classification in ASS can be performed by employing huge data produced by HIL and intelligent models of machine learning or deep learning.

1.2 Problem Statement

Simultaneous fault detection and classification in ASSs can be a complex and challenging task that requires advanced techniques. Most of the traditional methods have only focused on detecting and classifying single faults, but this is no longer sufficient as it can lead to overlooking other underlying issues [Abboush et al., 2022b], [Kazemi et al., 2021]. There could be chances that noises will be introduced into faults either by mechanical or electronic disturbances[Lee et al., 2016]. However, due to this noise faulty data might differentiate from the noise-free faulty data. There could be chances of having unbalanced data when only certain class fault data is majorly available compared to others. If features are extracted from the unbalanced data directly there could be chances of model learning inaccurately [Zhou et al., 2020].

To address this concern, our study aims to develop and implement a model that leverages data-driven methods to identify and classify various types of faults in ASSs. Our approach tackles the complexity of simultaneous fault patterns by utilizing advanced deep-learning techniques and selecting the most appropriate algorithms. With our cutting-edge approach, we can achieve simultaneous fault detection and classification, which can significantly improve the reliability, efficiency, and affordability of fault analysis in ASSs. This research will contribute significantly to the advancement of fault analysis methods and their successful application in the real world.

As a result of our study, we can gain a deeper understanding of the complex fault patterns that occur in ASSs. This knowledge can help us to develop more effective solutions that can accurately detect and classify simultaneous faults in real time. Our research will also provide valuable insights into the behaviour of HIL test systems, which can help industry professionals to improve the design and implementation of these systems. Ultimately, our innovative approach can help to ensure the safety, efficiency, and reliability of ASSs which are critical components of many industries. Simultaneous faults as data mirror different challenging characteristics:

Enhanced Complexity

Simultaneous faults exhibit heightened complexity compared to single sensor faults, stemming from the simultaneous occurrence of multiple faults in diverse sensors. This intricate combination introduces intricate patterns and interactions within the data, rendering its analysis and interpreta-

tion considerably more demanding.

Balance between Fault Types

Unlike single faults simultaneous faults contain combination of different fault types it would be challenged to understand the balance between these faults. There are high chances that simultaneous faults can be imbalanced in ratio of one fault type to the other.

Effects of Interconnected Fault

The existence of one problem affects the behavior or expression of another fault, according to the concept of interconnected fault effects. This increased difficulty in effectively identifying and categorizing faults is a result of the interrelated fault effects that cause unanticipated variances in the data.

Fault Pattern Ambiguity

Simultaneous faults introduce ambiguity into the data patterns, making it arduous to discern between normal and faulty signals. The overlapping effects and combination of various fault types impede precise fault diagnosis, potentially resulting in erroneous interpretations of the data.

Requirement for Advanced Analysis Techniques

The presence of simultaneous faults necessitates the development and application of advanced analysis techniques. Traditional fault detection and diagnosis methods may prove inadequate in handling the complexity and interdependencies associated with simultaneous faults. Employing advanced algorithms, machine learning, and statistical techniques becomes imperative to effectively identify and mitigate these intricate fault scenarios.

1.3 Research Questions

This work aims to answer the main research question.

RQ1: How to detect and classify simultaneous faults using a machine learning model?

The RQ1 is then further classified into two sub-questions:

RQ1.1: What would be the performance difference of the model between single and simultaneous fault detection and classification?

RQ1.2: What are the suitable models that can be used for the SFDC task?

RQ2: How to develop a model to handle imbalanced data and noise in data?

The RQ2 is then further classified into two sub-questions:

RQ2.1: How robust is the DAE model towards different levels of noise?

RQ2.2: Will the increase and decrease of healthy data have an impact on model performance?

1.4 Related Work

Over the past decade, fault detection and classification using data-driven method has gained significant attention. Utilizing the strength of deep learning algorithms, researchers have investigated numerous deep learning techniques to approach these difficult issues like simultaneous fault detection and classification. Research works on faults like fault detection, classification, diagnosis, or identification have been advantageous to different fields like the automotive system, the power industry, the aerospace sector [Chen et al., 2020], the manufacturing [Fan et al., 2020] and the renewable energy sector [Shakya, 2021].

[Ince et al., 2016] as used real-time data to detect the faults and to be more particular 1-D CNN which is a data-driven method is used for the diagnosis. Using real motors experimental results are collected. Authors in the paper [Ou et al., 2014] emphasize the advantages of real-time digital simulators in providing high-fidelity real-time simulation capabilities, allowing for dynamic studies and hardware-in-the-loop (HIL) testing. The integration of physical control systems enables the testing of actual control algorithms and hardware components, enhancing the realism and reliability of the simulation results. also highlighted the benefit of using HIL simulation in automotive diagnostic control systems. HIL can be used to reduce the time and cost factors during the testing phase. Automotives use a huge number of sensors and actuators in the electronic control unit (ECU) for

control. Also, a huge number of control functions are used to benefit in terms of emission and fuel consumption which is most important for the current era.

In [Palladino et al., 2012] authors used HIL for fault injection. In this research, HIL was used to simulate a real-time traction control system (TCS). The pipeline of the timing error is injected as a fault using HIL and static timing analysis is used to diagnose the faults. To validate the proposed methodology HIL results are used at the end. [Yang et al., 2021] represented how fault detection and fault diagnosis is conducted using HIL which simulates high-speed trains. Signal-based injection methods are used to inject the faults into HIL. Fault diagnosis is carried out with the help of a data-driven method and is internally classified into 2 parts: fault detection and fault diagnosis.

Authors in [Alvarez-Gonzalez et al., 2018] focused on the model-based fault detection method used by understanding the difference between the current measurements and real-time predicted values using the machine models. For validation of obtained results, the authors have used computer simulations and HIL. [Fragkoulis et al., 2009] focuses on a model-based approach to detect and isolate single, multiple and simultaneous faults. This study also mentioned focusing on the different combinations of faults will help for faster response to abrupt faults.

In [Dutta et al., 2022]have proposed machine learning models, artificial neural network (ANN) and ANFIS (ANN and Fuzzy) for fault detection. HIL simulator is used for result verification. Different machine learning models are tested for performance and ANN and ANFIS had good accuracy, prediction speed, and training time when compared to other models hence ANN and ANFIs was concluded as a better model. [Abboush et al., 2022b]Intelligent deep learning models are used in research to detect and classify single faults in the HIL test environment. A hybrid model with a combination of CNN and LSTM has obtained good accuracy of 98.8% fault detection and classification.

[Breiman, 2001] Proposed a consolidated type of multiple decision trees which is called random forest. In [Chakraborty et al., 2019] researchers proposed a random forest for fault classification in active power systems against other models like KNN, SVM, logistic regression and linear regression. Random forest was proved to be the best classifier with the highest accuracy of 97%. Researchers in [De Bruin et al., 2016] proposed a deep learning model LSTM for fault diagnosis of railway track circuits. Higher accuracy of 99.7% was achieved over the CNN model.

Multi-fault classification work in [Hua et al., 2023] proposed a deep learning model, light gradient boosting machine (LightGBM) and variational mode decomposition (VMD) method. The average accuracy of 93% is achieved and different levels of noise level are added. Parameter-optimized VMD method is used for denoising trajectory angle and LightGBM is used for classification. But noise-free and balanced data was considered in the work.

[[Chiang et al., 2019] have proposed a fully convolutional network-based denoising autoencoder to denoise the noise in the ECG signals. Performance of CNF-based DAE is measured in terms of root mean square error (RMSE), percentage root mean square difference (PRD) and signal-to-noise ratio (SNR). In terms of all the performance matrices FCN based DAE performed better than CNN and DNN. This work proves that FCN-based DAE is best suited for denoising data or reconstructing noisy data into noise-free data. However, still, denoising was applied to naturally balanced data.

Authors [Khuat and Le, 2019] highlighted the problem of having a disproportionate distribution of fault and non-fault instances in software datasets, which can negatively impact the performance of fault prediction models. The experiments conducted by the authors demonstrate the effectiveness of the random under-sampling technique to mitigating the impact of imbalanced data and achieving better predictive performance compared to traditional classification models. [Zhou et al., 2021]uses data-driven methods for fault diagnosis on imbalanced data. PCA-SMOTE is used to deal with the problem of imbalanced data. In comparison with other techniques, PCA-SMOTE has gained better accuracy with another model for fault diagnosis.

1.5 Solution

In this work supervised machine learning models are used to detect and classify the simultaneous faults. Not only simultaneous faults are detected and classified even single faults are detected and classified. To develop intelligent model data is collected from HIL real-time fault injection framework. Data collected is imbalanced and noise-free data. For classification of both single and simultaneous faults using same model technique called powerset labeling is used in data preprocessing steps. Since the data is imbalanced feeding data without auto-balancing will train the machine learning model in a biased manner hence auto balancing technique called random under-sampling

Reference	Manual Noise	Application	Solution	Fault/s	Proposed Algorithms	Accuracy
[Namburu et al., 2007]	No	Automotive Engines	Fault diagnosis	Single fault	SVM	99.25%
[Dutta et al., 2022]	No	Pumping system	Fault detection	Single fault	ANN	99.6%
[Abboush et al., 2022b]	No	Advanced software systems	Fault detection and classification	Single fault	CNN+ LSTM	98.8%
[Chakraborty et al., 2019]	No	Active power system networks	Fault classification	Single fault	Random forest	97%
[De Bruin et al., 2016]	No	Railway track circuit	Fault diagnosis	Single fault	LSTM	99.7%
[Hua et al., 2023]	Yes	Rotor system	Fault Classification	Simultaneous fault	LightGBM	93%
[Zhou et al., 2021]	No	Variable refrigerant flow system	Fault diagnosis	Single fault	SVM, BPNN	100%

Table 1.1: Related Work

is used in the thesis. To develop a robust DAE model for the task of data denoising intentionally different levels of noise are added to the data and DAE will be trained to denoise it.

1.6 Objective

The objective of the work is to develop deep learning model that can classify single and simultaneous fault types of sensor control signals using real time test system level data of ASS. Dedicated approach and model should be developed to deal with noise and imbalanced issue in the data. Since single and simultaneous faults are 2 different kinds of data effective machine learning model should be developed to classify both faults effectively.

1.7 Structure

The work is broken into six chapters and a short description of each chapter is as follows:

- Chapter 1 provides the necessary introduction to the problem addressed in the work. Research question that will be findings at the end of the work is addressed in same section. Different studies are considered and compared in field of fault detection and classification. Solution to the research problem is stated with proposed methods. Objective of the work which will state the solution to the problem is mentioned at the end with structure overview.

- Chapter 2 starts by discussing approaches used in the work. Background knowledge of different related things are also described. Different machine learning algorithms are stated to select the appropriate algorithm to develop SFDC model development. Chapter 3 explains proposed methodology from data collection process to development of SFDC model.

- Chapter 4 describes in detail about case study used in the thesis work. Clear implementation details from very beginning of fault injection to using different techniques to overcome issues in data like imbalanced data and noise in the data. Development of most suitable machine learning model to SFDC with hyperparameter tuning is also shown.

- Chapter 5 described the performance of the proposed model with different performance metrics and evaluation of each technique is explained.

- Chapter 6 recapitulates and concludes the work with future scope.

2 Background

This chapter will emphasize on knowledge and concepts needed for understanding the context of the topic.

2.1 Model-based Development Phases

In the software development process after software requirements and design, the next step would be the design of model-based specifications. Later this model-based specifications design will be tested or validated in the ECU in certain realistic conditions. The below section explains different test benches in detail.

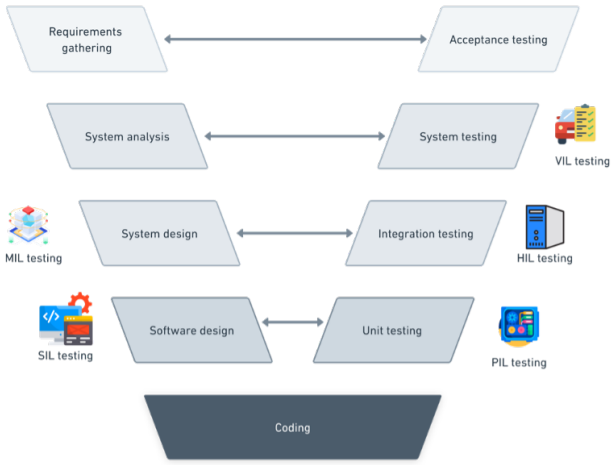


Figure 2.1: V-Cycle of Model-Based Software Development Process

2.1.1 Model-in-the-Loop (MIL)

Mathematical models of the complete vehicle are characteristic of a MIL phase in the development process i.e., vehicle operating external environment, ECU software models, engine physical models, and transmission components. Due to significant improvements in mathematical modelling tools and approaches over the past 20 years, a wide range of models are now available to

accommodate different types of ECU software development [Amalfitano et al., 2014]. This is a crucial step because it provides designers with information such as the mechanical construction of the machine's necessary parameters, including its dimensions, materials, number of actuators, type of sensors, etc., in addition to a mathematical model of the process. Testing the software model architecture and the functions themselves in a hardware-independent way and in a closed-loop with the controlled plant is the main objective of MIL development.

2.1.2 Software-in-the-Loop (SIL)

Required software or machine process begins with software-in-the-loop design simulation. In order to replicate the entire ECU, SIL uses the real ECU code and basic software [Lumpp et al., 2014]. SIL simulation offers a crucial advantage by enabling the assessment of a control algorithm's performance within its intended environment and facilitating the identification of suitable target hardware like target OS, target processor and amount of memory.

2.1.3 Processor-in-the-Loop (PIL)

The real target device and simulation PC are connected by industrial communication in the Processor-in-the-Loop simulation model. The model and control method are identical to those used in SIL simulation, which is the main benefit. The target device only receives an automated translation of the control algorithm. PIL enables the verification of controller behaviour in actual hardware [Molina Llorente and Molina Llorente, 2020]. Additionally, as flow testing also utilizes a simulation environment, controller real-time behaviour cannot be evaluated in PIL.

2.1.4 Hardware-in-the-Loop (HIL)

The control hardware is completely outfitted with all required I/O modules as part of the HIL, which is the final stage in the validation process. All inputs and outputs are connected to testing that has a counterpart for each I/O. The simulation model in the centre of the testing setup simulates all I/O signals for the hardware being tested. Using this method, all I/O modules, together with the control algorithm or sensors, are tested [Ahmad et al., 2020].

2.1.5 Vehicle-in-the-Loop (VIL)

As the last step, VIL is used for system validation in a real environment. In order to establish a closed-loop testing environment that accurately represents real-world driving situations, VIL entails merging virtual or actual automobiles with simulation models and real-time hardware. VIL enables early detection and resolution of possible problems by allowing automotive engineers to evaluate the behaviour and interaction of multiple vehicle subsystems, including the powertrain, chassis, and driver aid systems, under various situations. Because VIL enables complete testing and validation of algorithms, sensors, and control techniques in a safe and controlled environment prior to undertaking real-world road tests, it is particularly useful in the development of ADASs and autonomous cars [Li et al., 2023].

Matlab/Simulink

The widely used software platform MATLAB offers a complex computing environment for applications such as interactive programming, data analysis, and scientific computations. Simulink, which is a crucial component of Matlab, provides a powerful toolkit for system development using a model-based design methodology. Engineers can effectively design, simulate, and evaluate dynamic systems with Simulink by employing block diagrams and graphical modelling methods. Simulink also enables crucial system engineering processes such as system design, simulation, automated code generation, continuous testing, and embedded system verification. The most popular option for Model-in-the-Loop (MIL) testing, in which software components are tested against simulation models before implementation, is Matlab/Simulink due to its extensive capabilities and features. Engineers may increase the effectiveness, precision, and dependability of their MIL testing procedures by using Matlab/Simulink, which will eventually result in better software quality and system performance [Mihalič et al., 2022].

2.2 Fault Taxonomy

Faults are majorly classified into three types—permanent, intermittent, and transient which are distinguished based on their origin and behaviours. Permanent faults are physical alterations that

cannot be reversed [Park et al., 2015]. A device that is unstable or subpar causes intermittent problems, which can be triggered by environmental variations like temperature and voltage swings. Environmental circumstances that are only momentary give rise to transient faults [Park et al., 2015]. Going further in this section we will discuss more about faults.

2.2.1 Fault Types

This section discusses different kinds of faults based on the impact they cause on the output. Each kind of fault poses different difficulties and has the potential to affect the precision and dependability of sensor readings:

Stuck Fault

Struck fault occurs when the output value remains constant and fails to change or when the sensor is stuck at a particular value [Gong et al., 2019]. This fault can also be called a complete failure [Yu et al., 2015]. An example would be a temperature sensor that consistently reports the same value regardless of the original temperature.

Offset/Bias Fault

Offset or Bias fault is bias or deviation from the sensor output [Fabarisov et al., 2021]. Output in this kind of fault will differ from the true values by a fixed amount. For example, an accelerometer is constantly reporting values in the difference of +0.5 m/s^2.

Drift Fault

Even in the absence of changes in the quantity being measured, a sensor's output will gradually shift (increase or decrease) over time, which is called drift fault [Gong et al., 2019] [Gong et al., 2019]. As a result of the sensor's output progressively deviating from its initial calibration, results might be inaccurate. A pressure sensor that displays a gradual but consistent shift in pressure measurements over time is an example.

Noise Fault

The term "noise fault" describes the existence of random fluctuations or interference in the sensor output those results in inaccurate measurements [Safavi et al., 2021]. Electrical interference, environmental conditions, or built-in sensor technology restrictions are all potential causes. Noise flaws can induce uncertainty and reduce measurement accuracy [Gong et al., 2019].

Saturation Fault

When a sensor hits its maximum or minimum measurable value and is unable to continue to deliver correct readings, this is known as a saturation fault. When the sensor's measurement range is exceeded by the quantity being measured, this might occur [Laska et al., 2011]. Consider a light sensor that, when exposed to strong light, saturates, and gives the highest reading possible.

Gain Fault

A gain failure happens when the sensor's output is scaled improperly, departing from the predicted correlation between the output and the measured amount.

Hard-Over Fault

Failure or interruption in the flow of sensor data across various systems or components results in hand-over problems [Safavi et al., 2021].

Spike Fault

An abrupt and brief disturbance or irregularity in the sensor output that causes a rapid and fleeting departure from the predicted values is referred to as a spike fault.

Delay Time Fault

The sensor signal's delay time is the amount of time it takes to travel from the sensor to the output. Signal processing, communication protocols, and system reaction time can all have an impact on it [Safavi et al., 2021].

Packet Loss Fault

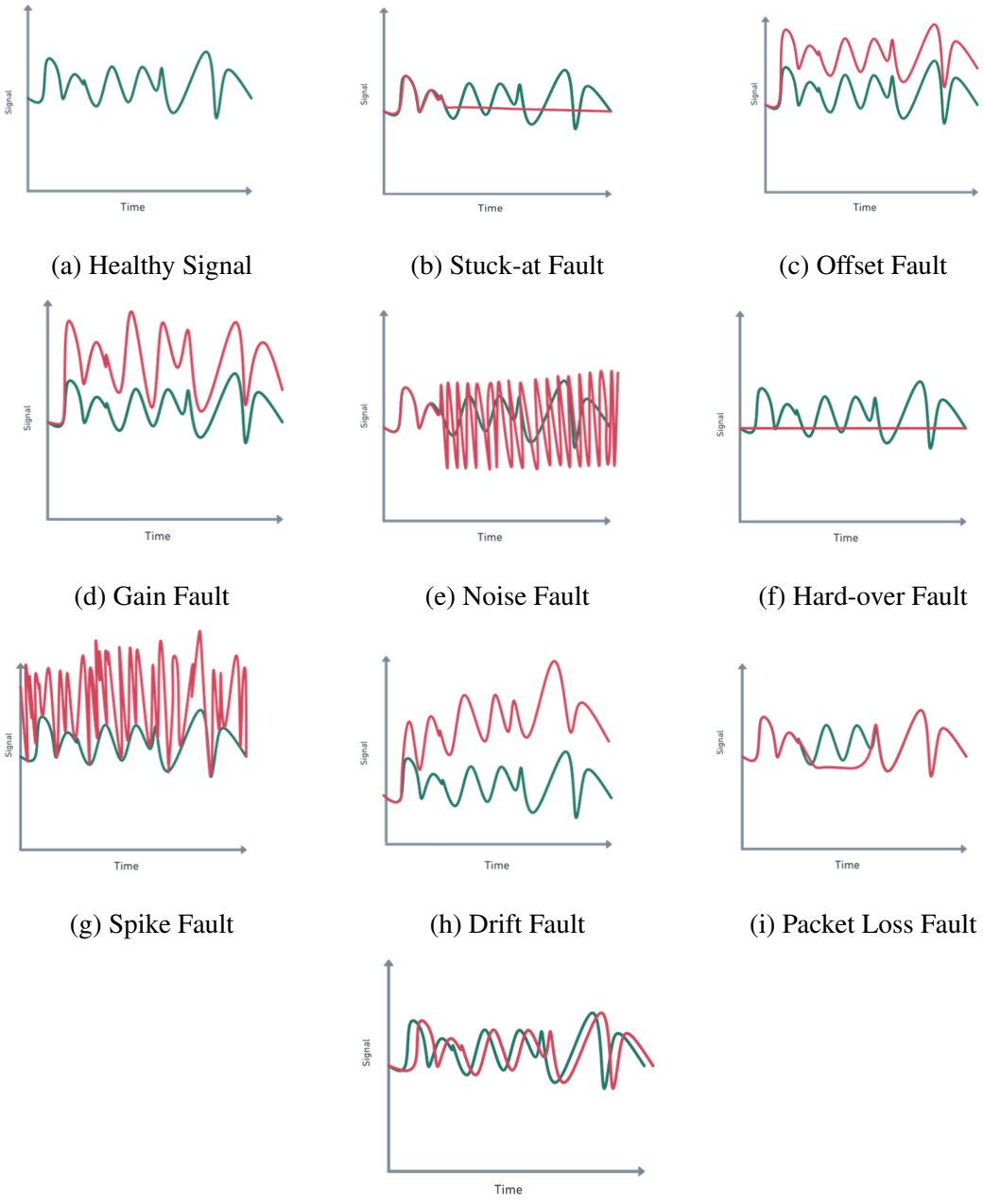
When data packets carrying sensor data are not correctly received or processed by the intended destination, a package loss fault occurs [Safavi et al., 2021]. In the figure, 2.1 package loss refers to the probability (P) of a data packet being lost (lost). It represents the likelihood of a data packet not being successfully received or processed by the intended destination.

Fault types	**Mathematical Representation (t=time)**
Healthy Signal	Output = True value
Stuck Fault	Output = Constant
Offset/Bias Fault	Output = True Value + Offset
Gain Fault	Output = Gain * True Value
Noise Fault	Output = True Value + Noise
Saturation Fault	Output = Max or Min Value (depending on saturation)
Spike Fault	Output(t) = True Value + Spike(t)
Drift Fault	Output(t) = T * True Value

Table 2.1: Mathematical Representations of Fault Types

2.2.2 Single V/S Simultaneous Faults

Single sensor faults are instances when only one sensor malfunctions or fail, producing incorrect or false data. The operation of the entire system can be affected by a single defective sensor, which jeopardizes the accuracy of the data collected. Simultaneous faults, on the other hand, provide an even bigger problem since they include the simultaneous occurrence of two or more faults in various sensors. In our research work, only 2 faults are considered simultaneous faults. Different things can start these kinds of situations. Due to the difficulty in detecting and classifying the issue, the complexity of the problem considerably rises when two sensors fail at once. This can also put the system's general operation in danger by starting a chain reaction of faults and there is a high possibility that simultaneous faults in different sensors would occur [Biddle and Fallah, 2021]. In this study only two faults are considering figure 2.3a and 2.3b represent struck-at fault and noise



(a) Healthy Signal (b) Stuck-at Fault (c) Offset Fault

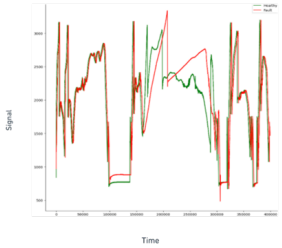
(d) Gain Fault (e) Noise Fault (f) Hard-over Fault

(g) Spike Fault (h) Drift Fault (i) Packet Loss Fault

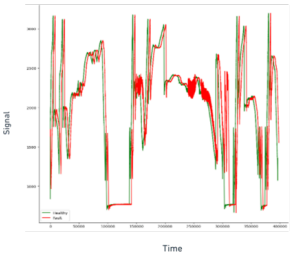
(j) Delay Fault

Figure 2.2: Different Kinds of Faults

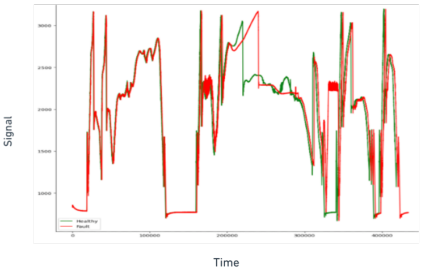
fault respectively. Simultaneous fault with a combination of struck-at and noise fault is shown in figure 2.3c. Where the green line below the signal indicates a healthy signal and red indicates a faulty signal.



(a) Single Fault: Gain Fault



(b) Single Fault: Noise Fault



(c) Simultaneous Faults: Gain and Noise

Figure 2.3: Single V/S Simultaneous Faults

2.2.3 Mitigation Techniques for Noise in Sensor Faults

The existence of undesired or arbitrary changes in data is referred to as noise in sensor faults, and it can have an impact on the precision and dependability of the data. Several factors, including electrical interference, the environment, faulty sensors, and distortions from signal processing, can cause noise [Grimaldi and Manto, 2010]. It is important to make noise in faults seriously and employ the right mitigation techniques.

Calibration And Maintenance Technique

Devices should be regularly calibrated and maintained to provide accurate measurements and minimize noise-induced inaccuracies [Coble et al., 2013]. However, this is more of a noise tolerance method than a noise removal method.

Signal Conditioning Technique

The impact of the noise can be reduced by using devices like amplifiers or signal conditioners, this will also improve the signal quality [Chung et al., 1990].

Filtering Techniques Technique

Using filtering techniques such as low-pass filters, to filter out high-frequency noise while keeping the intended signal [Li et al., 2014].

Sensor Redundancy Technique

Using redundant sensors and fusing their outputs will help to improve measurement accuracy and lessen the effect of noise on defect detection [Wang et al., 2014].

Advanced Signal Processing Technique

Advanced machine learning or other signal processing algorithms are used to remove the noise from the signal data. In order to maximize noise reduction, adapt to changing noise characteristics, and increase fault detection accuracy in the presence of noise, sophisticated signal processing techniques make use of the power of cutting-edge algorithms, machine learning, and data analysis.

2.3 Auto-Balancing Techniques for Class Imbalance

There are different auto-balancing techniques to handle class imbalance. when one or more classes have a disproportionately smaller number of instances than the dominant class or classes it is called class imbalance. When this occurs, the classifier can be biased toward the majority class, which would result in poor performance when trying to predict the minority class or classes [Bedi et al., 2021]. Furthermore, when the minority classes indicate significant or crucial patterns, occurrences, or faults in the data, this may be very troublesome. Below are a few auto-balancing techniques to come up with imbalanced data problems.

2.3.1 Random Under Sampling

In Random under sampling, until the required class balance is reached, randomly choose instances from the dominant class, and eliminate them from the dataset. This works by removing instances from the majority class till the class is effectively balanced [Majhi et al., 2019]. Since it removes the instances, the size of the data comes down and this technique is computationally efficient. Random under-sampling can be divided into 4 major steps.

Imbalanced Dataset

To begin, we choose a dataset that is imbalanced, having more instances of the majority class than the minority class. Because the model tends to be more biased in favour of the dominant class, this class imbalance may result in biased model performance.

Desired Class Balance

Before beginning the Random under-sampling process. Typically, this is determined by considering the intended trade-off between the two classes or using domain expertise.

Random Decision-Making

Up until the necessary class balance is achieved, the dominant class is under-sampled by randomly choosing instances. The cases chosen for elimination are chosen at random, preventing any bias or trend from being established.

Data Reduction

The amount of the dataset shrinks when instances from the majority class are eliminated. Because more data is for model training may increase computing efficiency.

Random Decision-Making

Up until the necessary class balance is achieved, the dominant class is under-sampled by randomly choosing instances. The cases chosen for elimination are chosen at random, preventing any bias or trend from being established.

2.3.2 Random Over Sampling

In contrast to under-sampling, over-sampling randomly duplicates minority class instances until classes are balanced [Yap et al., 2014]. Because of data duplication, there could be chances that model learns more about the minority classes and also loss of information is avoided. This technique would be more suitable for less training data. In some cases, duplication of instances can introduce noise and bias. On the other hand, it can also lead to overfitting.

2.3.3 SMOTE (Synthetic Minority Over-sampling Technique)

By interpolating between nearby instances in the feature space, SMOTE creates synthetic instances. Unlike other techniques, SMOTE increases the representation of the minority class while keeping the temporal relationships and patterns [Iwana and Uchida, 2021]. Hence it enhances the model's generalization and can capture the underlying distribution. But synthetic data cannot accurately match original time series patterns which might lead to performance issues or low performance.

2.3.4 ADASYN (Adaptive Synthetic Sampling)

It creates artificial instances for the minority class similar to SMOTE. It emphasizes difficult-to-learn examples by giving them more weight during the production of synthetic samples [Susan and Kumar, 2021]. Hence the performance of ADASYN could depend closely on the parameter selection. Incorrectly set weights may have an effect on the balance of the data.

2.4 Fault Detection and Classification Approach

Fault detection is the process of identifying fault signals. As discussed previously faults can be of different types and defining which kind of fault has occurred is known as fault classification. Fault detection and classification are performed based on the data collected as data will reflect this fault. To detect and classify the faults there is a need to closely monitor and understand data patterns. There exist different methods to understand the data and to deal with faults in intelligent ways. The choice of these methods depends on different factors like availability of historical data, desired level of robustness and characteristics of data.

2.4.1 Data-driven Method

Data-driven models create models that capture typical behaviour and trends using past sensor data [Chen et al., 2023]. These models may be based on statistical approaches, machine learning techniques, or data mining methods (such as regression, decision trees, or neural networks). Anomalies or deviations can be found by comparing real-time sensor data to the trained models, revealing probable flaws. Major note in the data-driven method analyses large data and also turns huge data advantageous [Ng and Winkler, 2014] [Yin et al., 2014]. Hence with huge data available data-driven method is more suitable for our fault detection and classification approach.

2.4.2 Signal-based Method

Analyzing the properties of the sensor signals itself is the main emphasis of signal-based models [28]. Techniques like signal processing, digital filtering, time-frequency analysis, or waveform analysis may be used in this. The frequency content, noise levels, signal structure, or time-domain characteristics of the sensor signals can be examined to look for anomalous patterns or anomalies that could point to a problem [Aziz et al., 2020]. Hence signal-based method would be more suitable when there are huge types of signals.

2.4.3 Model-based Method

This historical-based method analyses behaviour and connections between various system components by mathematical or system models [Gao et al., 2015]. Model-based methods involve process history-based methods, qualitative and quantitative mode-based [Venkatasubramanian et al., 2003]. The disadvantage of this model stays with complex systems, it is really hard to define model-based methods for complete systems like ASSs.

2.4.4 Hybrid Method

A combination of 2 or more above-mentioned methods is called a hybrid method. This method is more advantageous when there are weak or biased labels in data [Bergen et al., 2019]. This method needs information to merge between sensor data, historical knowledge, and system dynamics.

2.5 Machine Learning for SFDC

This section will explain different machine learning algorithms and their application for simultaneous fault detection and classification. The goal of this section is to find the appropriate algorithm for simultaneous fault detection and classification. Simultaneous faults are also called multilabel and multi-classification problems in machine learning terms. Also, data consists of deliberate noise which needs to be handled by the algorithm in a smart way.

Firstly, machine learning is classified into 3 major categories.

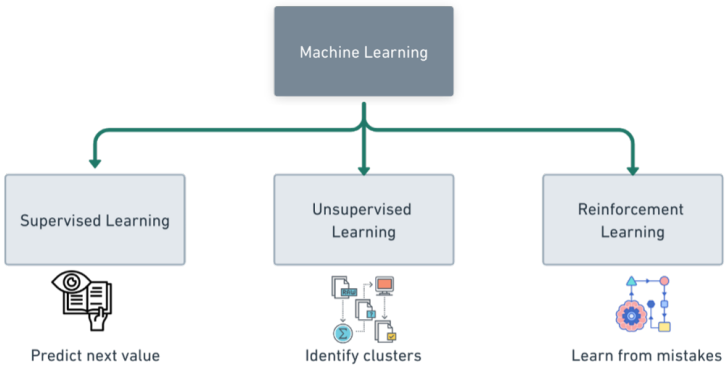


Figure 2.4: Machine Learning Classification

2.5.1 Supervised Learning

A machine learning technique called supervised learning uses labelled data to train a model while providing both the input features and the matching target values [Nasteski, 2017]. By learning the relationship between input features and the target values that correspond to them, supervised learning aims to provide models with the knowledge necessary to correctly forecast or categorize brand-new, unexplored data. When we have access to a labelled dataset and wish to train a model to generalize from the provided instances, supervised learning is appropriate. It is frequently employed for applications like classification and regression, where the target variable represents discrete groups or categories and a continuous value, respectively. Hence if the labelled data set is available it is always a good option to go with supervised learning. Figure 2.5 illustrates the steps in supervised learning.

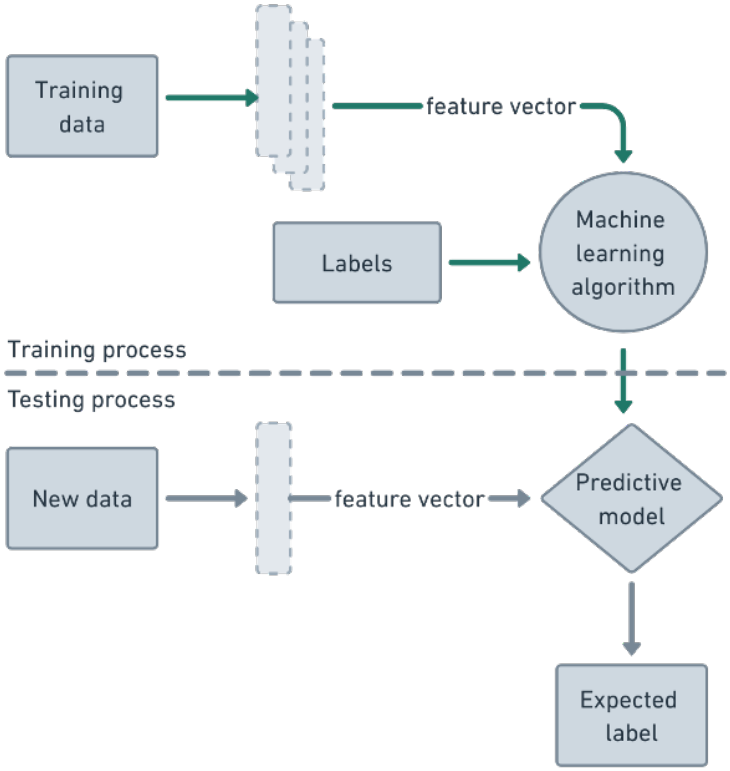


Figure 2.5: Supervised Learning Model

2.5.2 Unsupervised Learning

Unsupervised learning is technique where model is trained with unlabeled data and only input features are fed to the model to learn independently. It uses the learned features to predict new data [Mahesh, 2020]. Unsupervised learning techniques learn from patterns, structures, or relationships [Bergen et al., 2019]. Unsupervised learning will also learn the unseen patterns or clusters in the data. For tasks like clustering, where related data points are clustered together, and dimensionality reduction, where high-dimensional data is represented in a lower-dimensional space, unsupervised learning methods are frequently utilized [Hahne et al., 2008]. Figure 2.6 imitates the process of unsupervised learning.

2.5.3 Reinforcement Learning

Reinforcement learning is a machine learning technique teaches an agent to interact with its surroundings and take consecutive actions to maximize cumulative rewards [Mirchevska et al., 2018].

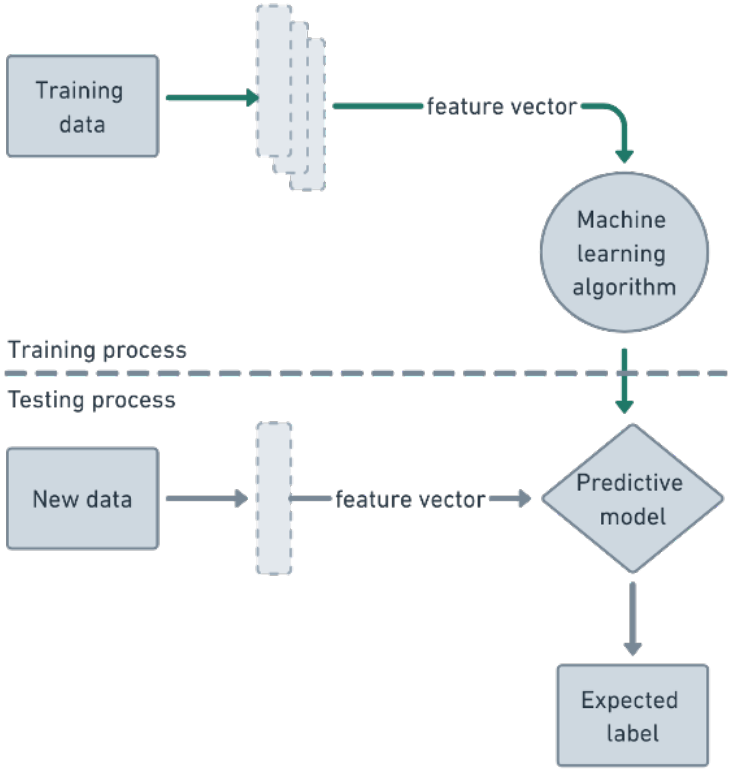


Figure 2.6: Unsupervised Learning Model

Reinforcement learning does not rely on labelled data like supervised does, instead, the agent learns about its surroundings by making mistakes, acting, and then getting feedback in the form of incentives or penalties. The reinforcement learning broad framework is shown in Figure 2.7.

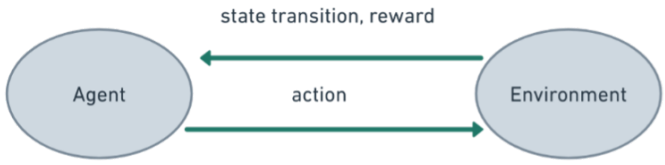


Figure 2.7: Reinforcement Learning Model

2.5.4 Machine Learning Techniques

2.5.4.1 Naïve Bayes

A probabilistic approach called Naïve Bayes assumes that characteristics are independent given the class label. A data instance's likelihood of belonging to each class is calculated, and the class

with the highest probability is chosen as the prediction. Naïve Bayes has good text classification [Kosmopoulos et al., 2008]and spam filtering performance, is computationally efficient, and needs minimal training data [Ting et al., 2011]. However, Naïve Bayes is a simple algorithm might be outperformed by other well-trained models [Ray, 2019].

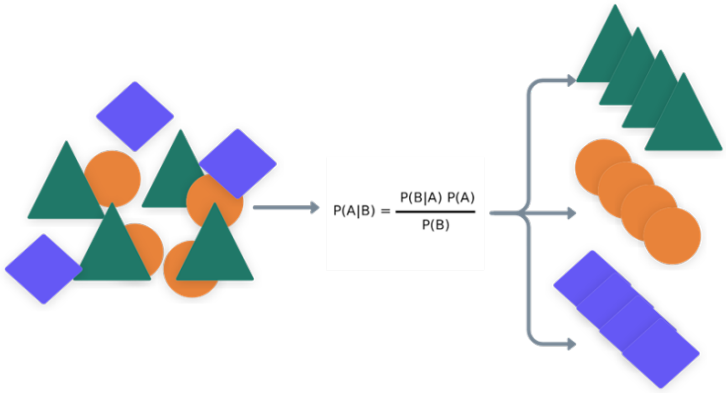


Figure 2.8: Naive Bayes as classifier

2.5.4.2 K-Nearest Neighbors (KNN)

A non-parametric method and lazy learner called KNN to categorize data based on how close one instance is to other classified examples. It chooses a class label from among its k closest neighbours in the feature space by a majority vote. KNN is a versatile tool that can be used for both classification and regression tasks. It is very simple to learn and performs well with non-linear boundaries. KNN as an algorithm will become computationally intensive with an increase in the training set size [Ray, 2019].

2.5.4.3 Linear Regression

Linear regression works on the linear relation between features and target values. Widely employed for regression tasks, linear regression is interpretable, computationally effective, and effective. It does, however, presuppose a linear connection that could miss non-linear patterns in the data [Ding et al., 2019]. Hence Linear regression might not be suitable for data which doesn't have linear relation.

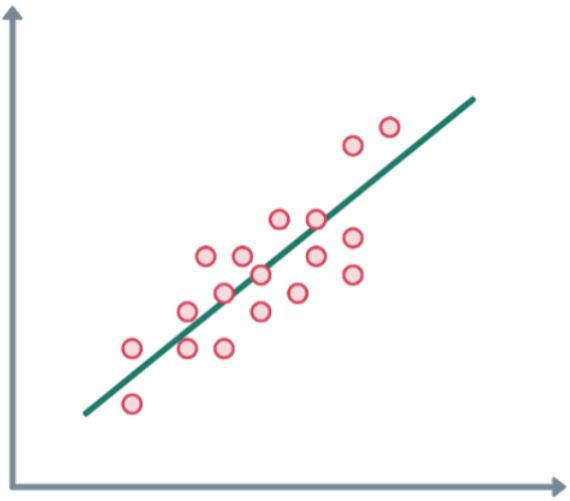


Figure 2.9: Linear Regression

2.5.4.4 Decision Tree

By recursively dividing the data based on feature values, decision trees produce a model that resembles a tree. Each leaf node corresponds to a class label or regression value, whereas each interior node represents a feature. Decision trees can capture non-linear connections, handle both numerical and categorical variables, and are simple to read. Decision trees have drawbacks, including the potential for instability, difficulty in controlling tree size, the potential for sampling error, and the tendency to provide locally optimum rather than globally optimal solutions [Ray, 2019]. Hence it might turn hard for decision trees to understand complex relationships.

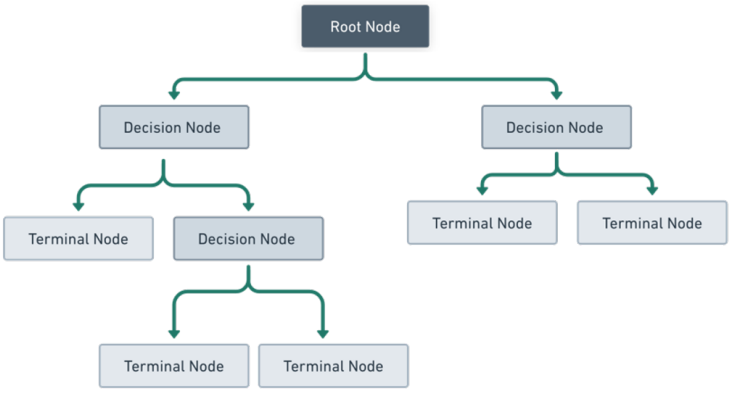


Figure 2.10: Decision Tree

2.5.4.5 Support Vector Machines (SVMs)

The effective SVM technique locates the best hyperplane to divide data points into distinct classes in a high-dimensional feature space. It allows for good generalization by maximizing the margin between the classes. By utilizing kernel functions to translate the data into higher dimensions, SVMs perform well with both linearly separable and non-linearly separable data [Ray, 2019]. However, if large data exist then the training time of SVM increases rapidly and finding a suitable kernel function will also be challenging.

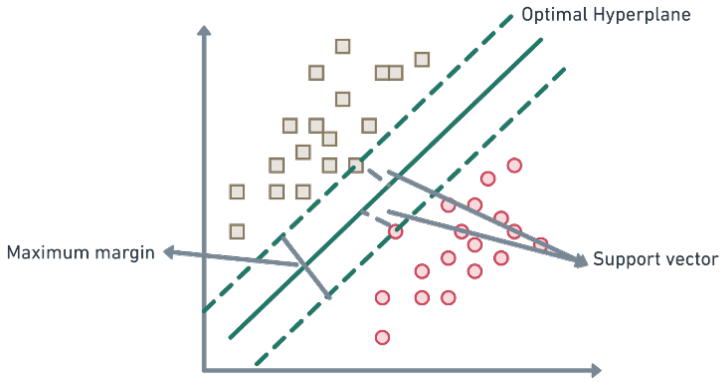


Figure 2.11: Support Vector Machine

2.5.4.6 Random Forest (RF)

A technique for ensemble learning called Random Forest mixes many decision trees to provide predictions. It is an efficient technique that may be utilized for both classification and regression problems [Izquierdo-Verdiguier and Zurita-Milla, 2020]. To create a final outcome, Random Forest aggregates the forecasts from all of the decision trees it has built. Random Forest is more suitable for classification tasks because it can deal with huge feature spaces and with high dimensional data efficiently [Zhang and Suganthan, 2014]. Random Forest is also capable of capturing features with nonlinear relationships with target values [68]. Random forest is more robust towards imbalanced data and noise-prone data [Khoshgoftaar et al., 2007].

The architecture of a Random Forest can be described as follows:

Tree Construction: Random Forest consists of a collection of decision trees, each constructed independently. The number of trees in the forest is denoted as N.

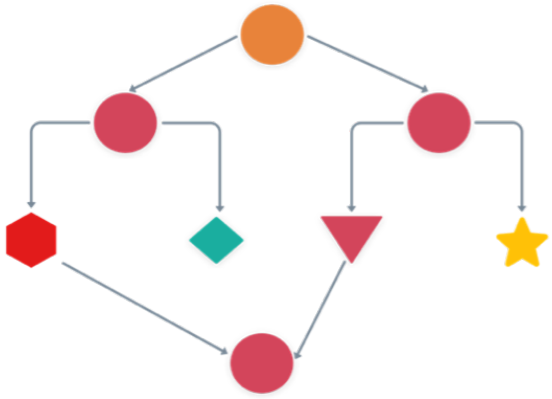


Figure 2.12: Random Forest

Random Feature Subset: At each node of a decision tree, a random subset of features is selected for determining the best split. This introduces randomness and helps prevent overfitting. The number of features considered at each node is denoted as F.

Bootstrap Aggregating (Bagging) : Random Forest employs a technique called bootstrap aggregating or bagging. It involves creating multiple bootstrap samples from the original training dataset. Each tree is trained on a different bootstrap sample, introducing diversity among the trees.

Decision Tree Training: Training each tree in the Random Forest involves recursively partitioning the data based on the selected features and their split points. This process continues until a predefined stopping criterion is reached, such as a maximum depth or minimum number of samples in a leaf node.

Prediction: To make predictions using the Random Forest, the outputs of all individual trees are combined. For classification tasks, the predicted class is determined by majority voting among the trees. For regression tasks, the predicted value is often the average of the individual tree predictions. In summary, the architecture of Random Forest can be represented in equation 2.1.

$$y = \text{Mode}(T_1(x), T_2(x), \ldots, T_N(x)) \tag{2.1}$$

Here, y represents the predicted output, Mode denotes the majority voting operation, and $T_i(x)$ represents the prediction of the i-th decision tree in the forest.

2.5.5 Deep Learning Techniques

2.5.5.1 Convolutional Neural Network (CNN)

CNN is known for its application in vision-related tasks [Qin et al., 2018] because CNN works mainly on the grid-based structure which makes it also work better for time-series data [Priyadarshini and Cotton, 2 CNN uses learning filter and convolutional layer to understand complex data patterns [Li et al., 2019]. They are well known for extracting features from the input data without need of feature extraction engineering [Shen et al., 2020]. Hence data with a lower grid structure or less spatial relationship might not be suitable for CNN. Due to its deeper structure, CNN needs huge training data for learning and will be computationally expensive.

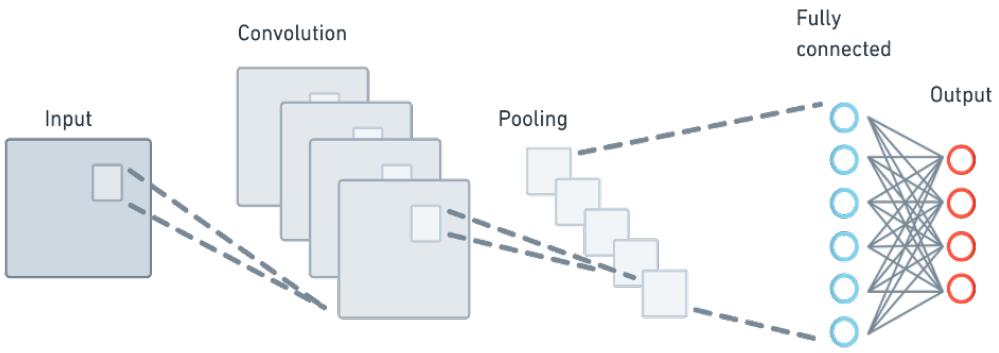


Figure 2.13: Convolutional Neural Network

2.5.5.2 Recurrent Neural Network (RNN)

For sequential data analysis tasks like language processing and time series prediction, deep learning models called RNNs are used [Shen et al., 2020]. Feedback links between them can detect context and temporal relationships [Cheng and Wang, 2007]. RNNs process data one element at a time and keep an internal state as a repository for previous knowledge. Language modelling, machine translation, speech recognition, and sentiment analysis are just a few of the activities that RNNs are utilized for [Adarsh et al., 2019]. They could experience the vanishing gradient issue, which interferes with their capacity to detect long-term relationships. The long-term dependency capture of the model is enhanced by advanced RNN variations like LSTM and GRU, which solve the vanishing gradient issue [Pavithra et al., 2019].

2.5.5.3 Gated Recurrent Unit (GRU)

The recurrent neural network (RNN) architecture known as GRU, or Gated Recurrent Unit, was created to recognize and represent sequential relationships in data [He et al., 2020]. It functions in a manner similar to other RNN versions but adds gating features that regulate the input flow inside the network. The vanishing gradient issue is solved by GRU by using gating techniques that enable the network to only preserve and update information that is relevant [Cho et al., 2014]. The model can more accurately represent long-term relationships, thanks to these gates, which control the gradient flow during training.

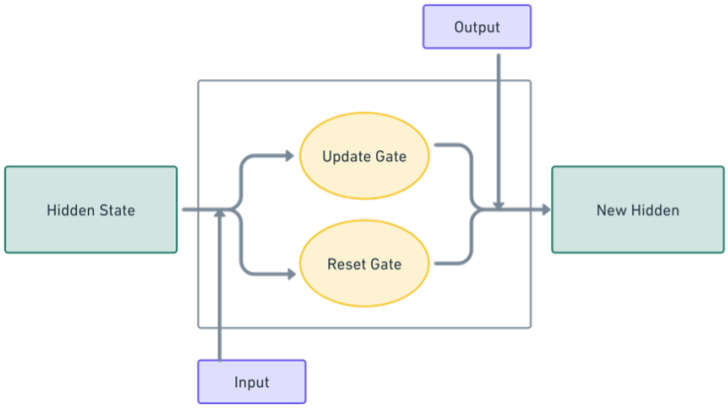


Figure 2.14: Gated Recurrent Unit

2.5.5.4 Long short-term memory (LSTM)

Long Short-Term Memory, or LSTM for short, is a kind of recurrent neural network (RNN) architecture made to tackle the vanishing gradient problem and identify long-term relationships in sequential data [Wang et al., 2014]. To control the flow of information throughout the network, it uses memory cells, input gates, forget gates, and output gates [Rahman et al., 2016]. A memory cell and gating mechanisms are used by LSTM to solve the vanishing gradient problem. While the gating mechanisms regulate the information flow, the memory cell enables the network to store and retain information across longer sequences [Chandar et al., 2019]. In comparison with GRU, GRU is more suitable with short datasets and if computational power not a concern LSTM outperforms GRU in a few cases [Yang et al., 2020].

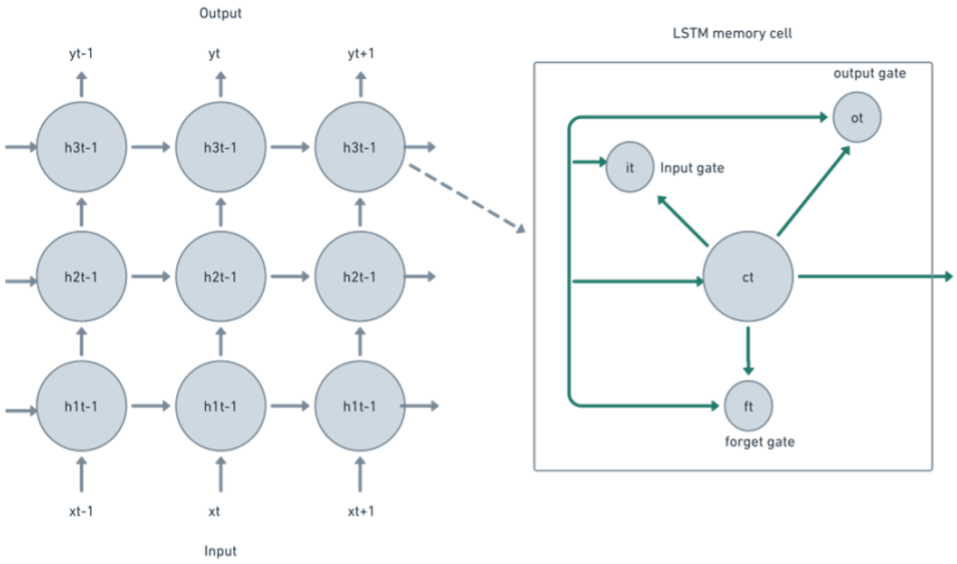


Figure 2.15: Long Short -Term Memory

LSTM architecture consists of 4 main components: memory cell, input gates, forget gate and output gates. Below is the explanation for the 4 main components with some other important components of LSTM.

Memory Cell (c_t): The memory cell represents the memory of the LSTM and is updated based on the input gate, forget gate, and candidate memory cell. In reference to Equation 2.2, the LSTM updates the memory cell state (c_t) by combining the previous memory cell state (C_{t-1}) with the forget gate (f_t) and the input gate (i_t) multiplied by the candidate memory cell state (g_t). This allows the LSTM to selectively forget or remember information from the past and incorporate new information based on the input and the gating mechanisms.

$$C_t = f_t \cdot C_{t-1} + i_t \cdot g_t \tag{2.2}$$

Forget Gate (f_t): The forget gate (f_t) determines how much of the previous memory cell (C_{t-1}) should be forgotten. It is computed by applying the sigmoid activation function (σ) to the weighted sum of the previous hidden state (h_{t-1}), current input (x_t), weight matrix (W_f), and bias vector (b_f).

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.3}$$

Input Gate (i_t): The input gate (i_t) controls how much of the candidate memory cell (g_t) should be

added to the updated memory cell (C_t). It is calculated by applying the sigmoid activation function (σ) to the weighted sum of the previous hidden state (h_{t-1}), current input (x_t), weight matrix (W_i), and bias vector (b_i).

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.4}$$

Candidate Memory Cell (g_t): The candidate memory cell (g_t) holds the information that could be added to the memory cell. It is computed by applying the hyperbolic tangent activation function (tanh) to the weighted sum of the previous hidden state (h_{t-1}), current input (x_t), weight matrix (W_g), and bias vector (b_g).

$$g_t = \tanh(W_g \cdot [h_{t-1}, x_t] + b_g) \tag{2.5}$$

Output Gate (o_t): The output gate (o_t) determines how much of the memory cell (C_t) should be exposed as the hidden state (h_t) at the current time step. It controls the flow of information from the memory cell to the output. The output gate is computed by applying the sigmoid activation function (σ) to the weighted sum of the previous hidden state (h_{t-1}), current input (x_t), weight matrix (W_o), and bias vector (b_o).

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.6}$$

2.5.5.5 Denoising Autoencoder (DAE)

A form of artificial neural network called a denoising autoencoder (DAE) is made to filter out noise from data [Lu et al., 2013]. They are a subset of autoencoders, which are unsupervised learning models that attempt to learn effective representations of the input data by encoding and decoding it. Another benefit of denoising autoencoders is that they may develop reliable representations of the input data. They are versatile for a variety of denoising activities across several domains since they can manage different levels and types of noise [Vincent et al., 2008].

The architecture of the Denoising Autoencoder can be summarized as follows:

Input Layer: The input layer receives the noisy data as input.

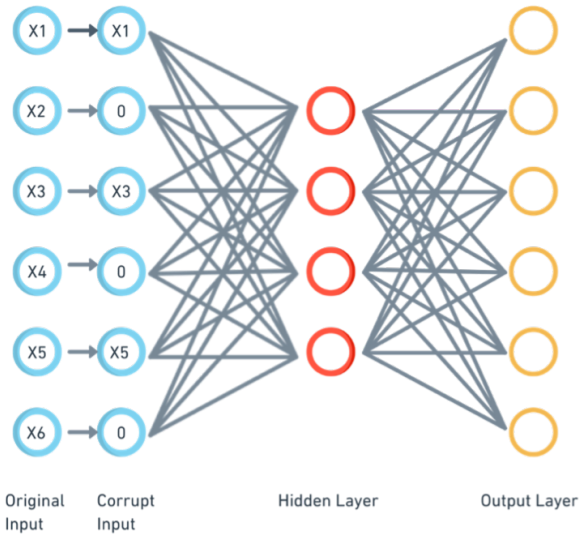


Figure 2.16: Denoising Autoencoder

Encoder Layer: Each encoder layer applies a linear transformation followed by a non-linear activation function. Let's denote the input to the encoder layer as x, and the output as h. The encoder layer can be represented mathematically as equation 2.7

$$h = f(Wx + b) \tag{2.7}$$

Encoding Layer: The encoding layer represents the compressed and denoised representation of the input data. It can be mathematically represented as in equation 2.8

$$h_{\text{enc}} = f_{\text{enc}}(W_{\text{enc}}h + b_{\text{enc}}) \tag{2.8}$$

where h represents the output of the last encoder layer, W_{enc} is the weight matrix, b_{enc} is the bias vector, and f_{enc} is the activation function.

Decoder Layer: Each decoder layer applies a linear transformation followed by a non-linear activation function to reconstruct the clean data. Let's denote the input to the decoder layer as h_{dec}, and the output as $\hat{x}$. Equation 2.9 represents the mathematical representation of the decoder layer.

$$hatx = f_{\text{dec}}(W_{\text{dec}}h_{\text{dec}} + b_{\text{dec}}) \tag{2.9}$$

where W_{dec} represents the weight matrix, b_{dec} is the bias vector, and f_{dec} is the activation function.

Output Layer: The output layer produces the denoised output, closely resembling the original clean input data.

Algorithm	Applications	Advantage	Disadvantage
NB	Text classification and sentimental analysis [Abbas et al., 2019]	Less computational time for training and improves classification performance by removing irrelevant features [Han et al., 2012]	Works on independent assumption hence would be hard to classify complex relationships [Dhande and Patnaik, 2014]
KNN	Image recognition [Hazra et al., 2017]	Ability to deal with noisy data [Jadhav and Channe, 2016]	Runs slowly because it is a supervised lazy learner [Jadhav and Channe, 2016]
LR	Regression analysis	Easy to implement and understand, highly interpretable	Does not capture complex relationships as it assumes a linear relationship
SVM	Image Classification	Works well on both linear and nonlinear data	Computationally expensive, proper tuning is required as it consists of huge parameters
CNN	Image classification and object detection	Learns hierarchical features	Requires huge training data
GRU	Speech recognition [Shewalkar et al., 2019]	Captures long-term dependencies well	May not capture complex relationship effectively as LSTM
LSTM	Feature extraction [Wang et al., 2020] and Sentiment analysis	Captured long-term and complex relationships, well suited for sequential data	Requires more training time
RF	Classification and regression [Wang et al., 2020]	Robust to noisy and imbalanced data	Complex implementation due to different tree combinations

DAE	Image denoising	Learns robust representations by removing noise	Computationally expensive and large amount of training data is required

Table 2.2: Overview of Machine Learning Algorithm

2.6 Summary

In the software development process, model-based development is extremely important, especially when defining and validating software requirements. The SIL, MIL, PIL, HIL, and VIL testing stages of development are all represented by the V-cycle model. In order to guarantee the quality and dependability of the program, it is imperative to comprehend these phases. Faults can be classified into different types such as stuck faults, offset/bias faults, drift faults, and more. Recognizing these fault types is vital for effectively detecting and addressing them. Single faults occur when only one sensor malfunctions, whereas simultaneous faults involve the occurrence of two faults in different sensors simultaneously. Simultaneous faults present a greater challenge due to their complexity. Therefore, accurate detection and classification methods must be employed to address such scenarios effectively. The presence of noise in sensor readings can significantly impact the accuracy and reliability of fault detection. Techniques such as calibration, signal conditioning, filtering, and sensor redundancy can help mitigate the impact of noise on fault detection. Another critical issue is how to address the data's class imbalance. To balance the data and enhance the effectiveness of fault detection algorithms, techniques like random under sampling, random over sampling, SMOTE, and ADASYN can be used. When it comes to fault detection and classification, various approaches can be used. Data-driven methods, such as statistical analysis, machine learning, and data mining, create models based on past sensor data to identify anomalies. Signal-based methods focus on analyzing the properties of sensor signals themselves. Model-based methods leverage mathematical or system models to analyze system behavior and component relationships. Hybrid methods combine multiple approaches to achieve more accurate fault detection. Machine learning algorithms, including supervised, unsupervised, and Reinforcement learning, can be utilized for fault detection and classification. The choice of algorithm depends on the problem nature, avail-

able data, and desired robustness. However, it is essential to address deliberate noise in the data effectively, which can pose a challenge in fault detection. In conclusion, understanding the phases of model-based development, recognizing different fault types, applying techniques to handle noise and class imbalance, and employing appropriate fault detection and classification approaches are all critical components for designing effective fault detection and classification model.

3 Methodology

This chapter explains about flowchart of the proposed Deep learning model in detail.

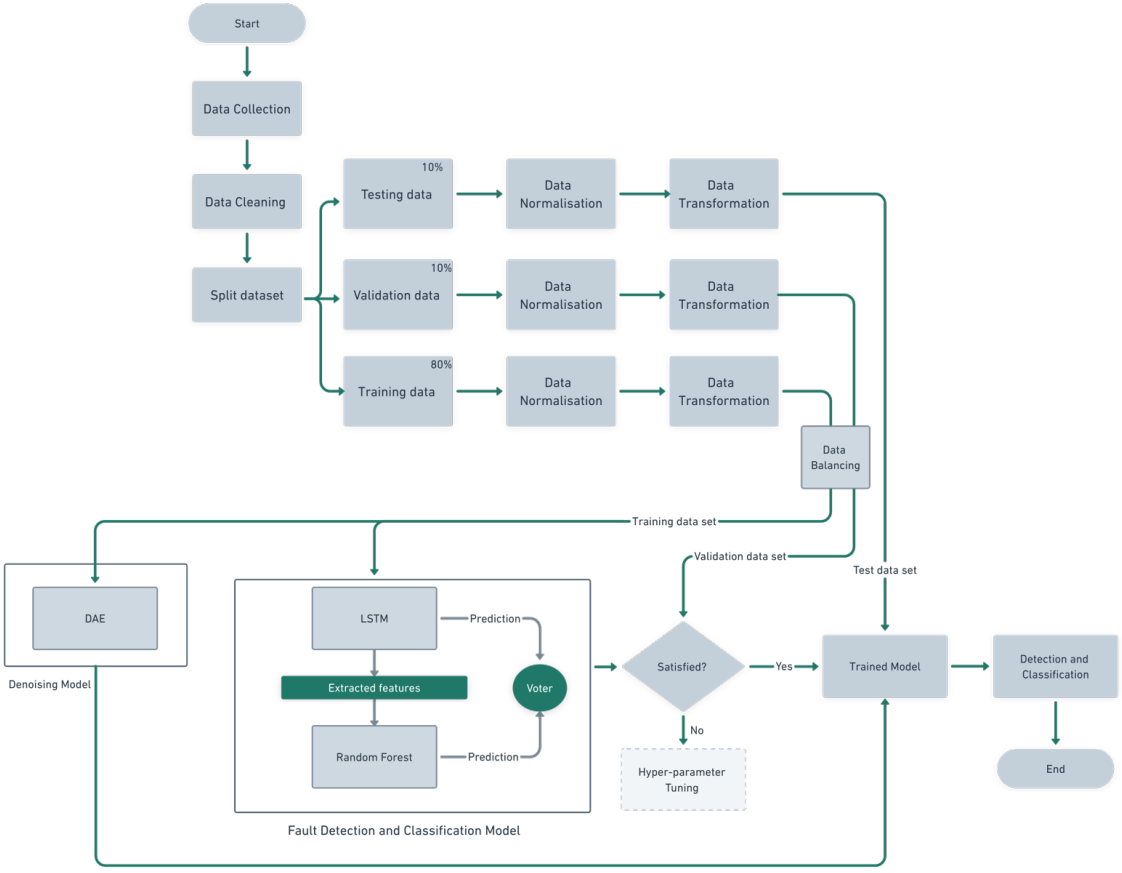


Figure 3.1: Fault Detection and Classification Model Approach

Any deep learning model is performed based on the data it is trained with hence very initial and important step is data collection. It would be hard for the model to understand from the raw data or model might not be trained properly with raw data hence some preprocessing steps are applied before feeding data into the model. Once preprocessed data is fed into the model and trained with it later comes the testing and validation using testing and validation datasets respectively. We will be walking through different phases as shown in figure 3.1 in detail.

3.1 Data Collection

Two types of data called healthy and faulty are collected from HIL systems. Healthy data can be collected directly from the system but faulty data needs fault injection into HIL. The fault is injected into HIL by using the fault injection framework Faulty data has different combinations of data as simultaneous faults. Data generated by HIL is stored in a computer machine with different file formats. Data collection in HIL has a few important components which are listed below.

3.1.1 HIL System Confirguration

A detailed grasp of the system setup is the first step in the data collection procedure for fault injection in HIL systems. A real-time simulator, control panels, actuators, and sensors are just a few of the physical parts that make up the HIL system. These elements cooperate with the software elements, such as the simulation models and the HIL software framework, to mimic the behaviour of the real-world system.

3.1.2 Fault Injection Framework

The configuration and management of fault situations rely heavily on the Graphical User Interface (GUI) employed in HIL systems for fault injection. We can configure fault parameters, injection timing, and fault types using the GUI's user-friendly interface. It provides simple features and choices for defining fault magnitudes, durations, onset timings, and propagation patterns. We can also quickly construct and simulate fault situations using the fault injection GUI, allowing to research the impact of various faults on the behavior and performance of the system [Abboush et al., 2022a].

3.1.3 Fault Injection Parameter

Accurate fault parameter definition is crucial while carrying out fault injection-based research. Fault types, such as sensor faults, communication faults, actuator faults, or system-level faults, are defined using the fault injection GUI. Even size of fault, how long they last, and when they first appear in the system is controlled using GUI. We may explore the system's reaction to various fault

scenarios by adjusting these parameters and simulating real-world fault occurrences. This way simultaneous and single faults are collected by indicating the time when the fault should occur.

3.1.4 Real-Time Data Collection

Real-time data collecting techniques are used to document the system's behaviour during fault injection studies. While actuators react to control inputs, sensors detect important system characteristics. The gathered information offers perceptions of the functionality of the system and enables the investigation and diagnosis of faults. The sample rates, data-collecting strategies, and synchronization mechanisms are carefully chosen and put into practice to guarantee reliable data collection. This makes it possible for us to get reliable data that precisely depicts the system's behaviour during fault injection.

3.1.5 Data Logging and Storage

For future examination, the data gathered during fault injection trials must be appropriately saved and maintained. The massive amounts of data collected are managed via logging procedures and storage systems. The gathered information may be kept in databases, file formats, or specialized data storage systems created for dependable and effective data management. Before storing the data, pretreatment or filtering procedures may also be used to improve its quality and remove any noise or artefacts that may have been added during the data-gathering process.

3.2 Data Preprocessing

Data preprocessing, which uses a variety of approaches to convert raw data into a clear, well-organized, and intelligible format appropriate for future analysis, is a basic stage in data analysis and machine learning. Before starting with preprocessing, we wanted to check data set for PCA.

3.2.1 Principal Component Analysis (PCA)

Architecture may use PCA to assess and minimize the dimensionality of architectural features and elements, providing a streamlined representation of complex designs while preserving the main

variations and patterns in the data [Gupta et al., 2021]. In an analysis of the dataset, PCA was applied to investigate whether dimensionality reduction could help improve the classification performance. The goal was to identify a smaller set of principal components that could capture a significant portion of the variance in the data, ideally around 95% or higher. This reduced feature set would then be used to train the classification model, potentially leading to better performance and computational efficiency.

However, upon examining the results of the PCA, it was found that the first six principal components only accounted for approximately 93.86% of the total variance in the data. This falls short of our desired threshold of 95% explained variance. As a result, we have decided not to use the PCA-derived components for classification. This decision is based on the concern that the reduced feature set might not adequately represent the original data and could lead to a loss in classification accuracy. Hence it was decided to explore alternative feature engineering techniques or other methods to improve the classification performance while maintaining the integrity of the original data. Figure and Table will help to understand how much of the total variance is present in the data as a principal component.

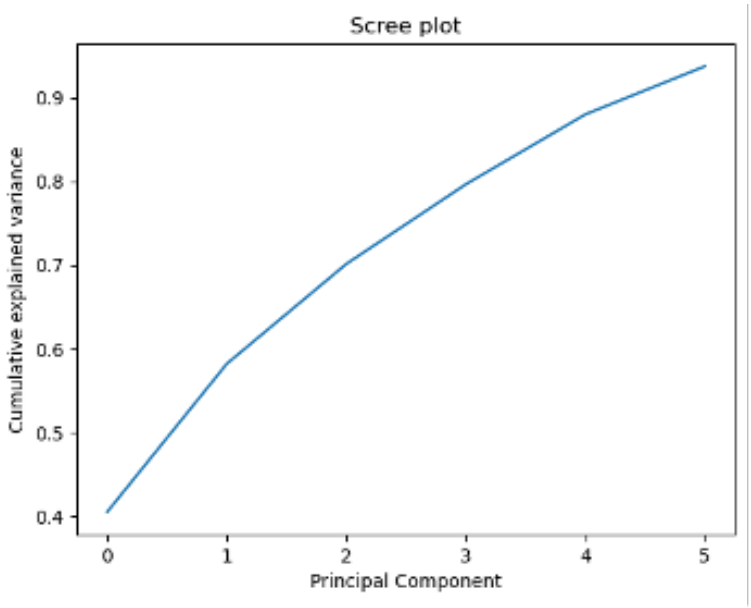


Figure 3.2: Contribution of Principal Components In PCA

Principal Component	**% of variance**
Principal Component 01	40.5%
Principal Component 02	17.7%
Principal Component 03	11.9%
Principal Component 04	9.5%
Principal Component 05	8.4%
Principal Component 06	5.8%
Total	**93.8%**

Table 3.1: Principal Component Inference

3.2.2 Split Dataset and Data Normalization

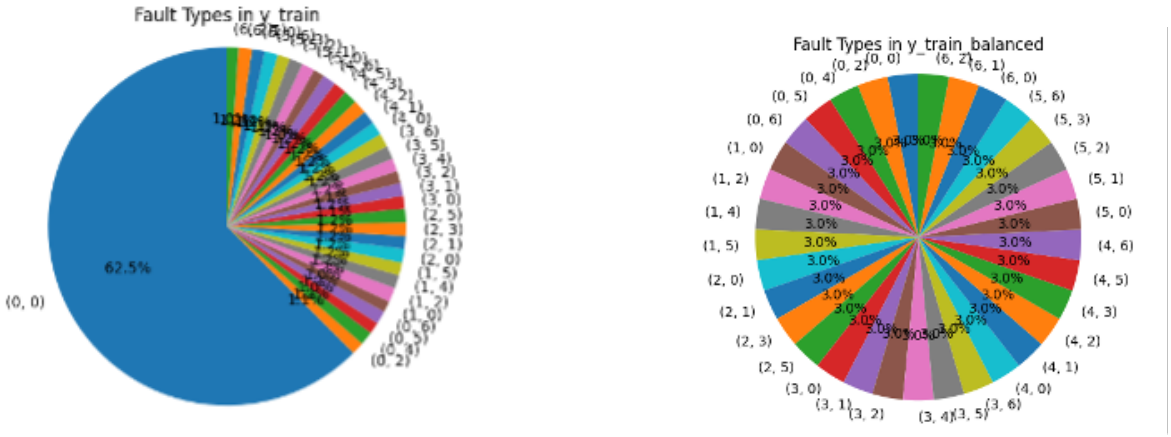
The dataset is split into features and target variables and further splits the data into training, testing and validation sets. Training, testing, and validation sets are divided in the ratio of 80:10:10 respectively. Later data is normalized using standard scalar.

3.2.3 Data Transformation

Since we will be dealing with single and simultaneous faults with the same model, we will be transforming data using a power label set. The power label set assigns each combination of faults with a unique index value using a dictionary.

3.2.4 Data Balancing

Using the Random under sampler data balancing of the majority class is performed until the number of samples in each class is the same. The input features are then reshaped to be compatible. Figure 3.3 represents different classes of data before and after balancing. Where (0,0) represents healthy data and other combinations are faulty types.



(a) Data before balancing

(b) Data after balancing

Figure 3.3: Representation of Data Before and After Auto-balancing

3.3 Data Denoising Model

Balanced data is fed into DAE which is used to denoise the data. Denoising autoencoder is trained against different noise levels of 1,4,8,10,15,20 and 30%. Training DAE against different levels of noise will enable it to be robust again noise.

3.4 Simultaneous Fault Detection and Classification Model

After data balancing, the output of auto-balanced data is fed into the LSTM+RF model for feature extraction and classification. The validation data set is used for hyperparameter tuning, using validation data set for hyperparameter tuning will enable the model to understand unseen data. The grid search method is used for hyperparameter tuning where different possible combinations of hyperparameters are cross-validated and the best combination is suggested. A testing data set is used to evaluate the trained model performance.

Relevant ML Algorithm for SFDC

In context of detection and classification of simultaneous and single faults with properties of noisy and imbalanced data, different models are used to address specific challenges. Using several tech-

niques and different models will leverage strength of the models. After auto-balancing the imbalanced data using random undersampling, data is added with different levels of Gaussian noise intentionally. DAE is used to incorporate noise removal. By intentionally adding noise to the data artificial patterns are added. DAE is trained with noisy data but attempts to reconstruct noise-free or clean data, hence it learns to remove the noise in the process. Architectural factors of DAE like encoder-decoder structure, reconstruction objective and nonlinear transformation make it learn compressed representations that capture essential features, discard noise, and reconstruct clean data accurately. The denoising step will increase the robustness and accuracy of the feature extraction process. LSTM are well suited for sequential data like time series data as it captures temporal dependencies and understands complex relationship of data. By utilizing LSTM for feature extraction one can capture meaningful representation of data which is essential for classification. LSTMs can efficiently describe and categorize complicated fault patterns because of their capacity to handle sequential data. One may make use of LSTM networks' capacity to gather temporal information and use the learnt representations to guide their judgments by using LSTM networks for classification. Though GRU is an improvised version of RNN like LSTM, LSTM overperforms GRU in capturing complex relationship [Yang, 2016]. Hence LSTM can be used for both feature extraction and classification task. An assembly-based approach called Random Forest is known for its robustness over noise and overfitting. Not only that, but its tolerance also to imbalanced data, capacity to capture complicated patterns and feature significance evaluation make it an accurate model for classification. In a recent paper [Wu et al., 2019] author has used DAE to remove noise in faulty data. Authors in [Oh et al., 2021] has proposed LSTM for fault detection and classification. But RF is also used in works like [Patel and Giri, 2016] for fault classification. In the current thesis work, RF is used to classify the faults based on the features extracted by LSTM. In the selection of the best approach for SFDC between LSTM and RF for classification, the research [Theissler, 2017] [Xiao et al., 2018] in deep learning was encountered, which proposed the use of a voter for a parallel model that will combine the predictions of classifiers based on the majority vote. In this way, the model gets the benefits of both algorithms. The general structure of the DAE+LSTM+RF is shown in Figure 3.4.

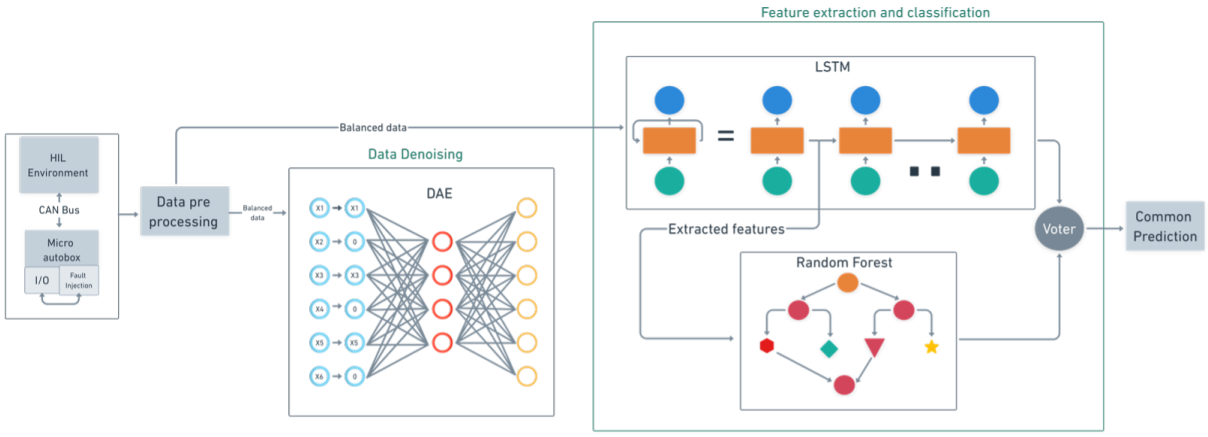


Figure 3.4: Fault Proposed Architecture of Model

3.5 Summary

In the process of building a deep learning model for fault detection and classification in HIL systems, data collection plays a crucial role. Both healthy and faulty data are collected, with healthy data obtained directly from the system and faulty data injected into the HIL system using a fault injection GUI. Real-time data collection techniques are employed to monitor the system's behaviour during fault injection studies, providing insights into its functionality and facilitating fault investigation. The collected data is appropriately logged and stored for future analysis, undergoing preprocessing steps to enhance its quality. Data preprocessing involves transforming the raw data into a suitable format for analysis and machine learning. Principal Component Analysis (PCA) was initially explored for dimensionality reduction, but alternative methods were considered due to the PCA results falling short of the desired explained variance threshold. The preprocessed data is then split into training, testing, and validation sets. The training data is used to train a denoising autoencoder (DAE) model to remove noise from the data, while the balanced data is fed into an LSTM+RF model for feature extraction and classification. Hyperparameter tuning is performed using the validation set, and the trained model's performance is evaluated using the testing set. This comprehensive approach allows for the detection and classification of faults in HIL systems using deep learning techniques.

4 Implementation

This chapter describes the implementation of the proposed solution of SFDC. To illustrate implementation first we will be explaining about gasoline engine case study.

4.1 Case Study: Gasoline Engine

Cars in today's era are powered by gasoline engines, which are a crucial part of contemporary automotive systems. The vehicle is propelled and given the required power to function as a result of the conversion of the chemical energy in gasoline into mechanical energy. Developing cutting-edge automotive technology, improving engine performance, and assuring efficient and dependable operation all depend on an understanding of the complex workings of a gasoline engine.

4.1.1 Automotive Simulation Models

In the pursuit of innovation and continuous improvement, automotive manufacturers and researchers rely on simulation models to gain insights into the behavior of complex systems. Automotive Simulation Models (ASM) is a complete toolkit offered by dSPACE [aut, 2019], a top supplier of automotive simulation solutions. This suite compresses a wide range of simulation models that were created especially for automotive applications, such as simulations of gasoline engines, vehicles, and electric components [aut, 2019]. By utilizing these simulation models, engineers and researchers may make precise and realistic assessments of automotive systems, facilitating the creation of cutting-edge solutions and the improvement of vehicle performance. The ASM suite's gasoline engine model, ECU model, CAN bus simulation, and sensor and actuator models offer a powerful and realistic simulation environment for advancing automotive technologies [aut, 2019]. Model modifications are carried out in Simulink level by dSPACE software called control desk. Once code is automatically generated by Simulink dSPACE SCALEXIO can be used for code deployment [aut, 2019].

Gasoline Engine Model

This model, which was created exclusively for automotive purposes, offers a thorough simulation of a 6-cylinder turbocharged gasoline engine. It offers both direct injection and manifold options, as well as manual and automatic gearboxes. The detailed simulation of engine behaviour, including fuel system dynamics, air path characteristics, coolant system performance, exhaust system effects, and piston engine operation, is made possible by the ASM gasoline engine model. The ASM gasoline engine model is a key element for in-depth analyses and simulations of gasoline engines due to its adaptable features and dependable performance.

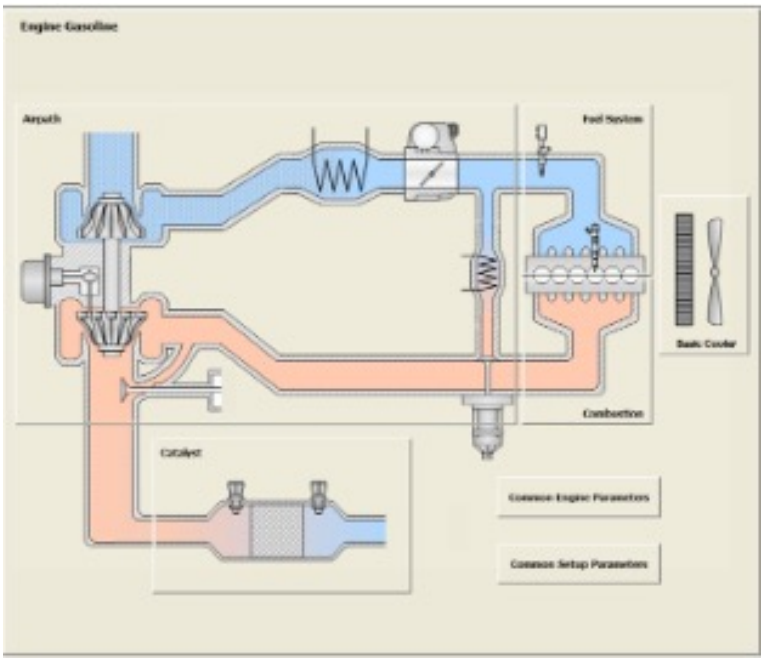


Figure 4.1: Simulation of dSPACE gasoline engine
[Abboush et al., 2022b]

Fuel System The fuel system simply represents a common-rail system of ASM. It guarantees that gasoline reaches the engine cylinders at the correct flow and pressure. The fault system consists of below components:

- High-pressure pump - A model of the fuel pump used to deliver fuel under high pressure.
- Fuel metering - Models the exact management of the gasoline supply to the engine cylinders via fuel metering.

- Pressure control valve - Fuel pressure in the common rail system is controlled by a pressure control valve.
- Fuel tank - The reservoir used to store gasoline.
- Injector - Recreates the fuel injectors that fill the cylinders with fuel.

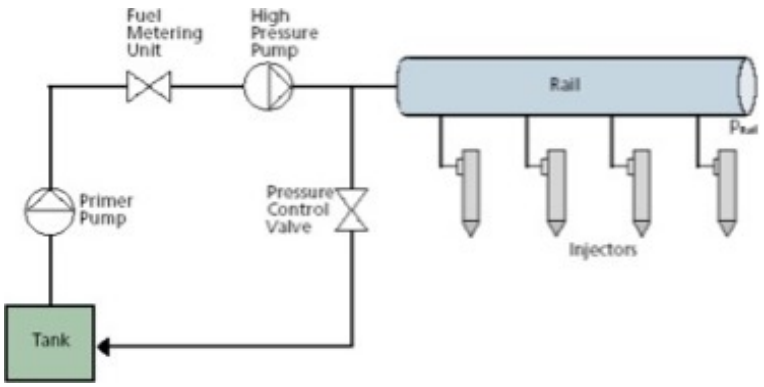


Figure 4.2: Common-rail Fuel System

Air Path Components including an intake manifold, intercooler, turbocharger, exhaust manifold, and air compressor, an exhaust gas recirculation (EGR) system are included in the air path simulation. It mimics the impacts of EGR and turbocharging on engine performance while modelling air flow and calculating air pressure.

- Compressor - Simulates the compression of entering air.
- Intercooler - Before pressurized air enters the engine; the intercooler simulates cooling it.
- Exhaust gas recirculation (EGR) - Simulates the recirculation of a portion of exhaust gases to reduce emissions.
- Turbocharger – Represents the device that compresses the incoming air for improved engine performance.
- Intake and exhaust manifold - Models the paths through which air enters and exhaust gases exit the engine.

Coolant System The blocks in the cooling system mimic how engine coolant behaves. It features a thermostat to control the cooling water temperature and estimates water temperature using variables like mean and friction torque.

- Cooling system blocks - Models the movement and control of the temperature of the engine coolant using cooling system blocks.
- Water temperature calculation - Based on friction torque and mean values water temperature values are simulated.
- Thermostat - The component in charge of controlling coolant temperature is represented by a thermostat.

Exhaust System The flow of exhaust gases and their interaction with the engine are represented by the exhaust system model. It offers a condensed illustration of how the exhaust system behaves.

Drivetrain The engine's torque is sent to the wheels through the drivetrain model. It consists of parts like the clutch, gearbox, and differential. The drivetrain model integrates inputs including environmental signals, engine torque, and wheel speed in addition to considering manual and automated transmission modes.

Vehicle Dynamics The vehicle dynamics model represents the load torque on the vehicle, taking into account factors like friction, slope, load, and external driving resistances like aerodynamics. To calculate the speed of the wheel, it takes into account inputs including the load on the vehicle, environmental signals, and torque from the drivetrain model.

Environment Includes a variety of elements linked to road conditions and outside factors that affect the vehicle. This model can be used to control driver signals, road signals, external resistances and driving manoeuvres. This model can be used to create real-time driving scenarios in simulation models and this will enable to evaluation of performance, response and efficiency with respect to different scenarios.

4.1.2 ECU

The sophisticated Electronic Control Unit (ECU) model in the ASM suite closely replicates the features of actual ECUs used in automotive systems. The ECU model is in charge of producing precise actuator control signals based on data from numerous sensors. In addition to other pertinent control parameters, it regulates important parts of engine function like fuel metering, common rail pressure management, fuel injection volume and timing, injection angles, and other crucial components of engine operation. dSPACE MicroAutoBox II is used as ECU in this work which is connected through CAN Bus to dSPACE SCALEXIO. In order to enable thorough simulations and assessments of concurrent fault detection and classification model, the ASM MicroAutoBox II model smoothly interfaces with the gasoline engine model.

4.1.3 Controller Area Network (CAN) Bus Simulation

A specific simulation component for the Controller Area Network (CAN) bus is offered by dSPACE's ASM suite to recreate the intricate communication network found within a car. The behavior of the actual CAN bus, which is widely utilized for communication between ECUs and different automotive components, is faithfully simulated by the CAN bus simulation. CAN bus is used to communicate sensor signal from MicroAutoBox II to dSPCASE SCALEXIO.

4.1.4 Sensor and Actuator Models

To enable realistic modelling of these components in automotive systems, dSPACE's ASM suite provides a wide range of sensor and actuator models. The behavior of different sensors, including as crank angle sensors, battery voltage sensors, accelerator pedal position sensors, and more, is simulated by these models. Models for actuators including fuel injectors, pressure control valves, throttle valves, and EGR valves are also included in the suite. Table 4.1 and 4.2

Sensor Name	**UOM**	**Information**
Coolant Temperature	[°C]	Evaluates the engine coolant's temperature. It aids in keeping track of the engine's operating temperature.

Oxygen Sensor	[V]	Determines the voltage coming from the exhaust system's oxygen sensor. It is employed to check whether the air-fuel ratio is within the desired range.
Mass Air Flow	[g/s]	Determines the air entering the engine's mass flow rate. It aids in calculating the quantity of air necessary for a good fuel combination.
Engine Oil Pressure	[kPa]	Determines the engine's oil pressure. It guards against engine damage by keeping an eye on the lubrication system's performance.
Engine Speed	[rpm]	Specifies the engine's rpms, or revolutions per minute. It offers details on the engine's rotational speed.
Fuel Pressure	[kPa]	Determines the fuel system's fuel pressure. It makes sure the engine gets the right amount of gasoline.
Crank Angle	[aTDC]	Timing for ignition and fuel injection are both determined by it.
Intake Manifold Pressure	[kPa]	Determines the engine's intake manifold's internal pressure. It provides information about the air pressure entering the combustion chamber.
Knock Sensor	[V]	Determines the knock sensor's voltage. Engine knock or explosion, which may signify irregular combustion, are detected.
Battery Voltage	[V]	Determines the battery voltage of the car. It displays the electrical system's voltage level.
Throttle Position	[%]	Shows the throttle valve's position, which displays how wide the throttle is opened. It discloses the amount of engine power the driver needs.

Post-throttle Position	[%]	The position of the post-throttle valve, which represents the amount of throttle opening following the primary throttle valve, is indicated.
Engine Outlet Temperature	[°C]	Determines the engine coolant's output temperature. It offers details on the effectiveness of the engine's cooling system.
Mean Effective Engine Torque	[Nm]	Determines the engine's average torque throughout the course of a whole combustion cycle.
Engine Speed	[rpm]	Specifies the engine's rpms, or revolutions per minute. It offers details on the engine's rotational speed.
Intake Manifold Pressure	[Pa]	Measures the pressure inside the intake manifold of the engine. It indicates the pressure of air entering the combustion chamber.
Rail Pressure	[bar]	Determines the fuel pressure in the fuel rail. It keeps track of gasoline rail pressure.
Vehicle Speed	[km/h]	Calculates the vehicle's speed in kilometers per hour.

Table 4.1: Sensor Signals

Actuator Control Name	**UOM**	**Description**
Fuel Injector Pulse Width	ms	Regulates the duration of fuel injection pulses
Throttle Position	%	Measures the opening angle of the throttle valve
Variable Valve Timing (VVT) Control	degrees	Controls the timing of the intake/exhaust valves in the engine
Exhaust Gas Recirculation (EGR) Control	%	Manages the recirculation of exhaust gas for emission control

Turbocharger Boost Control	bar	Regulates the pressure of the turbocharger's boost
Idle Air Control	%	Controls the amount of air supplied during idle conditions
Electronic Stability Control (ESC)	0, 1	Manages the braking and traction of the vehicle for stability
Transmission Shift Control	gear	Controls the shifting of gears in an automatic transmission
Active Suspension Control	mm	Controls the height or stiffness of the vehicle's suspension
Electric Power Steering Control	degrees	Manages the level of assistance provided in the power steering

Table 4.2: Actuator Signals

4.1.5 Fault Injection Framework

Fault injection is modelled in Matlab Simulink. Signals are modified or manipulated using Simulink and then the model is built in the configuration desk. Variable description file generated by configuration desk and fed into dSPACE control desk. Using the control desk user can observe the variation or changes. Figure 4.3 from [Abboush et al., 2022a] illustrates the fault injection framework.

4.1.6 Matlab Simulink

Creating a Simulink model containing sensor models that replicate actual sensor behaviour is the first step in the process. For real-time simulation, the model is linked to the dSPACE HIL simulator using the Real-Time Interface (RTI) CAN multi-message block set [dsp,]. The general dialogue box of the RTI CAN multi-message block set in Simulink is used. Specific parameters representing sensor readings are specified inside the Simulink model to inject single or simultaneous sensor errors. Depending on the type of sensor, these metrics may include pressure, location, temperature, or velocity. To mimic actual single sensor faults, fault characteristics like bias, offset, drift, or noise

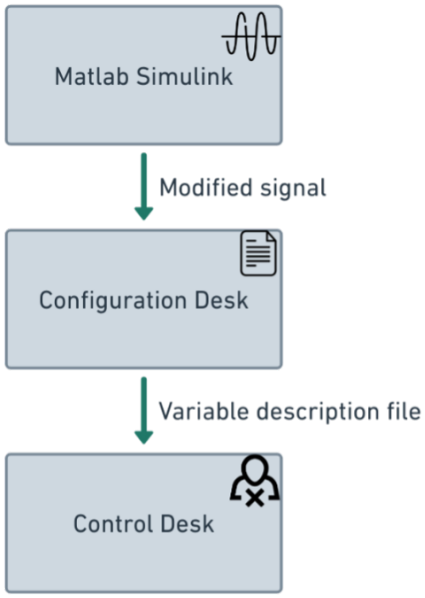


Figure 4.3: Fault Injection Framework

are given and to mimic simultaneous faults different combinations of faults like offset and drift with respect to different sensors. Real-time simulation is aided by the RTI CAN multi-message block set, which allows the fault injection framework to inject defined sensor failures into the appropriate parameters. dSPACE HIL simulator and Simulink model communication allow for synchronized simulation execution. Biased readings, erroneous measurements, irregular behaviour, or outright sensor failures are the symptoms of injection sensor faults. In real time, the system's reaction to these faults may be seen and examined. Fault injection settings may be configured, and system behaviour can be visualized thanks to the interaction between the configuration desk and the control desk.

4.1.7 Configuration Desk

The creation of a variable description file is a crucial step in the setting of the dSPACE ASM simulation environment and it is created by software called configuration desk. The communication and data interchange between various parts of the simulation setup is greatly facilitated by this file. After loading the project clicking on the 'start build' button shown in Figure 4.5 will create a variable description file successfully. You can sometimes notice an error message on the screen if

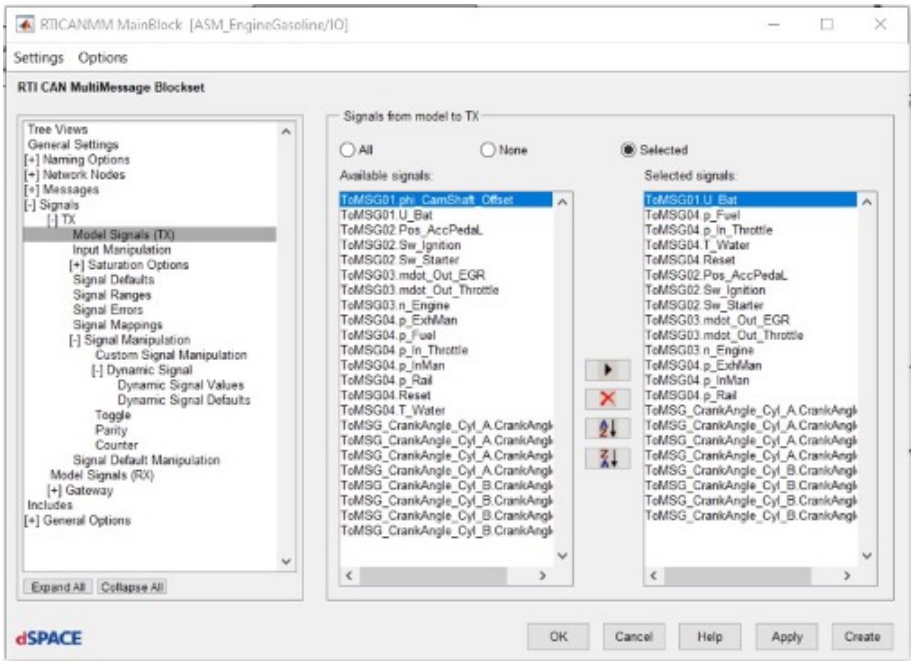


Figure 4.4: General Dialogue Box of RTI CAN Multi Message Block Set

the file is not created successfully.

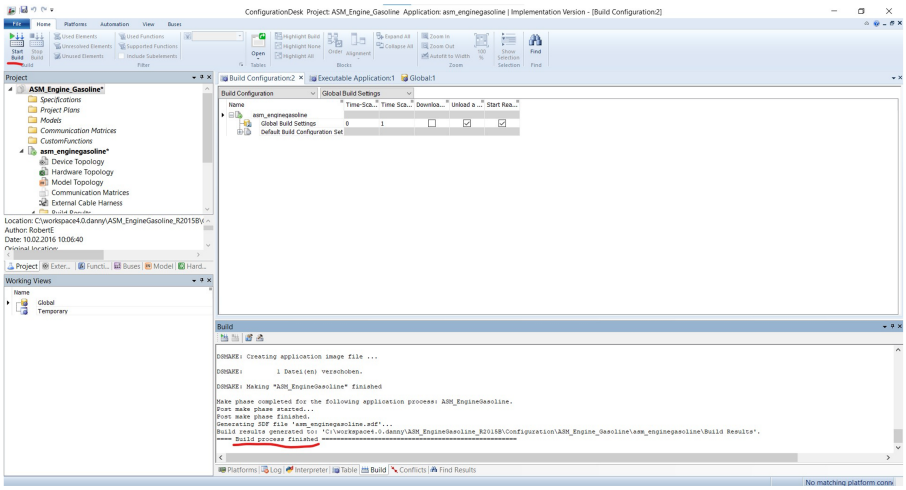


Figure 4.5: Graphical Layer of Engine Model

4.1.8 Control Desk

A crucial part of the dSPACE HIL (Hardware-in-the-Loop) system, the control desk offers a user-friendly interface for managing the simulation environment. Engineers and testers may interact

with simulation models and carry out a variety of tasks while testing and validating systems using the control desk. The control desk is illustrated in Figure 4.6 with fault injection GUI. The control desk starts flash's the variable description file as shown in Figure 4.7 which is sent by configuration desk hardware. Once the variable description file is flashed on the hardware you can manage CAN bus messages as shown in Figure 4.8. Now select the intended message which was prepared during model preparation to inject the fault and select 'Generate TX Layout'. This will take you to the next screen, which is the bus navigator array, you can observe this in Figure 4.9Signal manipulation is carried out by selecting a dropdown list where the dropdown list is displayed according to the Matlab Simulink instructions.

Figure 4.6: Control Desk Dashboard

The model root of the ECU variable description file has 2 major blocks. There is a fault injection block for each signal and different fault types under each signal. Simultaneous faults are selected by selection signal and faults respectively. During the runtime of the software tool, constant variables are controlled and Fault injection GUI is directly linked with the variables. Drag and drop is used to create the link between these variables.

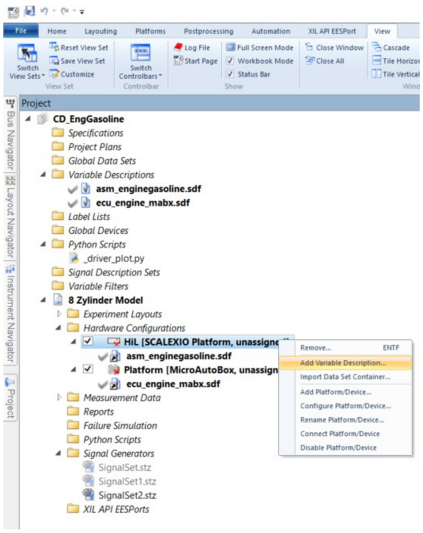


Figure 4.7: Hardware Configuration in Control Desk

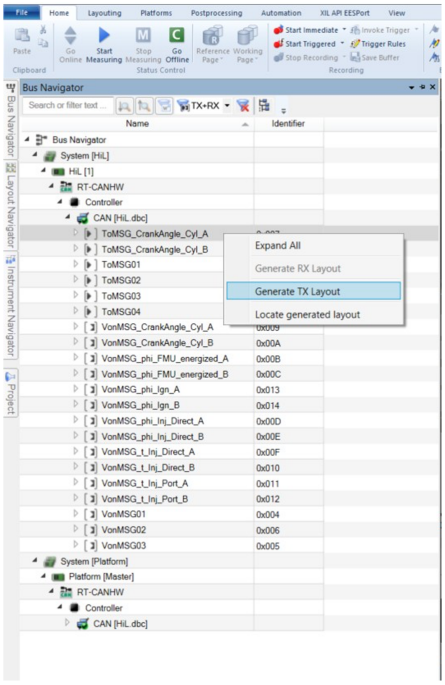


Figure 4.8: Bus Navigator Control Bar

4.2 HIL Data Collection

The primary goal of HIL data collection is to record numerous system-level variables that accurately reflect the performance and behaviour of the simulated system. In order to collect data

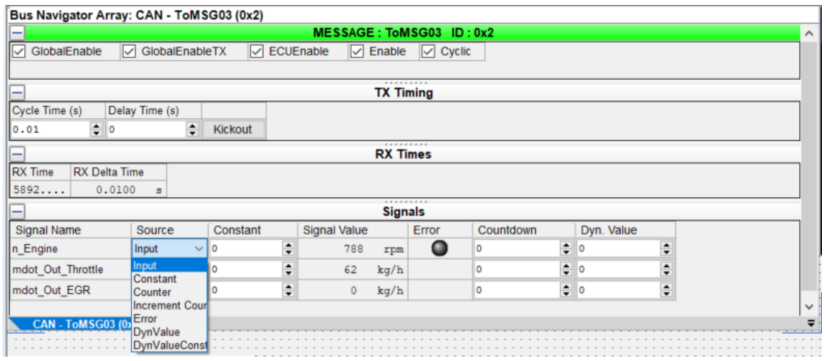


Figure 4.9: Bus Navigator Interface

related to different kinds of faults, different kinds of faults are injected into sensors.

4.2.1 Driving Scenario Control

Driving scenarios are treated as input for the system under test. Different kinds of faults are injected into one driving scenario. This includes specifying acceleration, braking, steering, and other control inputs that simulate real-world driving conditions.

4.2.2 Recording control

The control desk has buttons called “start immediately” and “stop recording” buttons. Once “start immediately” is clicked the selected signal in the measurement configuration control bar start recording and to stop the recording “stop recording” button is clicked.

4.2.3 File conversion and data clearance

Files are initially stored in IDV format which is only control desk readable. To make it convenient to see data outside the control lab there exists an option to convert files into. xlxs or .csv within the control desk. But for conversion file needs to be opened in the control desk measurement data pool and then the file is converted into the desired file format. Downloaded files might have trash data in terms of machine learning where it might have initial data details like location of variable description file and start and end time of recording. It is the very initial step of moving towards machine learning implementation by observing and cleaning this data. Each faulty .csv file will

have healthy and simultaneous or single faults in it. However, if the file concerns simultaneous fault, then the file will have 3 different patterns in it i.e., healthy, fault type 1 and fault type 2.

4.3 Simultaneous Fault Detection and Classification Model

After collecting data in the form of different .csv files each file mirrors single and simultaneous faults. Further steps were implemented using the programming language Python with the high-level deep learning framework PyTorch. Figure 4.10 shows the different steps involved in SFDC model development. The process of the SFDC model is divided into different steps: Fault dictionary, building data, data preparation, data balancing, model training and model result.

4.3.1 Data Dictionary

The data dictionary is a process which is used to label the data. During the data dictionary process, each file is defined with the start time of the fault, end time and combination of the fault. If a fault occurs between 170-350 seconds, data is labelled with fault combination only for this time period and the rest will be labelled as healthy data. The data dictionary is defined for each .csv data file based on fault visualization. Fault dictionaries or labels for fault types are defined in Table **??**. You can also view the example file which has healthy + simultaneous faults (Gain+Noise) labelled in Figure 4.11.

Fault type/healthy	**Dictionary**
Healthy (no fault)	0
Gain	1
Noise	2
Stuck-at-fault	3
Drift	4
Delay	5
Packet loss	6

Table 4.3: Data Dictionary

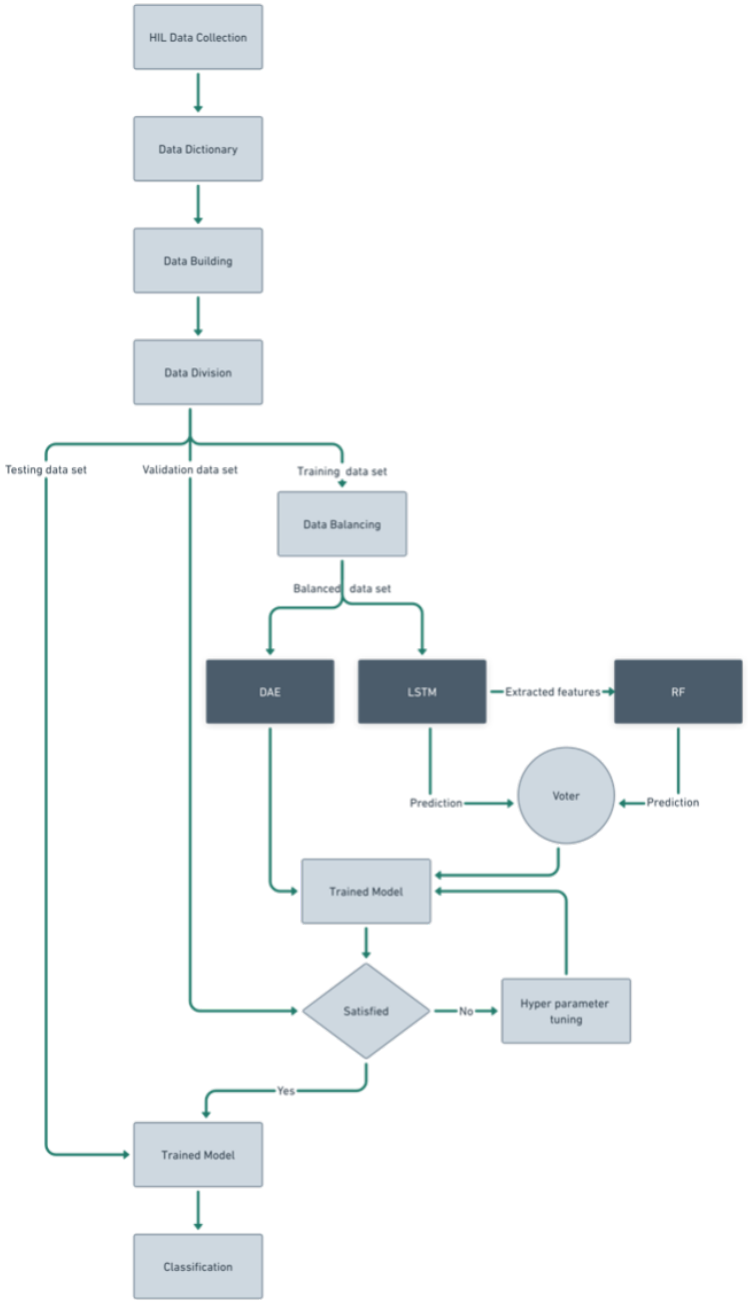


Figure 4.10: Flow Chart of SFDC Model

4.3.2 Data Building

Data building is the process of making data deep learning ready. In this section, the powerset labelling technique is applied and data are combined for further data division. Since data consist of both single and simultaneous faults using the same model for classification would be challenging

	Time	Pos_Throttle[%]	T_Out_Engine[	Trq_meanEff_Engine[	n_Engine[rpm]	p_InMan[Pa]	p_Rail[bar]	v_Vehicle[km|h]	Type 1	Type 2
305	0,3034218	100	712,93127	5,093801943	861,828482	99808,645	119,319668	0	0	0
306	0,3044216	100	712,963334	5,095204266	861,289608	99808,7407	119,327604	0	0	0
307	0,3054213	100	712,995305	5,095940556	860,686349	99808,8449	119,335224	0	0	0
308	0,3064224	100	713,027171	5,096289473	860,013082	99808,9602	119,342502	0	0	0
309	0,3074213	100	713,058921	5,115712948	859,512687	99809,089	119,363294	0	0	0
310	0,3084203	100	713,09054	5,135346361	859,195104	99809,2328	119,383799	0	0	0
311	0,3094192	100	713,12208	5,154179363	859,052021	99809,3495	119,403946	0	0	0
312	0,3104196	100	713,153575	5,172176588	859,072694	99809,4286	119,423832	0	1	2
313	0,3114184	100	713,185056	5,189353523	859,243321	99809,4696	119,44355	0	1	2
314	0,3124194	100	713,216551	5,205398011	859,546587	99809,4741	119,463184	0	1	2
315	0,3134216	100	713,248089	5,220157472	859,961339	99809,4449	119,48281	0	1	2
316	0,3144216	100	713,279692	5,233473326	860,462444	99809,3857	119,502493	0	1	2
317	0,3154213	100	713,311378	5,245181787	861,020685	99809,3009	119,522286	0	1	2
318	0,3164224	100	713,343163	5,255138428	861,603124	99809,1957	119,542228	0	1	2
319	0,3174212	100	713,375056	5,263228871	862,173713	99809,0763	119,562338	0	1	2
320	0,3184199	100	713,407058	5,269372146	862,693993	99808,9499	119,582621	0	1	2
321	0,3194188	100	713,439166	5,273536236	863,124149	99808,824	119,60306	0	1	2
322	0,3204196	100	713,471368	5,275752679	863,424383	99808,7071	119,623617	0	1	2
323	0,3214191	100	713,503647	5,276113929	863,556291	99808,6078	119,644234	0	1	2
324	0,322419	100	713,535976	5,274806895	863,484921	99808,5348	119,664833	0	1	2
325	0,3234218	100	713,568324	5,272096076	863,18055	99808,4966	119,685317	0	1	2
326	0,3244213	100	713,60065	5,275492428	862,755326	99808,5012	119,70557	0	1	2
327	0,3254214	100	713,632912	5,284532576	862,315849	99808,5552	119,725499	0	1	2

Figure 4.11: Flow Labeled Simultaneous Fault File

hence dictionary is defined which takes the numeric label pairs from the labelled data as keys and assigns them incrementing integer values. This new dictionary essentially provides a unique numeric label for each possible fault type pair.

4.3.3 Data Preparation

In data preparation, data is initially split into features and target variables. The next step was to split the data into 3 sets, 80% of training data, 10% for testing and 10% for validation. Once data is divided all 3 sets are normalized. Since the power set labeling technique is used fault combinations are replaced by pre-defined integer values.

4.3.4 Data Balancing

Data generated is imbalanced data where the ratio of one combination of data is different from the other and each csv file will have a ratio of healthy to faulty data. Using Random sampling technique majority class undergoes auto-balancing till the majority class is equal to the minority class. In our data majority class was healthy data.

4.3.5 Data Denoising

To make the model robust to the noise, the gaussian noise of different levels is added to the data. DAE is used to denoising the added noise. Since the model is only denoising and not doing any other complex task, hyperparameters are defined without tuning for training.

Input: Dataframe df, Healthy dataset H, faulty dataset F

Output: x_train, x_val, x_test, y_train, y_val, y_test

Function *PREPAREDATA(df, H, F)*

```
X = df.drop(['type1', 'type2']);
Y = df[['type1', 'type2']];
x_train, x_temp, y_train, y_temp = train_test_split(X, Y, test_size=0.2, satisfy=y);
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size=0.5);
foreach x_i in {x_train, x_val, x_test} do
    Normalize(x_i);
end
y_train, y_val, y_test = CombineAndEncodeFaultTypes(y_train, y_val, y_test);
```

end

Algorithm 1: Data Preparation Algorithm

4.3.6 Model Training and Tuning

The relevant machine learning algorithm selected in section 3.4is implemented. DAE is fed with balanced data where the model will denoise the data independently and the same balanced data is fed into LSTM to extract the features. After feature extraction, LSTM is used for classification and RF is in parallel used for classification from the features that is been fed from the LSTM. The voter is used to combine the predictions from LSTM and RF. Data from the DAE is not fed to LSTM only to maintain the originality of the data and make sure the model learns well from the clean data but not from the duplicate denoised data.

DAE

The denoising autoencoder is trained with a balanced data set after auto-balancing is performed. The model training process is depicted step-by-step in the flowchart shown in Figure 4.12. It starts with training data generation, and the model training loop begins, iterating over a total of 5 epochs. Within each epoch, the training loop involves several steps.

Input: $x_{\text{train}}, x_{\text{test}}, y_{\text{train}}, y_{\text{test}}, N_{\text{levels}} = \{0.01, 0.04, 0.08, 0.15, 0.2, 0.3\}$

Output: Trained models and combined prediction

Function *DENOISE_DATA_with_DAE(x_{train}, N_{levels})*

  foreach $j \in N_{levels}$ **do**

    Create a noisy version of the training data by adding noise N_j to x_{train}, obtaining

    $x_{\text{noisy}} = x_{\text{train}} + N_j$

    Train the DAE to reconstruct x_{train} from x_{noisy}, minimizing the reconstruction error

    $||x_{\text{train}} - DAE(x_{\text{noisy}})||_2^2$

  end

end

Function *FEATURE_EXTRACTION_and_CLASSIFICATION_with_LSTM(LSTM*)*

  foreach *LSTM in LSTM** **do**

    Define feature function $f : X \to R^d$ as the activation of the last hidden layer of the LSTM

    Extract features $F = f(x_{\text{train}})$

    Initialize classification

  end

end

Function *CLASSIFICATION_with_RF(T)*

  Initialize RF classifier with the number of estimators T

  Train the RF model on the reduced features F and labels y_{train}

end

Function *VOTER*

  Make predictions with LSTM: y_{LSTM}

  Extract reduced feature set from the test data: $F_{\text{test\_reduced}}$

  Make predictions with RF: y_{RF}

  Combine predictions of both models using a voting approach: $\hat{y}_{\text{train}} = \text{Vote}(y_{\text{LSTM}}, y_{\text{RF}})$

end

Algorithm 2: Model Training Algorithm

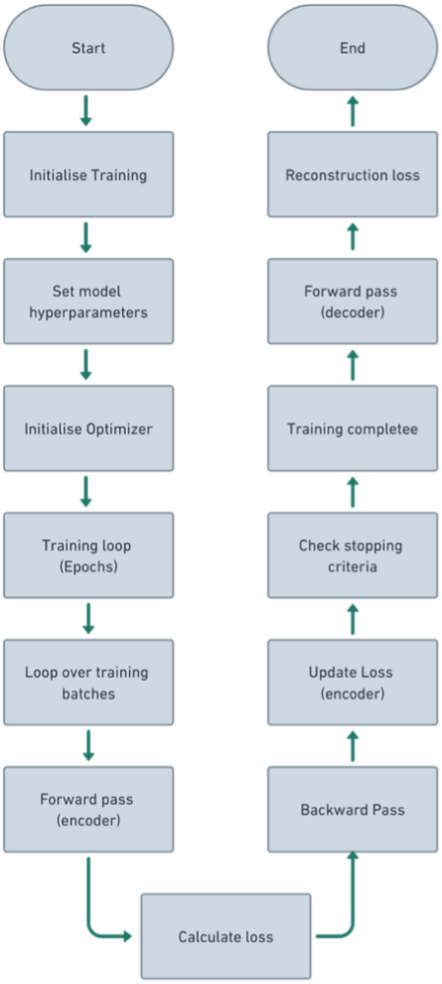


Figure 4.12: DAE Training Flowchart

The model hyper-parameters are initialized as shown in Table 4.4, and the optimizer Adam is used. For each batch, a forward pass is performed, where the input data is fed into the model, and the predicted outputs are obtained. The loss between the predicted outputs and the target outputs is calculated. Then, a backward pass is performed to compute the gradients of the model's parameters with respect to the loss. These gradients are used to update the model parameters through the optimizer. During the training loop, the loss is updated as a running total, allowing for monitoring the progress of the training. The loop continues until a stopping criterion, which is reaching 5 epochs. Figure 4.13 shows examples of noised data and denoised data after model training.

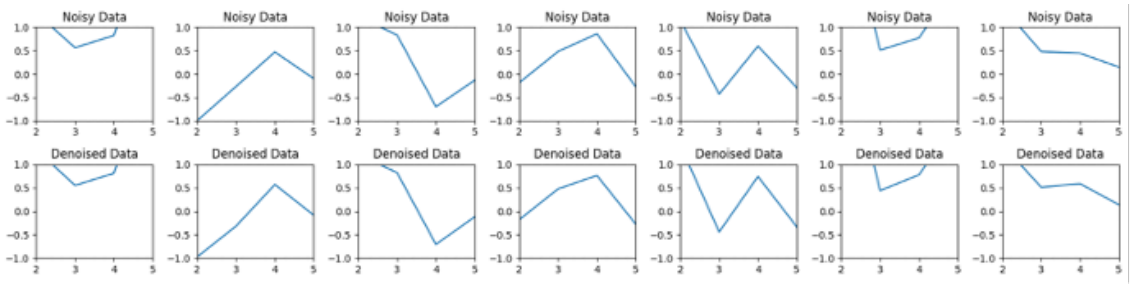


Figure 4.13: Example Noisy and Denoised Data

Hyper-parameter	**Value**
Encoder Layer	2
Decoder Layer	2
Activation Function	Leaky ReLU
Batch Size	512
Epochs	5
Optimizer	adam
Learning Rate	0.001

Table 4.4: Hyper-Parameters Used for DAE

LSTM

Once the LSTM model is trained from the training data set. Validation data is used to check with good hyperparameters of the model. For hyperparameter tuning method called grid search is used where it exhaustively searches through a specified grid of hyperparameter values to find the optimal combination that maximizes the performance of a machine learning model. Hyperparameter tuning is performed on the LSTM model. Different possible combinations of hyperparameter tuning are iterated to get the best suitable hyperparameters. The general hyperparameters used for the model are represented in Table 4.5. And we will be going through different model optimizations performed by grid search for the final best model.

Hyper-parameter	Value
Activation Function	tanh
Optimizer	adam
Hidden Layer Size	512, 128
LSTM Layer	1, 2
Epochs	5
Batch Size	1024
Learning Rate	0.001, 0.01

Table 4.5: Hyper-Parameters Used for LSTM

Hyperparameter test cases

Using grid search different combinations of hyperparameters are performed. Grid search works by establishing a grid of hyperparameter values to be searched and then methodically analysing the model's performance for each combination of hyperparameters. It iterates over all potential parameter combinations, allowing to compare and pick the set of hyperparameters that produce the best results [99]. Each combination goes through 5 epochs and best model is selected based on the performance comparison between them. Each of every test case has a unique combination. 3 hyperparameters i.e., Hidden layer size, LSTM layer and learning rate are varied between different combinations. Table 4.6 gives you clear overview of different test cases and performance.

The best hyperparameters are shown in table 4.7 and it is with a hidden layer size of 512, a learning rate of 0.001 and a model consisting of 2 LSTM layers. It has the highest accuracy of 0.66 and loss of 0.92 for the validation data set.

Test No.	01	02	03	04	05	06	07	08
Hidden layer size	512	512	512	512	128	128	128	128
LSTM layers	1	1	2	2	1	1	2	2
Learning rate	0.001	0.01	0.001	0.001	0.001	0.01	0.001	0.01
Training accuracy	0.50	0.55	0.65	0.64	0.46	0.51	0.54	0.60
Training loss	1.44	1.27	0.95	0.93	1.62	1.38	1.30	1.07
Validation accuracy	0.51	0.54	0.66	0.65	0.47	0.52	0.55	0.62
Validation loss	1.41	1.26	0.92	0.89	1.59	1.36	1.27	1.02
Test accuracy	0.78	0.80	0.85	0.84	0.76	0.77	0.80	0.82
Test loss	0.69	0.64	0.48	0.47	0.81	0.75	0.65	0.52

Table 4.6: Hyperparameter Tuning

Hyper-parameter	**Value**
Activation Function	tanh
Optimizer	adam
Hidden Layer Size	512
LSTM Layer	2
Epochs	5
Batch Size	1024
Learning Rate	0.001

Table 4.7: Optimized hyper-parameters for LSTM

Figure 4.14 will display the best hyperparameter training through 5 epochs. Training accuracy, validation accuracy, training loss and validation loss are shown.

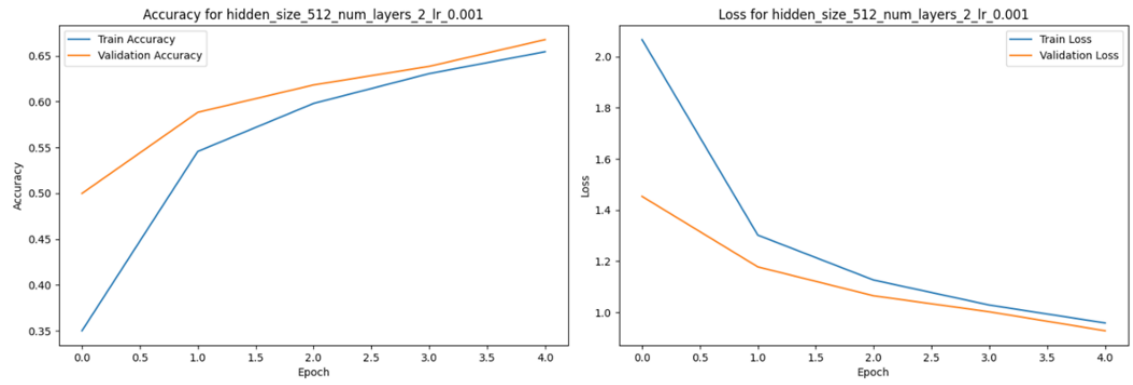


Figure 4.14: Optimized Hyperparameter Tuning for LSTM

RF

Random forest is fed with features extracted from the optimized LSTM model. Hyperparameter tuning is not performed on RF because it is used only as a classifier based on LSTM features extracted. Table 4.8 shows the pre-defined hyperparameters of the RF model.

Hyper-parameter	**Value**
Estimators	100
Random Seed	42

Table 4.8: Hyper-Parameters Used for RF

4.4 Summary

The implementation of the proposed solution for the SFDC system is described in this chapter, with a focus on the gasoline engine case study. Gasoline engines play a critical role in contemporary automotive systems, and understanding their intricate workings is essential for enhancing engine performance and ensuring reliable operation. Faulty data is collected by injecting fault into HIL systems. To inject the fault model is prepared using Simulink. Automatic code is generated using the dSPACE configuration desk based on the model prepared. With the use of a control desk, auto-generated code is flashed into hardware. All the process is carried out in user-friendly GUI.

Different kinds and different combinations of faults are injected. Data of faulty and healthy is collected in file format and preprocessed into training, testing and validation data sets. Data is split in the form of 80%:10%:10% of training: testing: validation respectively. Since the data is imbalanced, data balancing is carried out before it is fed into different models. DAE and LSTM models are fed with balanced training data set for training. Later hyperparameter tuning is performed on a validation data set of LSTM to make sure to select optimized hyperparameters. DAE and RF have their hyperparameters predefined.

5 Results and Discussion

In this chapter, the evaluation results using the test dataset will be discussed. Optimized models are used to conclude results. We will be evaluation DAE, LSTM classifier and LSTM+RF classifier for single and simultaneous faults. The models are evaluated using different matrices like balanced accuracy, loss, precision, recall, OOB score and reconstruction error. Also, finally, the evaluation of the model is done based on different levels of noise and different healthy: faulty data ratios.

5.1 Evaluation Metrics

A machine learning model's performance and efficacy are measured using evaluation metrics. The choice of evaluation metrics relies on the precise job and the nature of the problem being solved. Here are some typical assessment metrics:

- **Confusion Matrix** A table that enables the assessment and comprehension of a classification model's performance is known as a confusion matrix. It gives a thorough overview of how the model's predictions stack up against the dataset's actual labels. The confusion matrix aids in determining the sorts of errors the model makes and evaluating its accuracy.

- **Accuracy** Accuracy is a regularly and most used evaluation metric that gauges a classification model's overall accuracy. It computes the proportion of accurately predicted occurrences in the dataset based on the total number of instances. In other words, accuracy measures how often the model's predictions are right. The following equation 5.1 is used to compute it.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

  Where, TP represents the number of true positives (correctly predicted positive instances), TN represents the number of true negatives (correctly predicted negative instances), FP rep-

resents the number of false positives (incorrectly predicted positive instances), and FN represents the number of false negatives (incorrectly predicted negative instances).

When the classes in the dataset are balanced, meaning they have roughly equal representation, accuracy is a relevant metrics. However, in circumstances when there is a considerable class imbalance, accuracy might be deceiving since the model may obtain high accuracy by simply predicting the majority class most of the time. Additional evaluation criteria, such as balanced accuracy, can offer a more realistic assessment of the model's performance in such instances [Johnson and Khoshgoftaar, 2019].

- **Balanced Accuracy** A Balanced accuracy considers the class distribution in the dataset and gives a more balanced evaluation, especially in cases with imbalanced classes. It computes the average of the sensitivity (recall) for each class. Sensitivity is the percentage of accurately predicted instances of a given class out of all instances of that class. Balanced accuracy is calculated using he equation 5.2

$$Balanced\ Accuracy = \frac{Sensitivity(Class\ 1) + Sensitivity(Class\ 2) + \ldots + Sensitivity(Class\ N)}{N} \tag{5.2}$$

- **Precision** Precision-evaluated matric that qualifies the accuracy of a classification model's positive predictions. It calculates the fraction of real positive predictions (positive cases accurately anticipated) out of all positive predictions, including both true positives and false positives. The following formula is used to calculate the precision. The equation 5.3 can help you understand the mathematical representation of precision.

$$Precision = \frac{TP}{TP + FP} \tag{5.3}$$

- **Recall** Also known as sensitivity or true positive rate, is an assessment statistic that assesses a classification model's ability to properly identify all positive cases. It calculates the fraction of true positive predictions (positive cases successfully predicted) among all positive instances. The following equation 5.4 is used to calculate recall.

$$Recall = \frac{TP}{TP + FN} \tag{5.4}$$

- **F1-Score** The F1 score is a combination of precision and recalls matrics. It assesses a model's performance in a balanced way by taking into account both the capacity to properly identify positive examples (precision) and the ability to capture all positive occurrences (recall). Equation 5.5is used to calculate the F1 score.

$$F1score = \frac{2 \times (precision \times recall)}{precision + recall} \tag{5.5}$$

- **OOB Score** In random forests, the Out-of-Bag (OOB) score is a statistic used to evaluate model performance without the requirement for a separate validation set. It is calculated by averaging each data point's prediction accuracy over all trees in the random forest that did not include that data point during training. Mathematically OOB score is calculated as in equation 5.6

$$OOBScore = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \tag{5.6}$$

- **Reconstruction Error** The reconstruction error is a statistic used in denoising autoencoders to evaluate the model's success in recreating the original, clean input from the noisy input. It calculates the difference between the reconstructed output and the original input. Let X be the original, clean input data, and X_{hat} be the reconstructed output from the denoising autoencoder. The reconstruction error E is calculated using a chosen loss function, such as mean squared error (MSE).

$$E = \frac{1}{N} \sum ||X - \hat{X}||^2 \tag{5.7}$$

- **AUC-ROC Curve** The ROC curve depicts the True Positive Rate (TPR) vs the False Positive Rate (FPR) at various classification thresholds. The area under this curve, which varies from 0 to 1, is referred to as the AUC-ROC. A higher AUC-ROC score suggests that the model has superior discriminating and prediction ability.

5.2 DAE Model

DAE is the very first machine learning model used in this thesis work. As discussed before, DAE is used for the task of denoising noisy data. To make DAE more robust, it is trained with different levels of noise from 1% to 30%. The reconstruction error of the DAE varied from

0.0013 to 0.0879, depending on the noise level, which indicates promising DAE robustness towards denoising noisy data. The low reconstruction error suggests that DAE was able to successfully construct denoised data from the noisy data. Different reconstruction errors with levels of noise are represented in Table 5.1, and the diagrammatic representation of the reconstruction error to noise level is shown in Figure 5.1.

Noise Level %	Reconstruction Error
1	0.0013
4	0.0041
8	0.0094
10	0.0103
15	0.0272
20	0.0376
30	0.0879

Table 5.1: Reconstruction Error According to Different Noise Level

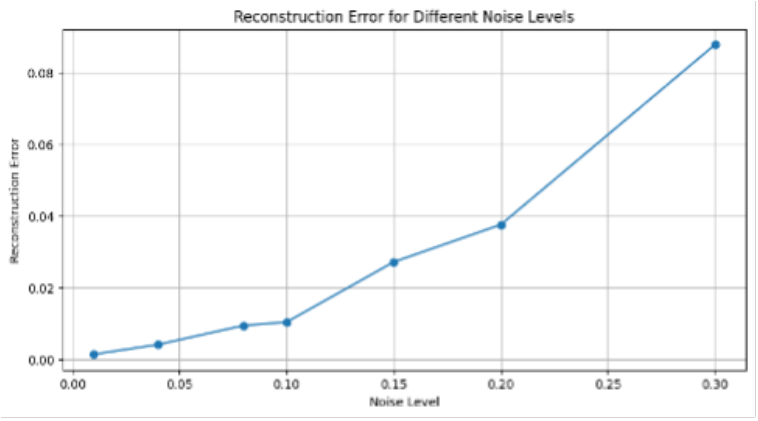


Figure 5.1: Example Noisy and Denoised Data

5.3 LSTM Model

LSTM-based model is used for 2 different tasks. The first one is featuring extraction and the second one is classification. LSTM is used for feature extraction and later extracted features

are used for classification purposes. LSTM model is classifying both single and simultaneous faults. After model training, accuracy and loss are calculated for training, testing and validation data sets. The complete measurement of values for all 3 sets is calculated for every epoch and is represented in 5.2.

Model test accuracy higher than validation accuracy and validation accuracy higher than training accuracy indicates that the model is performing well and has a good ability to generalize to new, unseen data. Since the model has been fine-tuned or optimized well with the grid search method and using the validation set, which can lead to better performance on the testing set. Similarly, this is the reason why test loss is lesser than validation loss and validation loss is lesser than train loss. In thinking of underfitting, underfitting happens when a model is too simple or lacks the capacity to capture the underlying patterns in the data, resulting in low accuracy on both training and testing/validation data. Since the model achieves reasonably high accuracy on both the validation and testing sets, underfitting is less likely to be the issue. Model is continuously learning and if you look into epochs of validation data it has continuous ups and downs trying to simulate proper learning. Also as discussed before LSTM is very good at capturing temporal dependencies of the time series data which could also be a possibility that the model has generalized well with the training data set.

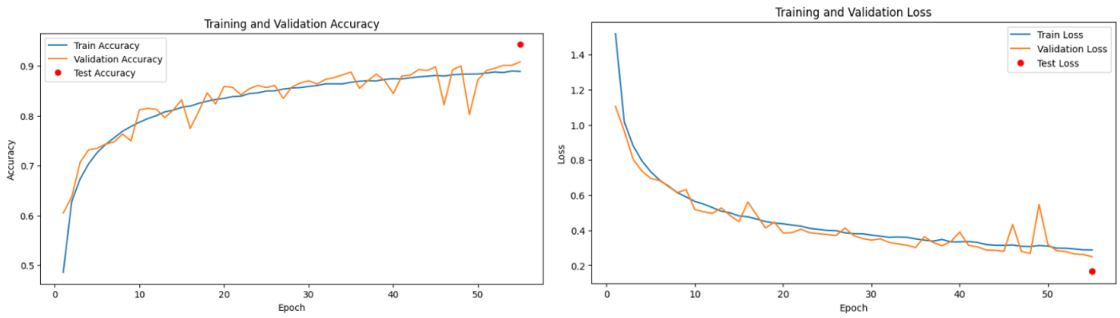


Figure 5.2: Loss and Accuracy calculation of LSTM Training Model

The model achieved good balanced accuracy of 95.4%. The average precision of a single fault is calculated as 93.9% and the average precision of a simultaneous fault is 85.8%. Recall is 89.3% and 87.7% for single and simultaneous faults respectively. Average of F1 score for single fault is 91.4% and simultaneous fault is 86%. From all the performance matrices, it is known that the model was able to learn and classify single faults better than multiple faults.

This could be due to the complex patterns that exist in simultaneous faults. The confusion matrix of LSTM, which seems to be having good results, is displayed with classes in Figure 5.3.

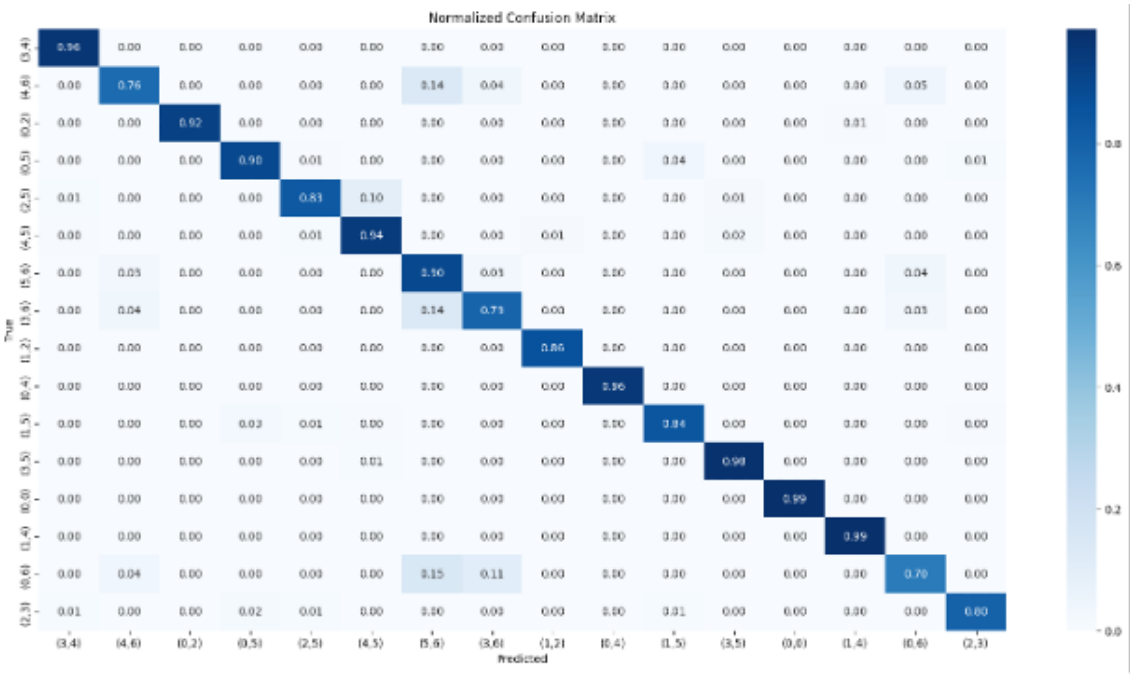


Figure 5.3: Confusion Matrix of LSTM Based Model

Please note the class numbers are mapped according to table 4.3. Some example class labels are class (0,0) stands for healthy, (0,2) means single Noise fault and (1,2) stands for simultaneous fault of Gain and Noise.

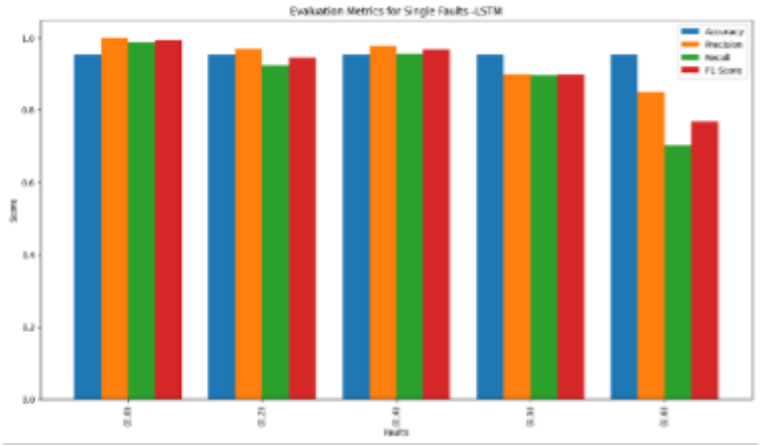


Figure 5.4: Single Fault Classification Performance of LSTM Model

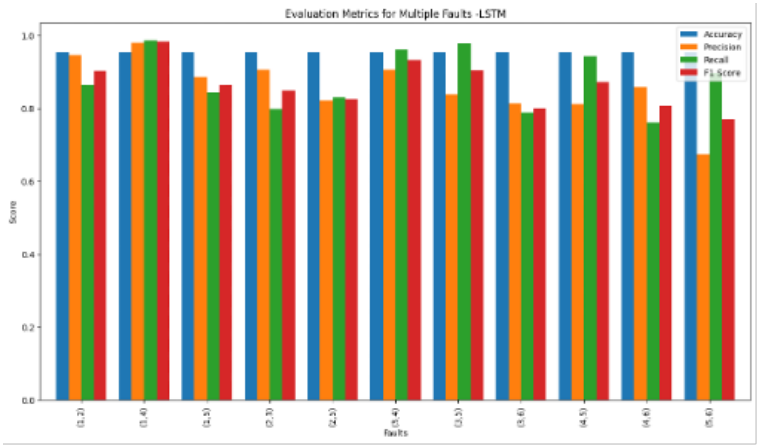


Figure 5.5: Simultaneous Fault Classification Performance of LSTM Model

5.4 Serial LSTM + RF Model

In parallel LSTM and RF, LSTM model is used for feature extraction and RF for classification. Figure 5.6 represents model architecture. Based on the results of the parallel model, the highest performing class among the simultaneous faults was the fault combination (1,4), which exhibited high precision, recall, and F1 score. On the other hand, the model achieved its lowest performance in terms of precision, recall, and F1 score for the fault combination (0,2) among the single faults. This indicates that the model had more difficulty accurately identifying this specific fault combination. The model performed better on the single fault with 71%, 74% and 72% precision, recall and F1 score respectively. Whereas precision, recall and F1 score is 69%, 71% and 70% respectively. Hence, we can conclude that the model is performing better for single faults over simultaneous faults.

Figure 5.6: Series LSTM (feature extraction) + RF (classification)

The performance of the model is shown in Figures 5.7 and 5.8for single and simultaneous faults. The performance of RF as a classifier is also measured using OOB score. RF OOB

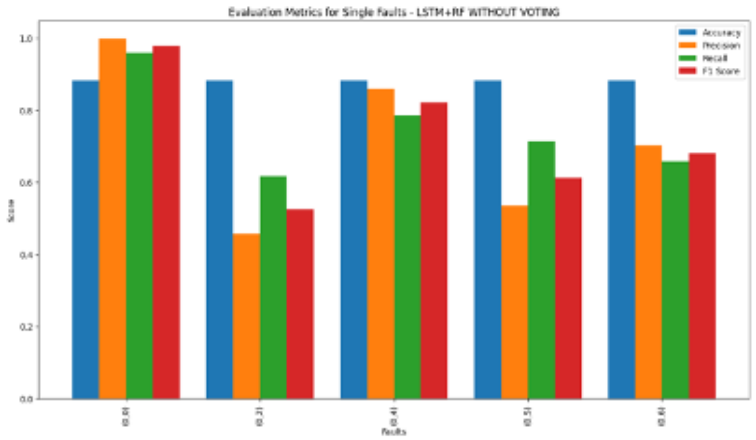


Figure 5.7: Single Fault Classification Performance of LSTM + RF Model

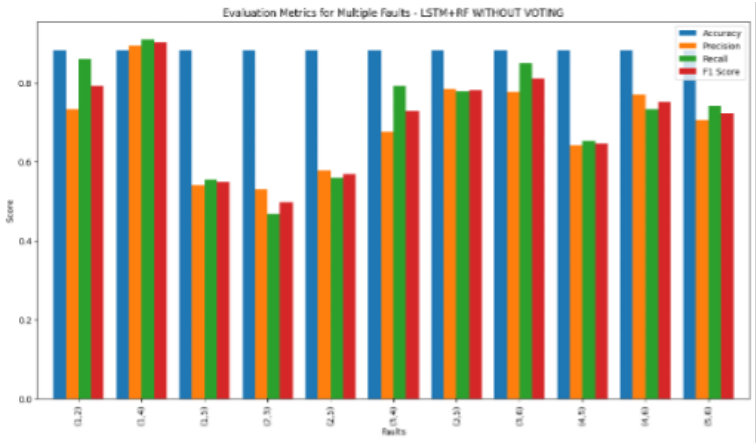


Figure 5.8: Simultaneous Fault Classification Performance of LSTM + RF Model

score is shown in Figure 5.9 which indicates good learning with improving results.

5.5 Parallel LSTM+ RF Model

In this model predictions from LSTM which is used for feature extraction classification and predictions from LSTM for feature extraction RF classifier are combined using voter, this is illustrated in Figure 5.10. A voter takes the predictions from both models and combines them based on the highest number of combined prediction counts.

With the use of parallel models, balanced accuracy of 95.5% is achieved which is almost similar to the LSTM model alone. The average precision value for a single fault is calculated

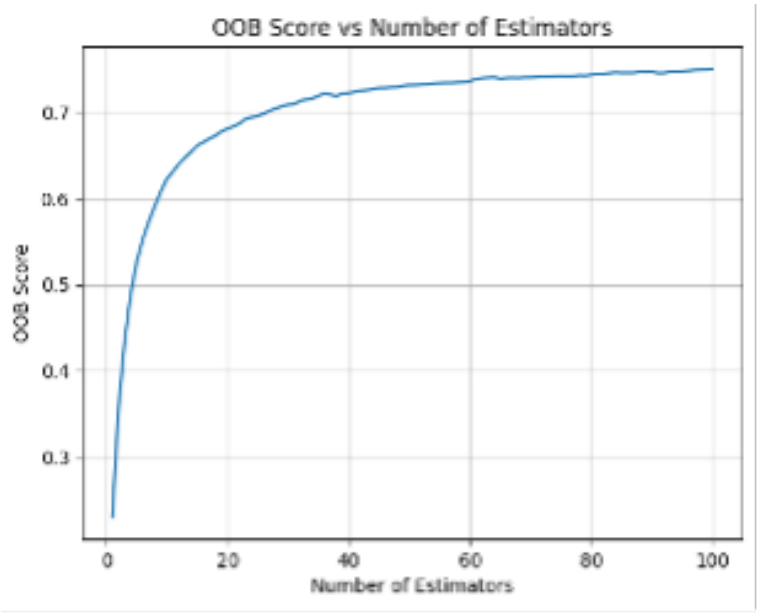


Figure 5.9: RF OOB Score

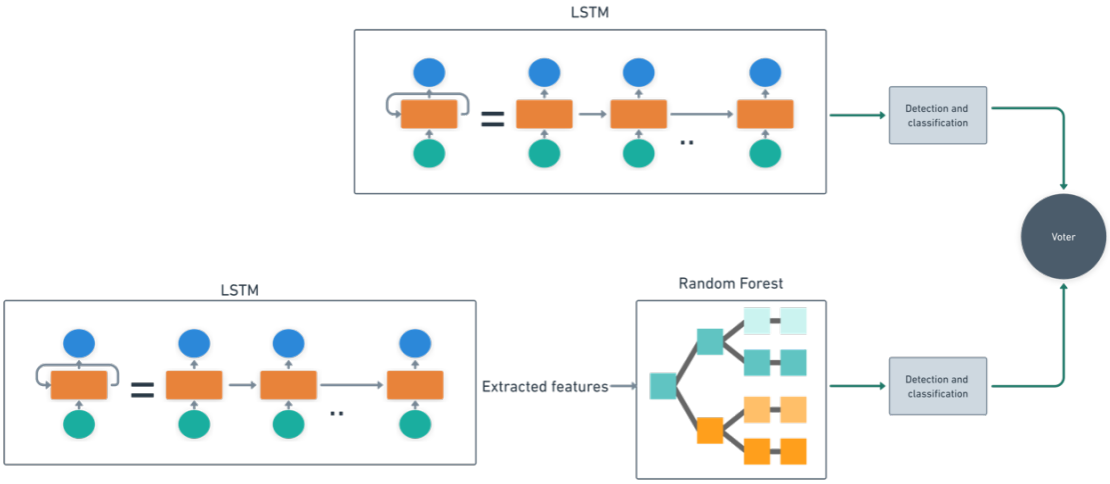


Figure 5.10: LSTM (Feature extraction+ Classification) LSTM (feature extraction) + RF(Classification)

as 95.4% and 85.2% for simultaneous faults. Recall has a value of 86.4% for simultaneous faults and 89.0% for single faults. F1 score is calculated as 92% and 85.6% for single and simultaneous faults respectively. The performance of LSTM+ RF for single and simultaneous faults are shown in Figure 5.11 and 5.12.

Table 5.2 shows the average performance matrices of the different models which are men-

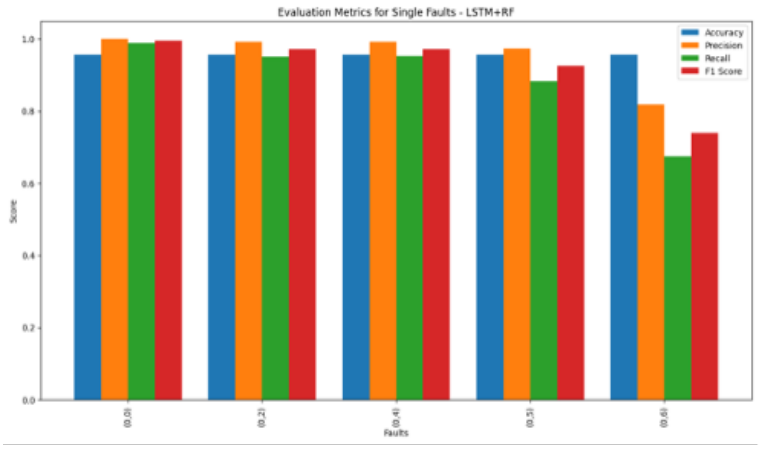


Figure 5.11: LSTM (Single Fault Classification Performance of LSTM+RF+Voter Model

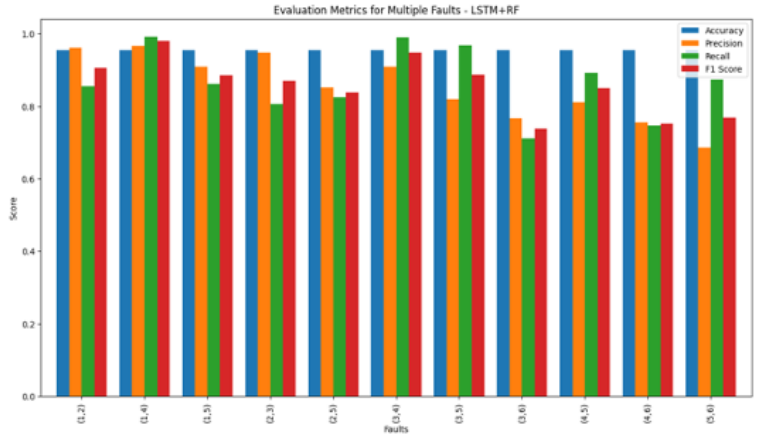


Figure 5.12: Simultaneous Fault Classification Performance of LSTM+RF+Voter Model

tioned above.

ML Model %	Accuracy	Precision	Recall	F1 Score
LSTM	95.4%	88.3%	88.2%	88.0%
LSTM+ RF	88.2%	69.9%	72.7%	71.1%
LSTM+ RF+ Voter	95.5%	85.2%	86.5%	85.6%

Table 5.2: Performance overview of different SFDC models

5.5.1 Robustness for noise

LSTM+ RF model is testing opposite to different noise levels. Different noise levels tested are, 1%, 4%, 8%, 15%, 20% and 30%. It is observed that an increase in the noise level has decreased the performance of all the metrics. Accuracy varied from 95.3% to 88% and precision is calculated from 95.9% to 91.2%. Recall is between 95.5% to 88.1% whereas the F1 score is calculated as 95.2% to 89.6%. You can observe the accuracy and loss calculations of each epoch in Figure 5.13.

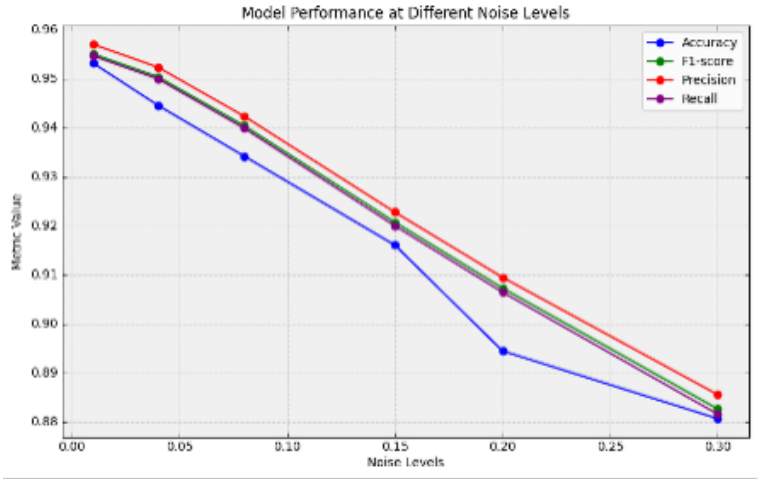


Figure 5.13: Model robustness to different noise levels

As the noise level increases, there is a slight decrease in accuracy, precision, recall, and F1 score. This suggests that the model's performance is affected by the presence of noise, which is expected. However, even at higher noise levels (e.g., 0.3), the model still maintains relatively high accuracy, precision, recall, and F1-score. This indicates that the model has some resilience to noise and can still make reasonably accurate predictions. The decreasing trend in performance metrics as the noise level increases suggests that the model's robustness decreases with higher noise levels. However, the decline is gradual, indicating that the model can handle moderate levels of noise fairly well.

5.5.2 Robustness for Unbalanced Data

Performance with Increased Healthy Data

The test accuracy for the model is 0.9501643258475708, indicating a high level of overall accuracy in predicting the labels. The precision, recall, and F1-score values for most classes are relatively high, ranging from 0.85 to 1.00. This suggests that the model is able to effectively classify the majority of the classes, with high precision and recall. The average F1-score is 0.95, suggesting that the model performs well across all classes, with a higher emphasis on the larger classes.

Performance with Decreased Healthy Data

The model achieves a test accuracy of 0.9126, slightly lower than the performance with increased healthy data. Most classes demonstrate reasonably high precision, recall, and F1 scores, ranging from 0.76 to 1.00. However, there is a slight decrease in performance compared to the scenario with increased healthy data. The weighted average F1-score is 0.91, indicating a slight reduction in performance compared to the increased healthy data scenario while maintaining a similar overall balance between precision and recall. This could be because the model is able to classify healthy patterns easily over complex faulty patterns. As discussed previously model had good accuracy for healthy signal classification hence the increase of healthy data also increased the overall performance of the model.

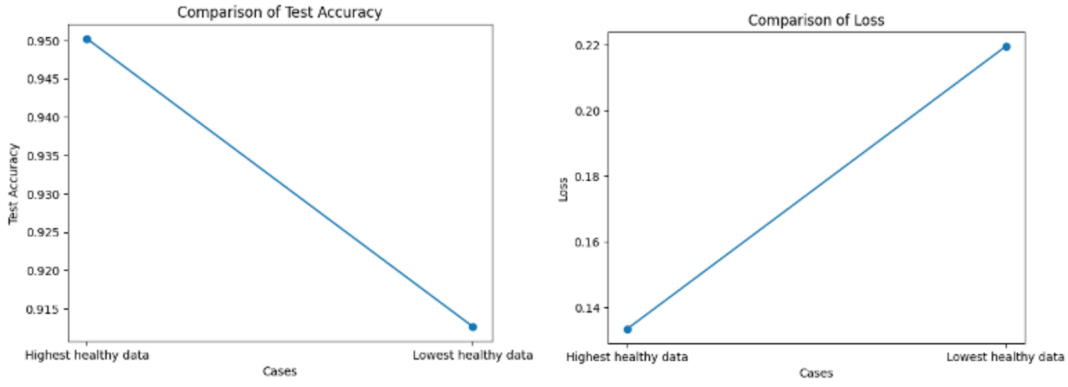


Figure 5.14: Accuracy and Loss Calculation Against Increased And Decreased Healthy Data

6 Conclusion and Future work

In this thesis work, machine learning-based SFDC is developed and implemented. Initially to develop and implement SFDC data is collected in the HIL environment. A real-time fault injection framework is used to inject and collect data with various single and multiple faults. Types of sensor signal faults have been considered, and different combinations of the faults are also considered. With faulty data healthy data is also collected. To address the issue of unbalanced data, an auto-balancing technique called random under-sampling was employed to prevent the development of biased models. In order to ensure robustness against noise, a Denoising Autoencoder (DAE) was incorporated to effectively reconstruct noisy data into noise-free data. For feature extraction and classification tasks, the power of Long Short-Term Memory (LSTM) networks was harnessed. LSTM networks are known for their ability to model and capture long-term dependencies in sequential data. Additionally, the potential of Random Forest (RF) was explored, which can capture complex non-linear relationships between features and the target variable, making it suitable for modelling time-dependent patterns in time series data. LSTM was used to extract features, and RF was used for classification based on the extracted features. Hyperparameter tuning was performed for LSTM to obtain suitable parameters. Additionally, a voter for a parallel model was experimented with, combining LSTM and RF predictions to benefit from both algorithms. The accuracy of the LSTM model alone was found to be 95.4%, while the LSTM+RF combination achieved an accuracy of 95.5%, demonstrating a marginal improvement. The utilization of the voter for model combination proved to be the best approach due to the similarities in performance. The trained LSTM+RF model was validated against different noise levels and various healthy-to-faulty ratios and as expected model performance has decreased as the noise level increased. Based on the performance metrics of the LSTM+RF model, it can be confidently used for SFDC applications. This research opens the door for further extensions, such as the identification of the fault location and timing of each fault type in simultaneous faults. The

same model can be extended to detect and classify simultaneous faults with more than two combinations, employing the same power set labelling technique. In conclusion, this thesis work presents a novel approach to automotive sensor fault detection and classification using machine learning techniques. The integration of LSTM and RF models through the voter demonstrates promising results, offering unique insights and potential implications for enhancing vehicle safety, reducing maintenance costs, and improving overall reliability in the automotive industry.

Bibliography

[dsp,] Rti can multimessage blockset. [Online]. Available.

[aut, 2019] (2019). Automotive simulation models. [Online]. Available.

[Abbas et al., 2019] Abbas, M., Memon, K. A., Jamali, A. A., Memon, S., and Ahmed, A. (2019). Multinomial naive bayes classification model for sentiment analysis. *IJCSNS Int. J. Comput. Sci. Netw. Secur*, 19(3):62.

[Abboush et al., 2022a] Abboush, M., Bamal, D., Knieke, C., and Rausch, A. (2022a). Hardware-in-the-loop-based real-time fault injection framework for dynamic behavior analysis of automotive software systems. *Sensors*, 22(4):1360.

[Abboush et al., 2022b] Abboush, M., Bamal, D., Knieke, C., and Rausch, A. (2022b). Intelligent fault detection and classification based on hybrid deep learning methods for hardware-in-the-loop test of automotive software systems. *Sensors*, 22(11):4066.

[Adarsh et al., 2019] Adarsh, R., Patil, A., Rayar, S., and Veena, K. (2019). Comparison of vader and lstm for sentiment analysis. *International Journal of Recent Technology and Engineering*, 7(6):540–543.

[Ahmad et al., 2020] Ahmad, N., Meng, A., and Sultan, M. (2020). Applications of hardware-in-the-loop simulation in automotive embedded systems. Technical report, SAE Technical Paper.

[Alvarez-Gonzalez et al., 2018] Alvarez-Gonzalez, F., Griffo, A., and Wang, B. (2018). Permanent magnet synchronous machines inter-turn short circuit fault detection by means of model-based residual analysis. In *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*, pages 647–652. IEEE.

[Amalfitano et al., 2014] Amalfitano, D., Fasolino, A. R., Scala, S., and Tramontana, P. (2014). Towards automatic model-in-the-loop testing of electronic vehicle information

centers. In *Proceedings of the 2014 international workshop on Long-term industrial collaboration on software engineering*, pages 9–12.

[Aziz et al., 2020] Aziz, F., Haq, A. U., Ahmad, S., Mahmoud, Y., Jalal, M., and Ali, U. (2020). A novel convolutional neural network-based approach for fault classification in photovoltaic arrays. *IEEE Access*, 8:41889–41904.

[Bedi et al., 2021] Bedi, P., Gupta, N., and Jindal, V. (2021). I-siamids: an improved siamids for handling class imbalance in network-based intrusion detection systems. *Applied Intelligence*, 51:1133–1151.

[Bergen et al., 2019] Bergen, K. J., Johnson, P. A., de Hoop, M. V., and Beroza, G. C. (2019). Machine learning for data-driven discovery in solid earth geoscience. *Science*, 363(6433):eaau0323.

[Biddle and Fallah, 2021] Biddle, L. and Fallah, S. (2021). A novel fault detection, identification and prediction approach for autonomous vehicle controllers using svm. *Automotive Innovation*, 4:301–314.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

[Chakraborty et al., 2019] Chakraborty, D., Sur, U., and Banerjee, P. K. (2019). Random forest based fault classification technique for active power system networks. In *2019 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*, pages 1–4. IEEE.

[Chandar et al., 2019] Chandar, S., Sankar, C., Vorontsov, E., Kahou, S. E., and Bengio, Y. (2019). Towards non-saturating recurrent units for modelling long-term dependencies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3280–3287.

[Chen et al., 2020] Chen, J., Chen, S., Ma, C., Jing, Z., and Xu, Q. (2020). Fault detection of aircraft control system based on negative selection algorithm. *International Journal of Aerospace Engineering*, 2020:1–10.

[Chen et al., 2023] Chen, Z., O'Neill, Z., Wen, J., Pradhan, O., Yang, T., Lu, X., Lin, G., Miyata, S., Lee, S., Shen, C., et al. (2023). A review of data-driven fault detection and diagnostics for building hvac systems. *Applied Energy*, 339:121030.

[Cheng and Wang, 2007] Cheng, T. and Wang, J. (2007). Application of a dynamic recurrent neural network in spatio-temporal forecasting. In *Information Fusion and Geographic Information Systems: Proceedings of the Third International Workshop*, pages 173–186. Springer.

[Chiang et al., 2019] Chiang, H.-T., Hsieh, Y.-Y., Fu, S.-W., Hung, K.-H., Tsao, Y., and Chien, S.-Y. (2019). Noise reduction in ecg signals using fully convolutional denoising autoencoders. *Ieee Access*, 7:60806–60813.

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[Chung et al., 1990] Chung, S.-H., Moore, J. B., Xia, L., Premkumar, L., and Gage, P. W. (1990). Characterization of single channel currents using digital signal processing techniques based on hidden markov models. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 329(1254):265–285.

[Coble et al., 2013] Coble, J., Ramuhalli, P., Meyer, R., and Hashemian, H. (2013). Online sensor calibration assessment in nuclear power systems. *IEEE Instrumentation & Measurement Magazine*, 16(3):32–37.

[De Bruin et al., 2016] De Bruin, T., Verbert, K., and Babuška, R. (2016). Railway track circuit fault diagnosis using recurrent neural networks. *IEEE transactions on neural networks and learning systems*, 28(3):523–533.

[Dhande and Patnaik, 2014] Dhande, L. L. and Patnaik, G. (2014). Review of sentiment analysis using naive bayes and neural network classifier. *Int. J. Sci. Eng. Technol. Res*, 3(07):1110–1113.

[Ding et al., 2019] Ding, C., Cao, X., and Liu, C. (2019). How does the station-area built environment influence metrorail ridership? using gradient boosting decision trees to identify non-linear thresholds. *Journal of Transport Geography*, 77:70–78.

[Drenth et al., 2014] Drenth, E., Törmänen, M., Johansson, K., Andersson, B.-A., Andersson, D., Torstensson, I., and Åkesson, J. (2014). Consistent simulation environment with

fmi based tool chain. In *Proceedings of the 10 th International Modelica Conference*, pages 1277–1283.

[Dutta et al., 2022] Dutta, N., Kaliannan, P., and Shanmugam, P. (2022). Application of machine learning for inter turn fault detection in pumping system. *Scientific Reports*, 12(1):12906.

[Fabarisov et al., 2021] Fabarisov, T., Mamaev, I., Morozov, A., and Janschek, K. (2021). Model-based fault injection experiments for the safety analysis of exoskeleton system. *arXiv preprint arXiv:2101.01283*.

[Fan et al., 2020] Fan, S.-K. S., Hsu, C.-Y., Tsai, D.-M., He, F., and Cheng, C.-C. (2020). Data-driven approach for fault detection and diagnostic in semiconductor manufacturing. *IEEE Transactions on Automation Science and Engineering*, 17(4):1925–1936.

[Fragkoulis et al., 2009] Fragkoulis, D., Roux, G., and Dahhou, B. (2009). Sensor fault detection and isolation for single, multiples and simultaneous faults: Application to a waste water treatment process. *IFAC Proceedings Volumes*, 42(11):934–939.

[Gao et al., 2015] Gao, Z., Cecati, C., and Ding, S. X. (2015). A survey of fault diagnosis and fault-tolerant techniques—part i: Fault diagnosis with model-based and signal-based approaches. *IEEE transactions on industrial electronics*, 62(6):3757–3767.

[Gong et al., 2019] Gong, S., Meng, S., Wang, B., and Liu, D. (2019). Hardware-in-the-loop simulation of uav for fault injection. In *2019 Prognostics and System Health Management Conference (PHM-Qingdao)*, pages 1–6. IEEE.

[Grimaldi and Manto, 2010] Grimaldi, G. and Manto, M. (2010). Neurological tremor: sensors, signal processing and emerging applications. *Sensors*, 10(2):1399–1422.

[Gupta et al., 2021] Gupta, S., Aga, D., Pruden, A., Zhang, L., and Vikesland, P. (2021). Data analytics for environmental science and engineering research. *Environmental Science & Technology*, 55(16):10895–10907.

[Hahne et al., 2008] Hahne, F., Huber, W., Gentleman, R., Falcon, S., Gentleman, R., and Carey, V. (2008). Unsupervised machine learning. *Bioconductor case studies*, pages 137–157.

[Han et al., 2012] Han, S. H., Lu, S. X., and Leung, S. C. (2012). Segmentation of telecom customers based on customer value by decision tree model. *Expert Systems with Applications*, 39(4):3964–3973.

[Hazra et al., 2017] Hazra, T. K., Singh, D. P., and Daga, N. (2017). Optical character recognition using knn on custom image dataset. In *2017 8th Annual Industrial Automation and Electromechanical Engineering Conference (IEMECON)*, pages 110–114. IEEE.

[He et al., 2020] He, Y., Chen, R., Li, X., Hao, C., Liu, S., Zhang, G., and Jiang, B. (2020). Online at-risk student identification using rnn-gru joint neural networks. *Information*, 11(10):474.

[Hua et al., 2023] Hua, C., Xiong, L., Lv, L., Dong, D., and Ouyang, H. (2023). Multi-fault classification of rotor systems based on phase feature of axis trajectory in noisy environments. *Structural Health Monitoring*, page 14759217231178652.

[Ince et al., 2016] Ince, T., Kiranyaz, S., Eren, L., Askar, M., and Gabbouj, M. (2016). Real-time motor fault detection by 1-d convolutional neural networks. *IEEE Transactions on Industrial Electronics*, 63(11):7067–7075.

[Iwana and Uchida, 2021] Iwana, B. K. and Uchida, S. (2021). An empirical survey of data augmentation for time series classification with neural networks. *Plos one*, 16(7):e0254841.

[Izquierdo-Verdiguier and Zurita-Milla, 2020] Izquierdo-Verdiguier, E. and Zurita-Milla, R. (2020). An evaluation of guided regularized random forest for classification and regression tasks in remote sensing. *International Journal of Applied Earth Observation and Geoinformation*, 88:102051.

[Jadhav and Channe, 2016] Jadhav, S. D. and Channe, H. (2016). Comparative study of k-nn, naive bayes and decision tree classification techniques. *International Journal of Science and Research (IJSR)*, 5(1):1842–1845.

[Johnson and Khoshgoftaar, 2019] Johnson, J. M. and Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):1–54.

[Kazemi et al., 2021] Kazemi, Z., Naseri, F., Yazdi, M., and Farjah, E. (2021). An ekf-svm machine learning-based approach for fault detection and classification in three-phase power transformers. *IET Science, Measurement & Technology*, 15(2):130–142.

[Khoshgoftaar et al., 2007] Khoshgoftaar, T. M., Golawala, M., and Van Hulse, J. (2007). An empirical study of learning from imbalanced data using random forest. In *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, volume 2, pages 310–317. IEEE.

[Khuat and Le, 2019] Khuat, T. T. and Le, M. H. (2019). Ensemble learning for software fault prediction problem with imbalanced data. *International Journal of Electrical & Computer Engineering (2088-8708)*, 9(4).

[Kosmopoulos et al., 2008] Kosmopoulos, A., Paliouras, G., and Androutsopoulos, I. (2008). Adaptive spam filtering using only naive bayes text classifiers. In *Proceedings of the Fifth Conference on Email and Anti-Spam (CEAS)*, pages 1–2.

[Laska et al., 2011] Laska, J. N., Boufounos, P. T., Davenport, M. A., and Baraniuk, R. G. (2011). Democracy in action: Quantization, saturation, and compressive sensing. *Applied and Computational Harmonic Analysis*, 31(3):429–443.

[Lee et al., 2016] Lee, H., Kim, Y., and Kim, C. O. (2016). A deep learning model for robust wafer fault monitoring with sensor measurement noise. *IEEE Transactions on Semiconductor Manufacturing*, 30(1):23–31.

[Li et al., 2023] Li, H., Makkapati, V. P., Wan, L., Tomasch, E., Hoschopf, H., and Eichberger, A. (2023). Validation of automated driving function based on the apollo platform: A milestone for simulation with vehicle-in-the-loop testbed. *Vehicles*, 5(2):718–731.

[Li et al., 2019] Li, Y., Gu, S., Gool, L. V., and Timofte, R. (2019). Learning filter basis for convolutional neural network compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5623–5632.

[Li et al., 2014] Li, Z., Wang, J., Gao, J., Li, B., and Zhou, F. (2014). A vondrak low pass filter for imu sensor initial alignment on a disturbed base. *Sensors*, 14(12):23803–23821.

[Lotz et al., 2019] Lotz, J., Vogelsang, A., Benderius, O., and Berger, C. (2019). Microservice architectures for advanced driver assistance systems: A case-study. In *2019 IEEE*

International Conference on Software Architecture Companion (ICSA-C), pages 45–52. IEEE.

[Lu et al., 2013] Lu, X., Tsao, Y., Matsuda, S., and Hori, C. (2013). Speech enhancement based on deep denoising autoencoder. In *Interspeech*, volume 2013, pages 436–440.

[Lumpp et al., 2014] Lumpp, B., Tanimou, M., McMackin, M., Bouillon, E., Trapel, E., Muenzenmay, M., and Zimmermann, K. (2014). Desktop simulation and calibration of diesel engine ecu software using software-in-the-loop methodology. Technical report, SAE Technical Paper.

[Mahesh, 2020] Mahesh, B. (2020). Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9(1):381–386.

[Majhi et al., 2019] Majhi, S. K., Bhatachharya, S., Pradhan, R., and Biswal, S. (2019). Fuzzy clustering using salp swarm algorithm for automobile insurance fraud detection. *Journal of Intelligent & Fuzzy Systems*, 36(3):2333–2344.

[Mihalič et al., 2022] Mihalič, F., Truntič, M., and Hren, A. (2022). Hardware-in-the-loop simulations: A historical overview of engineering challenges. *Electronics*, 11(15):2462.

[Mirchevska et al., 2018] Mirchevska, B., Pek, C., Werling, M., Althoff, M., and Boedecker, J. (2018). High-level decision making for safe and reasonable autonomous lane changing using reinforcement learning. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2156–2162. IEEE.

[Molina Llorente and Molina Llorente, 2020] Molina Llorente, R. and Molina Llorente, R. (2020). Embedded control system development process: Model-based design and architecture basics. *Practical Control of Electric Machines: Model-Based Design and Simulation*, pages 1–26.

[Namburu et al., 2007] Namburu, S. M., Chigusa, S., Prokhorov, D., Qiao, L., Choi, K., and Pattipati, K. (2007). Application of an effective data-driven approach to real-time time fault diagnosis in automotive engines. In *2007 IEEE Aerospace Conference*, pages 1–9. IEEE.

[Nasteski, 2017] Nasteski, V. (2017). An overview of the supervised machine learning methods. *Horizons. b*, 4:51–62.

[Ng and Winkler, 2014] Ng, H.-W. and Winkler, S. (2014). A data-driven approach to cleaning large face datasets. In *2014 IEEE international conference on image processing (ICIP)*, pages 343–347. IEEE.

[Oh et al., 2021] Oh, S., Han, S., and Jeong, J. (2021). Multi-scale convolutional recurrent neural network for bearing fault detection in noisy manufacturing environments. *Applied Sciences*, 11(9):3963.

[Ou et al., 2014] Ou, K., Rao, H., Cai, Z., Guo, H., Lin, X., Guan, L., Maguire, T., Warkentin, B., and Chen, Y. (2014). Mmc-hvdc simulation and testing based on real-time digital simulator and physical control system. *IEEE Journal of emerging and selected topics in Power Electronics*, 2(4):1109–1116.

[Palladino et al., 2012] Palladino, A., Fiengo, G., and Lanzo, D. (2012). A portable hardware-in-the-loop (hil) device for automotive diagnostic control systems. *ISA transactions*, 51(1):229–236.

[Park et al., 2015] Park, J., Ivanov, R., Weimer, J., Pajic, M., and Lee, I. (2015). Sensor attack detection in the presence of transient faults. In *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, pages 1–10.

[Patel and Giri, 2016] Patel, R. K. and Giri, V. (2016). Feature selection and classification of mechanical fault of an induction motor using random forest classifier. *Perspectives in Science*, 8:334–337.

[Pavithra et al., 2019] Pavithra, M., Saruladha, K., and Sathyabama, K. (2019). Gru based deep learning model for prognosis prediction of disease progression. In *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, pages 840–844. IEEE.

[Priyadarshini and Cotton, 2021] Priyadarshini, I. and Cotton, C. (2021). A novel lstm–cnn–grid search-based deep neural network for sentiment analysis. *The Journal of Supercomputing*, 77(12):13911–13932.

[Qin et al., 2018] Qin, Z., Yu, F., Liu, C., and Chen, X. (2018). How convolutional neural network see the world-a survey of convolutional neural network visualization methods. *arXiv preprint arXiv:1804.11191*.

[Rahman et al., 2016] Rahman, L., Mohammed, N., and Al Azad, A. K. (2016). A new lstm model by introducing biological cell state. In *2016 3rd International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)*, pages 1–6. IEEE.

[Ray, 2019] Ray, S. (2019). A quick review of machine learning algorithms. In *2019 International conference on machine learning, big data, cloud and parallel computing (COMITCon)*, pages 35–39. IEEE.

[Safavi et al., 2021] Safavi, S., Safavi, M. A., Hamid, H., and Fallah, S. (2021). Multi-sensor fault detection, identification, isolation and health forecasting for autonomous vehicles. *Sensors*, 21(7):2547.

[Shakya, 2021] Shakya, S. (2021). A self monitoring and analyzing system for solar power station using iot and data mining algorithms. *Journal of Soft Computing Paradigm*, 3(2):96–109.

[Shen et al., 2020] Shen, Q., Wu, Y., Jiang, Y., Zeng, W., Alexis, K., Vianova, A., and Qu, H. (2020). Visual interpretation of recurrent neural network on multi-dimensional time-series forecast. In *2020 IEEE Pacific visualization symposium (PacificVis)*, pages 61–70. IEEE.

[Shewalkar et al., 2019] Shewalkar, A., Nyavanandi, D., and Ludwig, S. A. (2019). Performance evaluation of deep neural networks applied to speech recognition: Rnn, lstm and gru. *Journal of Artificial Intelligence and Soft Computing Research*, 9(4):235–245.

[Susan and Kumar, 2021] Susan, S. and Kumar, A. (2021). The balancing trick: Optimized sampling of imbalanced datasets—a brief survey of the recent state of the art. *Engineering Reports*, 3(4):e12298.

[Theissler, 2017] Theissler, A. (2017). Detecting known and unknown faults in automotive systems using ensemble-based anomaly detection. *Knowledge-Based Systems*, 123:163–173.

[Ting et al., 2011] Ting, S., Ip, W., Tsang, A. H., et al. (2011). Is naive bayes a good classifier for document classification. *International Journal of Software Engineering and Its Applications*, 5(3):37–46.

[Venkatasubramanian et al., 2003] Venkatasubramanian, V., Rengaswamy, R., Yin, K., and Kavuri, S. N. (2003). A review of process fault detection and diagnosis: Part i: Quantitative model-based methods. *Computers & chemical engineering*, 27(3):293–311.

[Vincent et al., 2008] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103.

[Vishnukumar et al., 2017] Vishnukumar, H. J., Butting, B., Müller, C., and Sax, E. (2017). Machine learning and deep neural network—artificial intelligence core for lab and real-world test and validation for adas and autonomous vehicles: Ai for efficient and quality test and validation. In *2017 intelligent systems conference (IntelliSys)*, pages 714–721. IEEE.

[Wang et al., 2020] Wang, D., Du, B., Zhang, L., and Xu, Y. (2020). Adaptive spectral–spatial multiscale contextual feature extraction for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 59(3):2461–2477.

[Wang et al., 2014] Wang, W., Stuijk, S., and De Haan, G. (2014). Exploiting spatial redundancy of image sensor for motion robust rppg. *IEEE transactions on Biomedical Engineering*, 62(2):415–425.

[Wu et al., 2019] Wu, X., Jiang, G., Wang, X., Xie, P., and Li, X. (2019). A multi-level-denoising autoencoder approach for wind turbine fault detection. *Ieee Access*, 7:59376–59387.

[Xiao et al., 2018] Xiao, Y., Wu, J., Lin, Z., and Zhao, X. (2018). A deep learning-based multi-model ensemble method for cancer prediction. *Computer methods and programs in biomedicine*, 153:1–9.

[Yang et al., 2020] Yang, S., Yu, X., and Zhou, Y. (2020). Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example. In *2020 International workshop on electronic communication and artificial intelligence (IWECAI)*, pages 98–101. IEEE.

[Yang et al., 2021] Yang, X., Qiao, X., Cheng, C., Zhong, K., and Chen, H. (2021). A tutorial on hardware-implemented fault injection and online fault diagnosis for high-speed trains. *Sensors*, 21(17):5957.

[Yang, 2016] Yang, Y. (2016). *Temporal data mining via unsupervised ensemble learning*. Elsevier.

[Yap et al., 2014] Yap, B. W., Rani, K. A., Rahman, H. A. A., Fong, S., Khairudin, Z., and Abdullah, N. N. (2014). An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets. In *Proceedings of the first international conference on advanced data and information engineering (DaEng-2013)*, pages 13–22. Springer.

[Yin et al., 2014] Yin, S., Li, X., Gao, H., and Kaynak, O. (2014). Data-based techniques focused on modern industry: An overview. *IEEE Transactions on industrial electronics*, 62(1):657–667.

[Yu et al., 2015] Yu, Y., Li, W., Sheng, D., and Chen, J. (2015). A novel sensor fault diagnosis method based on modified ensemble empirical mode decomposition and probabilistic neural network. *Measurement*, 68:328–336.

[Zhang and Suganthan, 2014] Zhang, L. and Suganthan, P. N. (2014). Random forests with ensemble of feature spaces. *Pattern Recognition*, 47(10):3429–3437.

[Zhou et al., 2020] Zhou, F., Yang, S., Fujita, H., Chen, D., and Wen, C. (2020). Deep learning fault diagnosis method based on global optimization gan for unbalanced data. *Knowledge-Based Systems*, 187:104837.

[Zhou et al., 2021] Zhou, Z., Chen, H., Li, G., Zhong, H., Zhang, M., and Wu, J. (2021). Data-driven fault diagnosis for residential variable refrigerant flow system on imbalanced data environments. *International journal of refrigeration*, 125:34–43.