TU Clausthal

Mohammad Abboush

Machine Learning-based Intelligent Fault Detection and Diagnosis for Real-Time Validation Process of Automotive Software Systems during the Development Phase

ISSE | Institute for Software and Systems Engineering

Machine Learning-based Intelligent Fault Detection and Diagnosis for Real-Time Validation Process of Automotive Software Systems during the Development Phase

D o c t o r a l T h e s i s

(D i s s e r t a t i o n)

to be awarded the degree of

Doctor of Engineering

(Dr. -Ing.)

submitted by

Mohammad Abboush

from Latakia

approved by the Faculty of Mathematics/Computer Science and Mechanical Engineering, Clausthal University of Technology

Date of oral examination: 25 September 2024

Dissertation Clausthal, ISSE-Dissertation, 2024

- Chairperson of the Board of Examiners

  Prof. Dr. Jörg P. Müller

- Chief Reviewer

  Prof. Dr. Andreas Rausch

- 2. Reviewer

  PD Dr. Christoph Knieke

- 3. Reviewer

  Prof. Dr. Christian Siemers

Abstract

Ensuring the safety and reliability of today's modern automotive software systems (ASSs) in accordance with the ISO 26262 standard is a major concern of the automotive industry. The increasing complexity of such systems, including interconnected advanced functions with several hundred ECUs and several million lines of software code, poses an additional challenge for verification and validation during the development process. In the system validation phase of the V-model, a test drive is carried out under realistic conditions to evaluate the system properties, i.e. safety, reliability, robustness and performance. Typically, such a test is carried out using a vehicle prototype within a real or virtual environment. Alternatively, the test drive can also be carried out in the laboratory using a real-time hardware-in-the-loop (HIL) simulation with a real system, including sensors, actuators and control units.

At the present time, the failure analysis process of test drive records is carried out in one of two ways: either by employing an industrial-certified tool, or by applying knowledge-based analysis techniques. However, the industrial tools are limited in their capacity to identify anomalies, exhibiting a deficiency in their ability to discern critical faults and provide a reasonable interpretation of failure occurrence. Furthermore, the input of experienced professionals and a dedicated team is essential for a comprehensive assessment of the system's behavior and the identification of critical malfunctions. Consequently, the detection and identification of critical faults in the multivariate, multi-modal, nonlinear system behavior with a substantial data volume necessitates a considerable expenditure of effort and expertise, and is an extremely challenging and time-consuming process. Therefore, an effective approach for analyzing test recordings during the real-time validation process in an efficient manner is required.

The aim of this dissertation is to address the aforementioned challenge by proposing a novel approach to intelligently analyse the test records of a real-time validation process for ASSs during the development phase. For this purpose, a novel fault detection and diagnosis (FDD) model architecture has been developed based on data-driven machine learning (ML) and deep learning (DL) methods and utilizing historical representative datasets. Considering most sensor-related faults in ASSs, the developed approach is capable of detecting and identifying the nature of critical faults in different states, single and simultaneous occurrence, known and unknown faults. This, in turn, not

only improves the safety and reliability of automotive systems but also reduces the effort and time required during the development process.

In this dissertation, five novel contributions to the body of research on the FDD of ASSs during the analysis of test drive records are proposed and developed. Firstly, the work contributes to address the problem of unavailability of representative automotive faults datasets by proposing a novel real-time fault injection (FI) framework. This framework is designed to analyse the system behavior under abnormal conditions and determine the critical faults that cause the safety and functionality violation. As a consequence of the system analysis under fault conditions, a representative faults dataset is collected, which serves as the basis for the development of the FDD model. Secondly, a novel intelligent FDD model based on the hybrid DL methods is proposed, which demonstrates high accuracy and robustness compared to standalone methods. Third, the issue of concurrent FDD in the presence of imbalanced data has been addressed by proposing a novel methodology based on ensemble ML and DL method, i.e., random forest and LSTM. Fourth, an innovative robust model has been developed to detect and categorize single and concurrent unknown faults under noisy conditions. To this end, a gated recurrent unit-based denoising autoencoder with K-means-based multilevel clustering is developed. Finally, a HIL-based virtual test drive framework has been developed for the purpose of a real-time validation process in accordance with ISO 26262. The developed framework serves as a case study to validate the applicability of the proposed FDD models.

Zusammenfassung

Die Gewährleistung der Sicherheit und Zuverlässigkeit moderner Kfz Softwaresysteme (ASS) gemäß der Norm ISO 26262 ist ein wichtiges Anliegen der Automobilindustrie. Die zunehmende Komplexität solcher Systeme, einschließlich miteinander verbundener fortschrittlicher Funktionen mit mehreren hundert elektronischen Steuergeräten und mehreren Millionen Zeilen Softwarecode, stellt eine zusätzliche Herausforderung für die Verifizierung und Validierung während des Entwicklungsprozesses dar. In der Systemvalidierungsphase des V-Modells wird eine Testfahrt unter realistischen Bedingungen durchgeführt, um die Systemeigenschaften, d. h. Sicherheit, Zuverlässigkeit, Robustheit und Leistung, zu bewerten. In der Regel wird ein solcher Test mit einem Fahrzeugprototyp in einer realen Alternativ kann die Testfahrt auch im Labor mithilfe einer Echtzeit-Hardware-in-the-Loop-Simulation (HIL) mit einem realen System, einschließlich Sensoren, Aktoren und Steuergeräten, durchgeführt werden.

Derzeit wird die Fehleranalyse von Testfahrtaufzeichnungen auf zwei Arten durchgeführt: entweder mithilfe eines industriell zertifizierten Tools oder durch Anwendung wissensbasierter Analysetechniken. Die industriellen Tools sind jedoch nur begrenzt in der Lage, Anomalien zu erkennen, und weisen einen Mangel in ihrer Fähigkeit auf, kritische Fehler zu erkennen und eine angemessene Interpretation des Fehlerauftretens zu liefern. Darüber hinaus ist die Mitwirkung erfahrener Fachleute und eines engagierten Teams für eine umfassende Bewertung des Systemverhaltens und die Identifizierung kritischer Fehlfunktionen unerlässlich. Folglich erfordert die Erkennung und Identifizierung kritischer Fehler im multivariaten, multimodalen, nichtlinearen Systemverhalten mit einem beträchtlichen Datenvolumen einen erheblichen Aufwand an Arbeit und Fachwissen und ist ein äußerst anspruchsvoller und zeitaufwändiger Prozess. Daher ist ein effektiver Ansatz für die effiziente Analyse von Testaufzeichnungen während.

Das Ziel dieser Dissertation ist es, die oben genannte Herausforderung anzugehen, indem ein neuartiger Ansatz zur intelligenten Analyse der Testaufzeichnungen eines Echtzeit-Validierungsprozesses für ASSs während der Entwicklungsphase vorgeschlagen wird. Zu diesem Zweck wurde eine

neuartige Modellarchitektur zur Fehlererkennung und -diagnose (FDD) entwickelt, die auf datengesteuertem maschinellem Lernen (ML) und Deep-Learning-Methoden (DL) basiert und historische repräsentative Datensätze verwendet. Unter Berücksichtigung der meisten sensorbedingten Fehler in automatisierten Fahrzeugsystemen ist der entwickelte Ansatz in der Lage, die Art kritischer Fehler in verschiedenen Zuständen, einzeln und gleichzeitig auftretend, bekannte und unbekannte Fehler zu erkennen und zu identifizieren. Dies wiederum verbessert nicht nur die Sicherheit und Zuverlässigkeit von Fahrzeugsystemen, sondern reduziert auch den Aufwand und die Zeit, die während des Entwicklungsprozesses erforderlich sind.

In dieser Dissertation werden fünf neue Beiträge zum Forschungsbereich der Fehlererkennung und -diagnose von automatisierten Fahrzeugsystemen während der Analyse von Testfahrtaufzeichnungen vorgeschlagen und entwickelt. Erstens trägt die Arbeit dazu bei, das Problem der Nichtverfügbarkeit repräsentativer Fehlerdatensätze für Kraftfahrzeuge zu lösen, indem ein neuartiges Echtzeit-Fehlerinjektions-Framework (FI) vorgeschlagen wird. Dieses Framework ist darauf ausgelegt, das Systemverhalten unter abnormalen Bedingungen zu analysieren und die kritischen Fehler zu ermitteln, die die Sicherheits- und Funktionsverletzung verursachen. Als Ergebnis der Systemanalyse unter Fehlerbedingungen wird ein repräsentativer Fehlerdatensatz gesammelt, der als Grundlage für die Entwicklung des FDD-Modells dient. Zweitens wird ein neuartiges intelligentes FDD-Modell auf der Grundlage der hybriden DL-Methoden vorgeschlagen, das im Vergleich zu eigenständigen Methoden eine hohe Genauigkeit und Robustheit aufweist. Drittens wurde das Problem der gleichzeitigen FDD bei Vorhandensein unausgewogener Daten durch den Vorschlag einer neuartigen Methodik auf der Grundlage der Ensemble-ML- und DL-Methode, insbesondere Random Forest und LSTM, angegangen. Viertens wurde ein innovatives robustes Modell entwickelt, um einzelne und gleichzeitig auftretende unbekannte Fehler unter verrauschten Bedingungen zu erkennen und zu kategorisieren. Zu diesem Zweck wurde ein GRU-basierter Entrauschungs-Autoencoder mit K-Means-basiertem Multilevel-Clustering entwickelt. Schließlich wurde ein HIL-basiertes virtuelles Testfahr-Framework für einen Echtzeit Validierungsprozess gemäß ISO 26262 entwickelt. Das entwickelte Framework dient als Fallstudie zur Validierung der Anwendbarkeit der vorgeschlagenen FDD-Modelle.

Acknowledgements

Firstly, it is important to acknowledge that the completion of this work is due to the grace of **Allah**. Without His benevolence, this endeavour would not have been achieved. As the Quran states, Allah has bestowed upon us knowledge that we did not previously possess, and His favour has been great upon us.

This research was conducted as part of the HIL Research Project at the University of Clausthal, ISSE. I am extremely grateful for the opportunity to conduct research in cutting-edge laboratories and with state-of-the-art industrial equipment, including the HIL system. I would like to express my sincerest gratitude to my primary supervisors, **Prof. Dr. Andreas Rausch** and **PD Dr. Christoph Knieke**, for their invaluable guidance, support, motivation, and direction throughout my research endeavors. I found our numerous, at times challenging, discussions to be highly beneficial and enlightening. They have played an important role in my personal and academic development and growth. They have selected me, provided me with support, and instilled in me a belief in my abilities, encouraging me to strive for excellence in my scientific endeavors. I would like to express my gratitude for all that you have done for me. It is a truly wonderful experience to engage in research under the guidance of both of you. I will cherish this time as my most precious memory. I am indebted to my co-supervisor, **Prof. Dr. Christian Siemers**, for his invaluable support, motivation, and guidance. He has played a pivotal role in enhancing the technical content and research publications by facilitating open dialogue and constructive feedback on new ideas.

I would also like to express my gratitude to my family, particularly my mother and father, and my brothers **Abdullah, Rasha, and Amer Abboush**, who have consistently provided me with encouragement and support, even during challenging periods. I would like to express my gratitude to my mother, **Amneh Joumah**, who has been a constant source of support throughout my academic career, from my first year of elementary school to the present. Her unwavering dedication to my education and her provision of material and moral support, as well as her prayers for my success in every critical situation, have had a profound impact on my academic life. I therefore dedicate this

work to her, along with my past and future successes.

I would like to express my profound gratitude to my wife, **Mariam**, for her unwavering support. Without her unwavering support, patience, and perseverance in the challenging circumstances of the diaspora, far from her loved ones, I would not have been able to complete this work. Mariam was consistently present and provided invaluable assistance throughout the entirety of this undertaking. She consistently demonstrated resilience in the face of my emotional fluctuations and provided guidance to maintain focus on the objective, even in the face of setbacks. She also assumed the role of primary caregiver and nurturer for our children, **Nesma and Rima**, the fruits of our love.

I would also like to express my gratitude to my second family (my wife's family), my uncle **Abdul Fattah**, and my uncle's wife **Khadija** and their sons and daughters for their support and prayers for success. In particular, I would like to acknowledge my cousin **Muna**, who provided invaluable assistance during the stages of publishing the results of the thesis chapters.

I am grateful to my colleagues and fellow students for their unwavering moral and material support. I hope that we may have the opportunity to collaborate again in the near future.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context and Motivation

Recently, the advent of sophisticated functions in contemporary Automotive Software Systems (ASSs), such as Advanced Driver Assistance System (ADAS), has played a pivotal role in transforming our mobility landscape. However, as software-driven systems have rapidly evolved, major testing and validation challenges have also emerged [5] [6]. This is due to the ever-expanding system architecture, with several hundred Electronic Control Units (ECUs) connected to the system via multiple communication buses [7]. Consequently, in such a complex system with many millions of lines of code [8], the probability of a failure occurrence is very high. Undetected software failures of the safety-related systems, e.g., braking system, entails catastrophic incidents and damage to the infrastructure and the environment due to the loss of vehicle control [9].

At the system integration testing level, a systematic testing is conducted to prove the correct functionality, and to detect the systematic faults caused by deviations in the specifications, e.g., design and implementation [10]. On the other hand, unpredictable faults that do not exhibit consistent or repeatable behavior indefinitely are called random faults [11]. For example, abnormal behavior of the sensor signals or unexpected actuator responses is considered a form of random faults leading to failure at system level [12]. It is noteworthy that these types of faults can cause an error and propagate from the faulty component through the subsystems to other components, resulting in a system-level failure as a termination of the intended functionality [13]. Various factors can cause random faults, e.g., memory corruption, intermittent communication faults, environmental factors, hardware failures and faulty sensors or actuators [14].

To achieve an acceptable level of dependability, the functional safety standard ISO 26262 was established, defining the requirements and recommendations for the development process [15]. In particular, to ensure the safety and reliability properties, several testing methods and techniques have been proposed by ISO 26262 to be conducted during the development process. For exam-

ple, at the system integration testing level, requirements-based test, interface test, Fault Injection (FI) test, resource usage test and back-to-back test have been recommended [4]. Meanwhile, to conduct the mentioned testing methods, several platforms have been introduced during the development process. Specifically, according to V-development model, Model-in-the-Loop (MIL) [16], Software-in-the-Loop (SIL) [17], Processor-in-the-Loop (PIL) [18], Hardware-in-the-Loop (HIL) [19], Vehicle-in-the-Loop (VIL) [20], and real test drive [21] have been introduced. Thanks to the mentioned platforms, the System Under Test (SUT) can be validated against the requirements during development lifecycle in different states, i.e., behavioral model, model code and executable code on the target machine [22]. The commonality between these platforms is that the SUT is connected to the controlled system (plant) in the form of a closed loop, forming the so-called X-in-the-loop [23].

According to the ISO 26262 structure (Part 4: product development at the system level), i.e., 4-8 system and item integration and testing, and 4-9 safety validation, tests derived from field experience and driving tests are performed. In these phases, the developed embedded software is validated in the actual target environment to ensure the functionality and safety of the developed product. At this level, the environmental conditions and the bus communication between the SUT (real ECUs) and other vehicle systems with real elements are considered in the validation process. By doing so, the systematic and non-systematic faults can be detected. In other words, driving scenario testing is conducted to ensure the correct integration of all ECUs in a vehicle with real physical systems under more realistic conditions [24]. The core idea of this approach is to conduct real driving scenarios by manufacturers using vehicles prototype on the public roads or within a virtual environment.

Considering the real-world usage and realistic environmental conditions, real test drive plays a vital role in evaluating the system attributes, i.e., safety, reliability, robustness, performance [25]. In such testing, the integration of developed ECU with other system components, e.g., other ECUs, real sensors and actuator and in-vehicle network is evaluated. To this end, during the test drives with several thousand of kilometers, nonlinear, multi-mode dynamic system behavior is captured. Specifically, by logging and monitoring system behavior, internal variables of ECUs are recorded as a multivariate time series data [26]. The recordings of the target system are then analysed to detect unexpected patterns or anomalies. However, besides the high cost, time consumption and the

uncontrolled environment with required expert knowledge, this approach has burdens of covering critical test situations without risks [27][28]. Recently, HIL-based virtual test drives have been emerged to overcome the limitations of the real test drives [29]. Thanks to HIL platform, being reliable, reproduceable and safe, many representative and relevant test kilometers can be conducted with low cost [30]. In the virtual driving environment, the SUT on the real target machine interacts with a simulated controlled system that employs realistic simulation. Besides the real steering system, i.e., steering wheel and pedals, real sensors and actuators can be connected to HIL simulator for integration testing. In some cases, a real subsystem, e.g., a real engine, is included in the loop with the SUT, known as system-in-the-loop tests [31].

In both aforementioned testing approaches, i.e., virtual and real test drives, to analyse the captured behavior resulting from driving scenarios, expert knowledge and system understanding is required [32][33]. In other words, beside checking ECUs' diagnostic trouble codes, the recorded data is inspected by the test engineer manually to detect unexpected faults. However, the more complex the ASSs are with heterogeneous components, the higher the cost, effort, and difficulty of manual analysis of the recordings [34]. The reason behind this fact is the vast amount of the data including several hundreds of sensor and actuator signals. Notably, in the real-world application, uncertain and noisy data samples is considered another complicating factor [35]. Besides, relying on a diagnostic trouble code to find unexpected or non-modelled sensor malfunction is infeasible [36]. Furthermore, based on the structure of the designed Test Cases (TCs) in HIL simulation method, any deviation of the measured values from the expected values results is considered as a failure of the test execution. The test results, including pass or fail, are documented in the test report without reasonable interpretation of the results, i.e., without determining the causes and natures of the occurring failures.

Therefore, an innovative solution capable of analysing the virtual/real test drive records efficiently based on historical dataset is required. Automated computer-aided recordings analysis of test execution can obviously contribute to effectively and accurately identifying and eliminating potential risks and vulnerabilities at an early stage of the system development process. Compared to manual inspections, consequently, the random unexpected faults at system level can be detected and identified with less cost, time, and effort. This in turn leads not only to an improvement in safety and reliability aspects, but also to a reduction in the cost and effort of the development

process.

1.2 Contributions

This dissertation contributes to the body of research by proposing a novel approach for failure analysis of the test drive recordings during the real-time validation of ASSs. Specifically, the main concern of the thesis is to address the problem of detection and diagnosis of the known and unknown critical sensor-related faults that could occur at the system integration phase of the V-cycle development model of ASSs. It investigates the applicability of Machine Learning (ML) and Deep Learning (DL) methods in developing an intelligent model to achieve the thesis objectives.

In other words, an intelligent fault detection and diagnosis (FDD) approach based on historical data and hybrid ML and DL methods is presented, focusing on the ASSs development process, i.e., system integration test. The scalability of the proposed work is not limited to a specific system, but covers the general HIL-based validation process. The focus of the proposed research is on the systematic and random unexpected faults that occur in the sensors. Some examples of the target fault types are gain, offset/bias, noise, hard-over, spike, stuck-at, packet loss, delay and drift faults. Based on historical data, the detection of single and simultaneous faults in the recordings of the test drive is addressed. Besides, identifying the type of the detected faults is the objective of the proposed work. Notably, not only known modelled faults but also unknown faults under noisy and unbalanced data conditions are covered. Compared to conventional approaches, the proposed method not only provides accurate detection and identification of random faults with high coverage, but also contributes to cutting down the cost and effort. More specifically, by employing the developed intelligent FDD, the inspection process of records by test engineers can be optimized by focusing on the relevant part of the records that contains the critical faults and avoiding irrelevant data. The main contributions of this study can be summarized as follows:

- **Real-Time Fault Injection Framework:**

  With the aim of identifying the critical sensor-related faults that lead to the violation of safety goals and the termination of functionality, a novel approach is developed to analyze the system behavior in the presence of faults under realistic operating conditions. For this purpose, a real-time FI framework is proposed, which is integrated with HIL system to inject the faults

during real-time simulation. Most of the sensor-related fault types are considered in the proposed framework to achieve a high degree of coverage. The corresponding system behavior of the identified critical faults is captured as multivariate time series data. Thus, a representative real-time dataset is generated and collected under fault-free and fault conditions, which in turn is used later to develop an intelligent FDD model.

- **Intelligent FDD based on Hybrid DL Methods:**

  An intelligent FDD model is developed using the representative dataset generated from the real-time FI framework. For this purpose, hybrid DL methods are used to design and develop the model structure, i.e., one dimensional-Convolutional Neural Network (1D-CNN) and Long Short-Term Memory (LSTM). The proposed DL model architecture overcomes the drawbacks of stand-alone methods and benefits from each other. The superiority of the proposed model in terms of accuracy is demonstrated using unseen test data from real-time virtual test drive and based on the evaluation metrics, i.e., precision, recall, F-score and accuracy. In addition to the healthy class, eight different fault classes are considered as the prediction output of the developed model.

- **GRU-based DAE for Detection and Clustering of Unknown Concurrent Faults:**

  In this study, a novel approach that considers the noisy conditions is proposed to solve the problem of detecting and clustering unknown faults in time series data. The proposed method achieves the mentioned goal by performing feature extraction and multi-stage clustering in two separate phases. A gated recurrent unit (GRU)-based denoising autoencoder (DAE) is developed for feature extraction from the noisy dataset. Then, the extracted denoised features are categorized on multiple levels using the K-means algorithm. In this way, not only critical known single faults but also unknown simultaneous faults can be detected and efficiently clustered. Based on unseen data from automatic and manual virtual test drives, the proposed model is verified and evaluated using two case studies, i.e., a complex gasoline engine and a dynamic vehicle system.

- **FDD of Simultaneous Faults using DL Methods under Noisy and Imbalanced Data:**

  For addressing the problem of simultaneous fault detection and identification, a novel method

is proposed in this study. Based on a representative dataset collected from a real-time FI framework, an ensemble learning model, i.e., LSTM+ Random Forest (RF), is developed considering the challenge of imbalance classes. In addition, a DAE model is employed to overcome the challenge of noisy data. In this work, ten different combinations of five fault types have been considered as the output of the model. The developed model is verified using test data from real-time virtual test drive. Furthermore, the superiority of the proposed method is demonstrated by comparing the evaluation results with state-of-the-art methods.

- **Real-Time Virtual Test Drive Framework:**

  This study presents a real-time virtual test drive framework for the validation of the aforementioned FDD models. The primary attributes of the proposed framework are its suitability for the analysis of system behavior under fault conditions during the virtual test drive and its capacity to collect a representative dataset. In addition to the automated driving mode, the user is permitted to engage in manual driving during real-time simulation through the use of a real steering wheel and pedals. Furthermore, a 3D visualization of the diverse test environments and scenarios has been developed, taking into account varying road network topology and environmental conditions.

1.3 Publications

- **Journal Articles Accepted and Published**

  - Abboush, M.; Bamal, D.; Knieke, C.; Rausch, A. Hardware-in-the-Loop-Based Real-Time Fault Injection Framework for Dynamic Behavior Analysis of Automotive Software Systems. Sensors 2022, vol. 22, p. 1360.

    DIO: https://doi.org/10.3390/s22041360

  - Abboush, M.; Knieke, C.; Rausch, A. Representative real-time dataset generation based on automated fault injection and hil simulation for ML-assisted validation of automotive software systems. Electronics 2024, vol. 13, p. 437.

    DOI: https://doi.org/10.3390/electronics13020437

- Abboush, M.; Bamal, D.; Knieke, C.; Rausch, A. Intelligent fault detection and classification based on hybrid deep learning methods for hardware-in-the-loop test of automotive software systems. Sensors 2022, vol. 22, p.4066.

  DOI: https://doi.org/10.3390/s22114066

- Abboush, M.; Knieke, C.; Rausch, A. GRU-Based Denoising Autoencoder for Detection and Clustering of Unknown Single and Concurrent Faults during System Integration Testing of Automotive Software Systems. Sensors 2023, vol. 23, p. 6606.

  DIO: https://doi.org/10.3390/s23146606

- Abboush, M.; Knieke, C.; Rausch, A. Intelligent Identification of Simultaneous Faults of Automotive Software Systems under Noisy and Imbalanced Data based on Ensemble LSTM and Random Forest. IEEE Access 2023, vol. 11, p. 140022–140040.

  DIO: 10.1109/ACCESS.2023.3340865

- Abboush, M.; Knieke, C.; Rausch, A Virtual Testing Framework for Real-Time Validation of Automotive Software Systems based on Hardware-in the-Loop and Fault Injection. Sensors 2024, vol. 24, p. 3733.

  DIO: https://doi.org/10.3390/s24123733

- Amyan, A; Abboush, M.; Knieke, C.; Rausch, A. Automating Fault Test Cases Generation and Execution for Automotive Safety Validation via NLP and HIL Simulation. Sensors 2024, vol. 24, p. 3145

  DIO: https://doi.org/10.3390/s24103145

- **Conference Papers**

  - Abboush, M.; Knieke, C.; Rausch, A. Representative Dataset Generation Framework for AI-based Failure Analysis during real-time Validation of Automotive Software Systems. In Proceedings of the 57th Hawaii International Conference on System Sciences (HICSS), Honolulu, HI, USA, 6–10 January 2024; pp. 7312–7322.

    URL: https://hdl.handle.net/10125/107263

  - Ailane TM, Abboush M, Knieke C, et al. (2021) Toward formalizing the emergent behavior in software engineering. In: 2021 IEEE/ACM Joint 9th International Workshop

on Software Engineering for Systems-of-Systems and 15th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (SESoS/WDES), pp 32–39. **URL:** https://doi.org/ 10.1109/SESoS-WDES52566.2021.00010

1.4 Structure

This dissertation is organized into eleven chapters. The main contributions of this study, including the experimental results, are summarized in chapters 6 to 10, while the introductory level is presented in chapters 1 to 4. The conclusion and future work are presented in chapter 11.

- Chapter 1 presents the context and motivation behind the research. Besides, the research objectives and the contributions are described in detail with list of publications. Finally, structure of the thesis is presented.

- Chapter 2 explains the theoretical background of the concepts and terminologies used in this study. It includes an overview of the concepts of FI and the scenario-based testing approach for ASSs. Furthermore, the existing methods and techniques of FDD for complex multivariate systems are presented. Besides, the main testing methods and platforms according to ISO 26262 are presented.

- In chapter 3 the problem statement is analyzed and the research questions to be addressed in the study are derived. Specifically, an experiment is described based on current state-of-the-art approaches for HIL-based real-time validation of ASSs. Subsequently the shortcomings and drawbacks of the recording analysis process are highlighted.

- In chapter 4 the related works and applications are discussed. In particular, the most important contributions and findings of the related work are presented. Besides, the limitations and differences compared to our study are pointed out. As a result, the novel aspects of the proposed approach towards solving the problem are emphasized and the research direction is defined.

- Chapter 5 provides an overview of the proposed approach as a solution concept, explaining the main phases of the methodology and highlighting the contributions to answer the defined

research questions.

- Chapter 6 presents and explains our innovative real-time FI framework. The developed framework enables the analysis and capture of the system behavior under faulty conditions considering real-time constraints. The main stages of development are outlined, and gasoline engine as a case study is used to evaluate the effectiveness of the framework. Finally, the experimental results is described, involving an analysis and discussion of the results obtained from the recorded dataset.

- Chapter 7 developed a hybrid DL-based approach for single FDD, in which the superiority of the combined methods compared to stand-alone techniques is demonstrated. The aim is to detect and classify single faults in the recordings of HIL tests. To this end, a DL CNN-LSTM model is developed using representative healthy and faulty data collected in Chapter 6. The effectiveness of the proposed model is evaluated using gasoline engine system as a case study, and the experimental results are analyzed.

- In chapter 8 the problem of detecting and clustering unknown concurrent faults is addressed. For this purpose, a GRU-based DAE is proposed so that comprehensive representative fault features can be extracted under noisy conditions. The extracted features are then clustered into specific groups using multi-level clustering algorithms. The proposed methodology is evaluated based on two cases study from automotive domain. Finally, the validation results are presented and discussed showing the superiority among other traditional methods.

- In chapter 9 a novel method for detecting and identifying the known simultaneous faults under imbalanced data is presented. In particular, an ensemble learning-based approach using LSTM and RF is developed. The detailed development and implementation phases are explained in this chapter. Besides, the validation results based on real-time simulation testing data is discussed and analyzed.

- Chapter 10 describes the development process of a virtual real-time test drive framework using a HIL simulation system. The proposed framework serves as an experimental prototype not only to generate the required data but also to validate the concepts developed in chapters 6, 7, 8 and 9. Using various test setups and driving conditions, the applicability and

effectiveness of the developed framework is validated.

- Finally, in Chapter 11, the conclusions of the contributions presented in the previous chapters are discussed. Besides, the potential improvements and future research direction are outlined.

2 Conceptual and Theoretical Background

This chapter provides an overview of the general background for the concepts and definitions used in this study. In particular, Section 2.1 introduces the system development lifecycle, highlighting the key phases of the V-model and the current standards for automotive systems development. In this chapter, besides presenting an overview of the AUTOSAR standard in Section 2.2, the current test methods according to the ISO 26262 standard as well as the testing process are outlined in Section 2.3. The main test platforms recommended by ISO 26262 for test execution are delineated in Section 2.4. In addition, Section 2.5 provides an overview of the fault taxonomy in automotive systems. Section 2.6 of this chapter presents the current approaches for developing fault detection and classification models. Finally, Section 2.7 introduces ML and DL techniques as a subset of the data-driven approach. As a result of this chapter, the underlying concepts, theories, and technological advances in the area of system validation, including test methods and platforms, fault detection, and classification, are reviewed in detail. This serves to provide a solid foundation for the subsequent chapters, ensuring a comprehensive grasp of the subject matter.

2.1 System Development Life Cycle

As a systematic and organized approach, the Software Development Life Cycle (SDLC) is used in software engineering to structure, plan and execute the activities associated with the development of an information system [37]. The SDLC is comprised of a number of phases, each of which involves distinct activities aimed at achieving specific goals related to the overall software quality and development process [38]. The core phases of the system life cycle include system concept, system design, implementation, testing, deployment, and maintenance. These phases are outlined in detail in the study by [39].

Definition 2.1 (System) *A system is a collection of interconnected objects that regularly interact or depend on each other and work together to achieve a specific goal* [40].

The system design phase defines the overall scope and goals of the target system. The main features and functions are also defined in this phase. In addition, various activities, e.g., feasibility study, cost analysis and risk management are performed and documented for stakeholder review. Once the design phase documents have been reviewed and approved, the requirements and expectations of the stakeholders are analyzed in the **requirements analysis** phase. This provides a clear and comprehensive understanding of what the system should do. This phase also identifies the detailed description of the system input, interface and process so that the functional requirements, non-functional requirements and constraints are established. Based on the results of this phase, the entire development process is carried out, including design, implementation, and testing. In the **system design** phase, the system is designed at a high level, including the specification of the architecture, components, modules and database design. In addition to specifying the software and hardware components, this phase also defines the intended system behavior and interface design. During the **implementation** phase, the high-level system design is transformed into functional systems by writing the actual software code according to the specifications. This phase also specifies the infrastructure required to execute the target functionality, including software, hardware, and communication systems. It should be noted that unit testing is also performed at this level to ensure that the functions are correctly executed according to the requirements. In the **test and integration** phase, several levels of testing are performed to verify that the system meets the requirements, starting with unit testing, through integration and system testing, to acceptance testing. The purpose of each level of testing is to verify and validate the SUT at each stage of the development process and to resolve any faults or problems that are found. Once the target system has been developed and tested, it is deployed to the target machine in the production environment during the **deployment** phase. During this phase, the performance of the system is monitored so that any issue identified during the initial deployment can be resolved and documented. After completion and delivery of the developed system, the **operation and maintenance** phase provides maintenance and support to resolve the problems and bugs, taking into account user feedback. In addition to implementing enhancements and updates, user issues are identified and documented for consideration in future development.

Definition 2.2 (Requirement) *A requirement is a statement that describes a necessary attribute, capability, feature, or quality of a system so that it provides value and benefit to a user*

[41].

Definition 2.3 (Testing). *Testing is the execution of a program with the goal of discovering the bugs and faults* [42].

Definition 2.4 (Component) *A software component is a composition unit with explicitly defined interfaces and context dependencies, as defined by a contract* [43].

It should be noted that the development strategy should be in line with the customer's requirements in the concept phase and the expected quality level, taking into account the cost and time trade-off. Depending on the project objectives and requirements, different approaches to systems engineering have been introduced. These include V-model, waterfall model, Scrum, DevOps, spiral model, and agile model, which guide the process of designing, developing, and managing complex systems [44].

In the development of automotive systems, safety requirements are considered as an additional aspect in the SDLC, in addition to the complexity of the system technology [45]. Furthermore, the coordination of the interaction between the Original Equipment Manufacturer (OEM) and the suppliers is usually part of the development process. In particular, the system design and architecture are described by the OEM. On the other hand, the product development is carried out by the suppliers according to the requirements of the OEM. The V-cycle development model is used as a software development and testing methodology in the automotive industry [7].

2.1.1 Model-Based Development

Model-based development (MBD) is a mathematical and visual approach that is employed in numerous domains of the automotive industry for the advancement of software systems. In particular, it is utilized in the design and implementation of intricate, safety-critical embedded systems [46]. In contrast to conventional software development patterns, verification and validation enables the identification of faults and anomalies at an early stage of the development process. Consequently, the role of testing is of paramount importance in reducing development time and costs, as well as the effort required [47]. Furthermore, there has been a notable increase in the interest in the automatic generation of vehicle software code from the model in recent years. Automatic code generation not only leads to an improvement in code quality and efficiency, but also to a reduction in the possibility of manual coding errors. In certain application domains, it is of paramount impor-

tance to assess the interaction between the SUT and its surrounding environment. MBD makes it possible to include the plant model in a closed loop with the SUT in the simulation, so that potential integration and communication faults can be identified and corrected at an early stage of system development. The core idea of the approach is to visualize the mathematical equations and algorithms using graphical models. The main features of MBD are executable specification, system-level simulation, modeling, virtual prototyping, and continuous testing and verification [48]. In particular, the MBD approach has been introduced based on the V-cycle development model.

Definition 2.5 (Simulation) *Simulation is the process of creating a model of a real system and conducting experiments with this model to understand the behavior of the system and evaluate different strategies for its operation* [49].

Definition 2.6 (Model) *To represent a collection of items or ideas in a form different from that of the entity itself* [49].

Definition 2.7 (Verification) *Verification is the process of determining whether a system adheres to the specified properties and requirements* [13].

MATLAB, developed by MathWorks, is one of the most widely used programming and development environments for system development, including numerical computation, data analysis, and visualization, in various industries [50]. As an extension to MATLAB, Simulink is a powerful tool for modeling and simulating dynamic multi-domain systems using the MBD approach and graphical block diagrams [51]. It allows to simulate and visualize the dynamic behavior of the system using an executable model. The key feature of Simulink is the use of block diagrams to visualize the system components, interfaces, and interactions between them. Both continuous and discrete systems, including functions, mathematical operations, and the system can be modeled. In addition to modeling and simulating the system architecture, the Simulink tool can automatically generate the model code. Besides, the system behavior can be analyzed under various normal and abnormal conditions, so that the verification and validation process can be carried out at an early stage of system development.

2.1.2 V-Cycle Development Model

V-model [52] is defined as a structured and systematic approach to a project where requirements are fixed and unlikely to change significantly during development. The visual representation of

the V-model describes the relationship between the development lifecycle and the corresponding test phase. It consists of two main parts that form the V-shape as a linear refinement process. The left side of the V-model represents the design phase and starts with requirements analysis, system design, and software design. On the right side of the V model, verification and validation activities take place at different levels, i.e., unit test, integration test, and system test [53]. Therefore, due to the shape mentioned above, each phase of the development lifecycle on the left side is linked to the corresponding test level on the right side, as shown in Figure 2.1. It is noteworthy that each color in the aforementioned figure represents a specific developmental phase. In the V-model, the system development process begins with the definition of requirements, from which the system specifications are derived and the target system functionalities are defined. In the second phase, system design, the system architecture is designed according to the derived specifications and user requirements documents. In this phase, the software interfaces of the software components, including inputs and outputs, are defined. The software code is written in the software design phase. In the case of MBD, the model code is generated automatically using simulation tools such as MATLAB/Simulink. Once the model code is generated, verification and validation activities take place on the right-hand side of the V-model at three levels, following a bottom-up approach. Each level is designed to ensure that the developed component conforms to the specifications. Unit testing is the first level where the software modules are tested against the requirements on the PC-environment. Notably, the corresponding hardware components of the SUT are also verified against the specifications using hardware mock-ups. The developed code is then deployed on the target machine where the integration test phase is performed. At this level, the environment of the SUT is stimulated to validate the system behavior, taking into account the interaction of the SUT with other systems and components. In the final phase on the right-hand side of the V model, i.e., system validation, all the real system components are integrated and tested against the system requirements. As mentioned earlier, the safety process in the development of safety-related automotive systems is performed through the phases of the V-model [54]. Specifically, safety analysis and safety validation activities are performed to identify the potential failures that lead to a possible hazard, and to address them by defining safety concepts [45]. In this way, the safety objectives and functional safety requirements of the products can be ensured together with the functionalities.

Definition 2.8 (Safety Goal) *A safety goal is a top-level safety requirement that defines the*

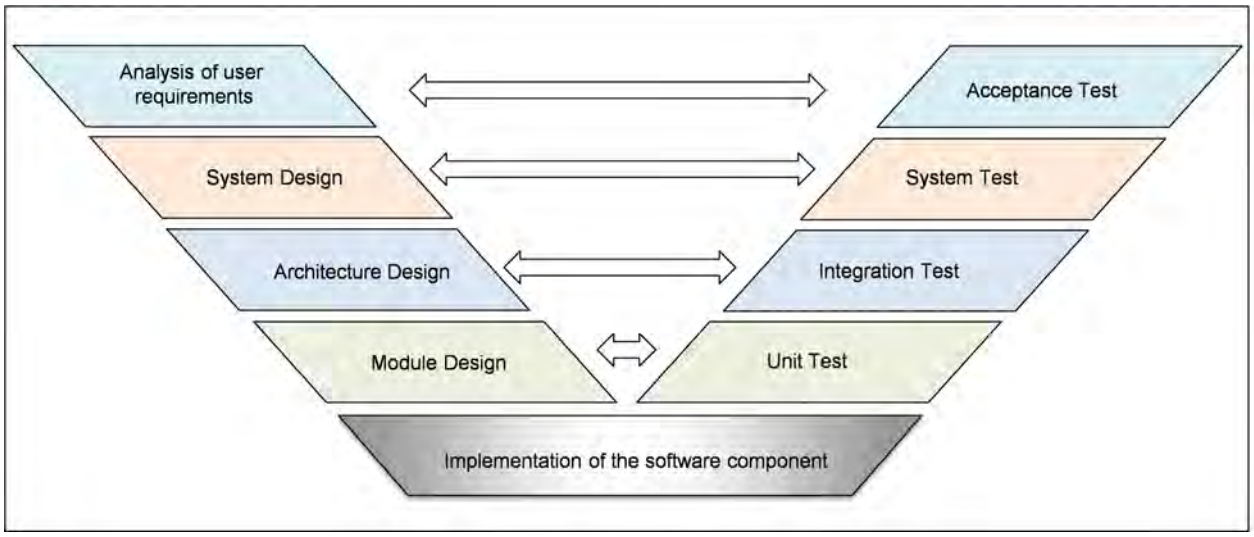


Figure 2.1: V-Cycle Development Model

functional objectives required to mitigate the associated hazards and risks. It does not deal with the technical implementation of the required safety measures [1].

Definition 2.9 (Functional Safety Requirement) *A functional safety requirement is the specification of an implementation-independent safety behavior or measure, including its safety-related attributes* [1].

Definition 2.10 (Functional Safety) *Functional safety is characterized by the fact that there is no unacceptable risk due to hazards caused by erroneous behavior of E/E systems* [1].

Definition 2.11 (Safety Validation) *Safety validation involves examinations and tests to confirm that the safety objectives are sufficient and have been achieved* [1].

Definition 2.12 (System Specification) *The specification is a set of requirements that defines the functions and attributes of an item or element* [1]. A system specification is a documented set of binding requirements for a system.

Definition 2.13 (Hazard) *A hazard is a system condition or event that represents a potential danger to objects and people* [1].

In complex systems, unforeseen sensor and actuator faults can potentially propagate through systems to various components and subsystems, which is known as fault propagation [13]. More specifically, the activation of the fault, by internal or external causes, leads to a change in the system state and results in a error. The occurrence of an error can cause the system to fail, resulting in the

cessation of intended functions. According to ISO 262626, the fault, error and failure have been defined as follows:

Definition 2.14 (Fault) *A fault is an abnormal condition that can lead to a reduction or loss of the ability of an element to perform a required function* [1].

Definition 2.15 (Error) *An error is a deviation between a calculated, observed or measured value or condition and the true, specified or theoretically correct value or condition* [1].

Definition 2.30 (Failure) *Failure is the cessation of the ability of an element or object to perform a function as required* [1].

In the field of multivariate time series data monitoring of dynamic system, the fault can be identified as:

Definition 2.16 (Fault) *A fault is an unauthorized, authorized, or permitted deviation of one or more system parameters, characteristics, behaviors, or patterns from the normal or standard state of the system* [55].

2.1.3 Automotive SPICE

Automotive SPICE (Software Process Improvement and Capability dEtermination) [56] has been introduced as a process evaluation standard in the automotive sector to assess and improve the quality, safety and efficiency of development processes [57]. The core of the standard is to address the challenges and requirements of developing software-intensive systems in the automotive industry by providing a set of standardized processes and practices. This in turn leads to improved traceability between work products. According to the standard, two models have been introduced to improve and evaluate software development and management processes, i.e., the Process Assessment Model (PAM) and the Process Reference Model (PRM) [58]. Based on the PAM range (0-5), several process capability levels are defined, where Level 0 indicates an incomplete process, Level 1 an implemented process, Level 2 a managed process, Level 3 an established process, Level 4 a predictable process, and Level 5 an optimized process. On the other hand, PAMs provide a reference guide for organizations where best practices in software development processes are achieved to ensure high traceability between work products.

2.1.4 Functional Safety Standard ISO 26262

ISO 26262 [59], Road vehicles - Functional safety standard, is derived from IEC 61508, which was published in the 1990s. It is intended for passenger cars up to 3.5 tons gross vehicle weight. Safety-related systems that include one or more electrical and/or electronic (E/E) systems are covered by this standard. It is also fundamentally designed for the automotive sector and adapted to the V-model lifecycle of the product development process.

The specifications for model-based development are included in the standard, along with guidelines for system design, hardware and software design and development, and integration of components to realize the complete product. Notably, suppliers are required to comply with ISO 26262 in a number of areas, including concept, design, development, validation, production, delivery and support. Validating the safety objectives and functional safety requirements according to ISO 26262 is called safety validation. On the other hand, ensuring the correctness of the functional requirements during development is called verification. According to the standard, the safety development process is carried out along all phases of the V model in which functional safety is addressed in a project. The first phase of the safety process is the safety concept, where the preliminary hazard analysis and the hazard analysis and risk assessment take place. The basic idea of the hazard analysis and risk assessment is to reduce the potential damage to an acceptable level. As shown in Figure 2.2, the process starts in the concept phase of ISO 26262 where the potential hazards are identified and the risk classes are defined. Safety objectives are then formulated to avoid or minimize these potential hazards. In addition, each safety goal is assigned to the ASIL level, which is determined by a systematic assessment of hazardous situations. Several factors are used to categorize the potential hazard, namely Severity (S - severity of the failure, danger to the user or the environment), Exposure Probability (E - probability of occurrence, i.e., frequency and/or duration of the operating condition) and Controllability (C - ability of the system to avoid the specified damage). If the components are hazardous and the safety requirements of ISO 26262 should be met, the combination of S3, E4 and C3 represents a very hazardous situation. On the other hand, a combination of S1, E1 and C1 means that the components are not hazardous and a QM level is required.

Definition 2.17 (Systematic Failure) *A systematic failure is a failure whose malfunction manifests itself in a deterministic manner that can only be prevented by the application of process or*

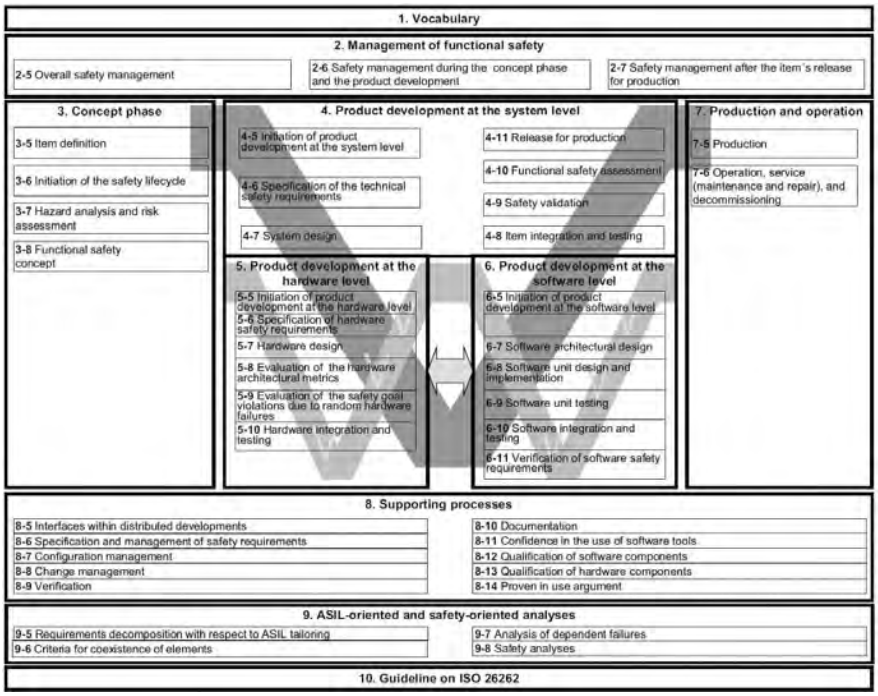


Figure 2.2: Structure of the functional safety standard ISO 26262 [1]

design measures [1].

Definition 2.18 (Safe State) *The safe state refers to the mode of operation of an object with an acceptable level of risk* [1].

Definition 2.19 (Safety) *Safety is the probability that a system will not fail in the event of a catastrophic failure* [1].

Definition 2.20 (FMEA) *(Failure Mode and Effects Analysis) is a method for assessing the severity of possible failure modes and providing information for risk reduction measures* [1].

Definition 2.21 (Safety Assessment) *A safety assessment involves independent experts who examine and evaluate the arguments in favor of the safety of systems* [60].

It is worth noting that the tool development process is not explicitly mentioned in the standard. Instead, the main requirements for tool qualification are defined.

2.2 AUTOSAR

Automotive Open System Architecture (AUTOSAR) [2] is a well-known automotive standard for standardizing software architecture during the development process. Using the AUTOSAR stan-

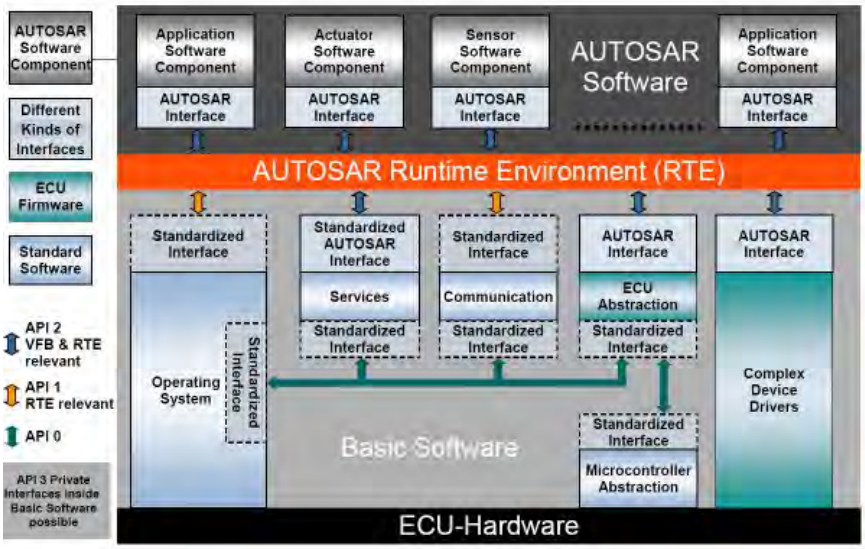


Figure 2.3: Overview of AUTOSAR Architecture [2]

dard to manage increasing software complexity not only improves software quality but also reduces development time. It also improves collaboration between OEMs and suppliers. Last but not least, it provides a basic software architecture and a metamodel for a standardized software architecture in the vehicle [61]. The AUTOSAR standard is structured on the basis of four different layers, i.e., the application layer, the runtime environment (RTE), the basic software layer and the ECU hardware layer [62], as shown in Figure 2.3. This architecture allows the application software to be developed independently of the hardware. By using the standardized automotive software architecture in the automotive industry, not only interoperability and scalability can be improved, but also the reusability of software components can be increased. The function of the basic software layer is to provide functionality for abstracting hardware-specific details so that software can be efficiently ported to different ECUs. The next layer, i.e., the RTE layer, implements the bus interfaces for each ECU. More specifically, the RTE layer plays an important role as a communication layer, facilitating the exchange of information between software components within the same or different ECUs. Finally, the application layer is responsible for the implementation of the ECU functionality by the software components (SWCs). Each SWC has an interface to communicate with other SWCs and the underlying layers, depending on the functionality it is to provide.

2.3 Test Methods According to ISO 26262

This section provides an overview of the test methods recommended by ISO 26262 for the system integration and testing phase of the V-model [59]. According to part 4 of the ISO 26262 structure, i.e., 4-7 integration and testing, different test methods are recommended to ensure the safety and functionality of the developed system. Some of the methods are requirements-based testing, back-to-back testing, communication and interaction testing, FI, and drive testing. Furthermore, the testing process used to perform the testing method during the system validation and testing phase is introduced.

2.3.1 Requirements-based Test

Requirements-based testing is defined in the context of ISO 26262-4-7 (Table 6) as testing against the functional requirements. In other words, requirements-based testing verifies that the functionality of the SUT has been correctly implemented in accordance with the specifications. Understanding and analyzing the requirements of the software or system is therefore an important step in the implementation of the method. The **test plan** is the first step in implementing the test method and includes defining the test strategy, scope, resources, schedule and evaluation criteria. To achieve the test objectives with full coverage of the functional requirements, three different phases are then carried out, i.e., the test generation phase, the test execution phase and the test evaluation phase, as shown in Figure 2.4.

In the **test case generation** phase, the test specifications are derived from the requirements and documented. Test specifications include the identification of the test object, the environmental conditions and prerequisites, the specific TCs, the test data and the expected results for the test evaluation. In the second phase, the TCs are implemented and executed, either manually or using automation tools. Requirements management tools such as DOORS are used to ensure continuity and traceability between requirements, faults and TCs. In the second phase, **test case execution**, all identified TCs and their test data are automatically/manually executed in the test environment. The actual results of the TCs execution are then passed on to the third phase. Finally, in the **evaluation phase**, the actual measured behavior of the SUT is compared with the theoretically expected behavior. In particular, the output values of the executed TCs are compared with the

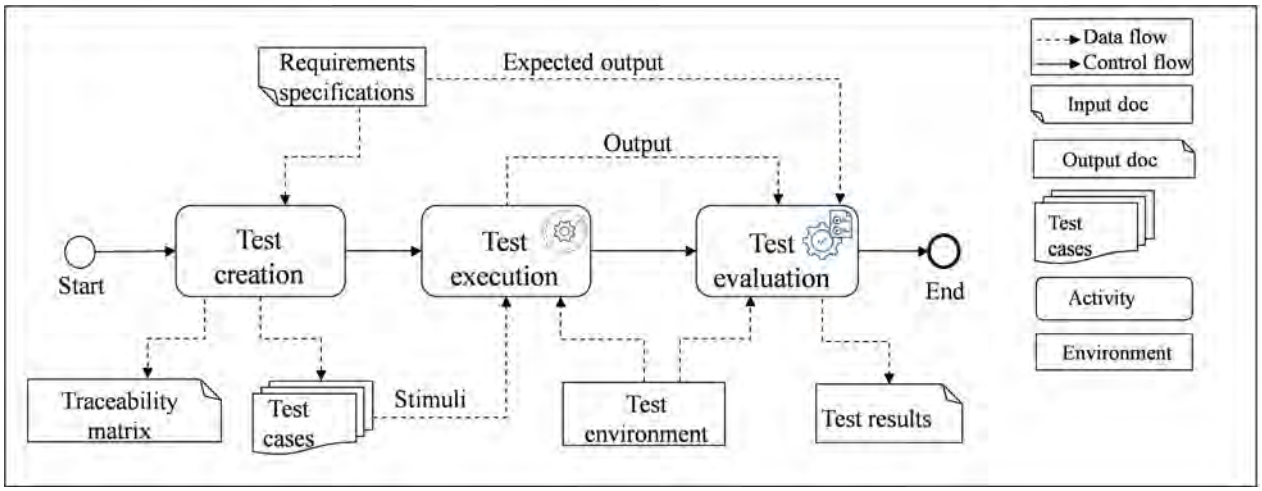


Figure 2.4: Requirements-based Test Method

expected values. Finally, a test report with the evaluation results (pass or fail) is automatically generated. The traceability between TCs and requirements in this method ensures a high degree of test coverage, which in turn promotes confidence in the quality of the software.

2.3.2 Back-to-Back Test

The Back-to-Back test method [63] is introduced as a complementary test method for requirements-based testing and is strongly recommended by ISO 26262. The basic idea of this test method, according to ISO 26262 parts 4 and 6 (Tables 5 and 6), is to compare the response of the test object and the test response of a simulation model to the same stimuli in order to detect differences or faults. This ensures that the behavior of the model-level test object and the generated object code are equivalent. Back-to-Back testing can be used at different stages of the development cycle, e.g., at the model level and at the implementation level. To illustrate the process of the method, an overview of the concept is given in Figure 2.5. It starts with the generation of TCs from the requirements. Then, All TCs, including the input values (stimulus), are applied to the system model and the software application loaded onto the target hardware. Finally, a comparison is made between the response of the model and the test object. It is taken into account that the same stimuli should be applied at both levels in order to detect unexpected faults. It is worth noting that Back-to-Back testing has been proposed not only for system integration testing, but also for software unit testing and software integration testing.

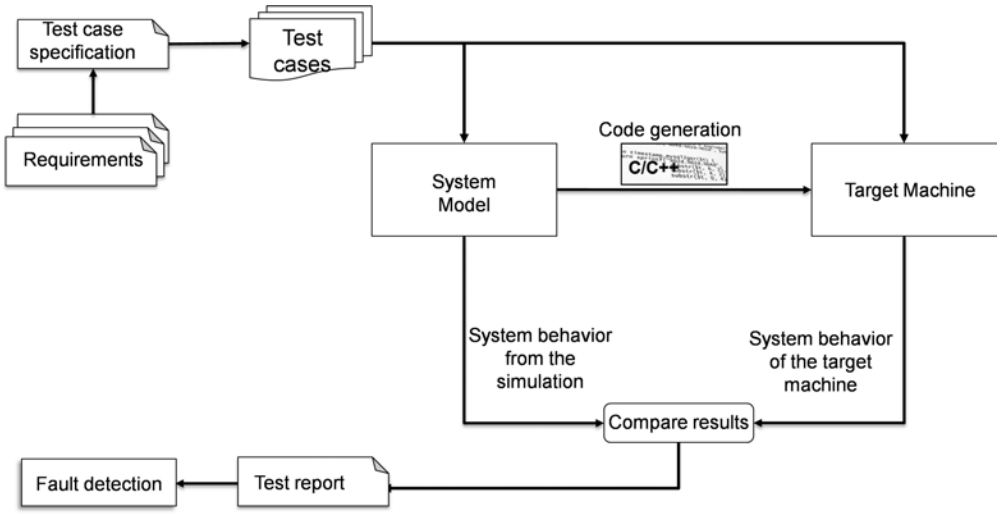


Figure 2.5: Back-to-Back Test Method

2.3.3 Communication and Interaction Test

Communication and interface testing [64] is defined in ISO 26262-4-7 (Tables 5, 6 and 7) as a test method for verifying system interfaces, including analog and digital inputs and outputs. It can be used to ensure that communication between system components, ECUs and subsystems is performed correctly to achieve the intended functionality. It is also used to test compatibility, timing and other specified characteristics of the system. In this way, the correct communication between the tested components, subsystems or systems is verified, focusing on the test interfaces regardless of the implementation. In automotive systems, several communication protocols are considered during interface testing. For example, CAN communication tests, LIN tests, FlexRay tests and Ethernet tests are the main aspects of this method. In addition, this method tests the diagnostic communication protocol, such as OBD-II, to ensure the ability to retrieve and interpret diagnostic fault codes. Finally, it tests the communication between the sensors/actuators and the corresponding ECUs to ensure the accuracy of sensor data transmission and the response time and behavior of the actuators.

2.3.4 Fault Injection Test

Fault injection (FI) [3] is a technique for assessing system reliability in which faults are intentionally introduced into a system through controlled experiments in order to analyze the behavior of the

system in the presence of faults [65]. The approach enables an effective assessment of various dependability attributes, including safety, reliability, availability, and protection against random faults [66, 67]. The faults considered in this approach are predominantly non-linear and unpredictable faults, also known as random hardware faults (HW faults). These faults are mainly caused by environmental factors and occur as an impairment of component functionality. An example of a random hardware fault is a delay or the loss of data in the event of a communication faults. In contrast, systematic faults are deterministic and are caused by design-related factors such as deviations in the device specifications.

Definition 2.22 (Reliability) *Reliability refers to malfunctions of systems that are due to components not functioning according to their specification at the time of design* [68].

In accordance with ISO 26262-4 (Table 9), FI can be employed to enhance the scope of testing for safety requirements by implementing specific methodologies to introduce defects into the object, element, component, or communication between them. FI represents a highly effective technique utilized during the development of ASSs to verify and validate safety and reliability aspects. The technique is predicated on the premise of injecting faults into systems with the objective of analyzing and evaluating the system's response to abnormal conditions.

The evaluation of fault-tolerant systems is often carried out with the aid of FI methods, whereby the robustness of the fault tolerance is assessed in the presence of faults. In this regard, reference is made to the literature [69, 70]. In contrast to other analysis methods, such as FMEA, this method is distinguished by the injection of faults into the system and the subsequent observation and analysis of the system's dynamic behavior through experimental means. A typical FI environment comprises five primary components: the controller, fault injector, system or concept under test, load generator, and monitor [3], as illustrated in Figure 2.6. The definition of the system inputs and faults to be injected into the target system is either based on expert knowledge or utilizes probability distributions. The key attributes of FI are a set of faults (F) with which the injection is to be performed, activations (A) practiced at the time of the experiment, a set of readouts (R) representing the output range, and derived metrics (M) resulting from the application of fault TCs. Collectively, these form the FARM environment model [69]. The implementation of the FI method necessitates the definition of three principal dimensions of configuration faults: type, location, and time of injection. These three dimensions collectively define the fault space.

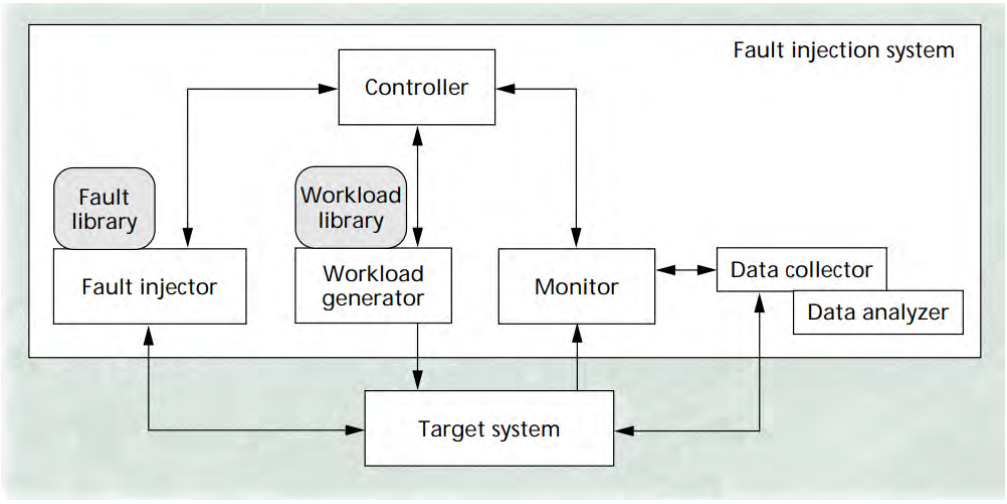


Figure 2.6: Fault Injection Test Method [3]

A substantial number of FI strategies have been proposed in the literature. Depending on the underlying method, these can be divided into five categories: hardware-based, software-based, simulation-based, emulation-based, and hybrid FI. A detailed description of the respective advantages and disadvantages of the individual techniques can be found in [65]. Subsequently, a variety of techniques for implementing the individual methods were proposed. The most significant techniques for implementing software-based FI include code change injection, data FI, and interface FI [71]. Additionally, several tools based on these techniques have been proposed in the literature. These include XCEPTION [72], FIAT [73], FTAPE [74], Orchestra [75] and FERRARI [76]. In contrast, simulator commands, saboteurs and mutant-based techniques are employed to perform simulation-based FI [77]. In addition, the following tools are employed along with simulation-based FI: VERIFY [78], MEFISTO [79], and SST [80]

Definition 2.23 (Element) *Elements refer to sub-units of items, including system, subsystem and component, as defined in ISO standards* [1].

Definition 2.24 (Environment Model) *An environment model is a representation of the real world in a simulation. The model includes the scenery of static objects, such as roads and signs, and dynamic objects, such as vehicles and pedestrians, including their behavior* [81].

Definition 2.25 (Fault Activation) *Fault activation is the application of an input (the activation pattern) to a component that activates a dormant fault* [13].

2.3.5 Driving Test

To meet the requirements of the development process defined in ISO 26262, the test drive [82] is performed at the system integration level to validate the safety and reliability of the target systems under real conditions. This is done through real test drives on public roads under realistic conditions and in different driving scenarios. In this way, the test engineers evaluate the system behavior under different road conditions, traffic situations and environmental factors [83]. In the system integration phase of the ISO 26262 structure, i.e., 4-8 and 4-9, validation activities are performed to ensure the correct integration of all ECUs in a vehicle with real physical systems under more realistic conditions. The steps for performing the test method are defined according to the test objectives.

The main phases in the execution of the test method are test planning, test scenarios development and criteria definition, test scenario execution, data acquisition, data analysis, test evaluation and reporting. In the **planning** phase, the test object is defined, i.e., the evaluation of safety, functionality or vehicle performance. In the next phase, the **test scenarios** are determined and created based on the defined test objects. In addition to defining the evaluation criteria for evaluating the test results, the driving conditions, routes and maneuvers are selected in this phase. In the **test execution** phase, the test drives are performed based on the defined scenarios and driving conditions, taking into account different road types, weather conditions, traffic situations, and critical situations. In the **data acquisition** phase, the vehicle parameters, i.e., the ECU variables and sensor signals, are recorded by the vehicle data loggers during the test drives. The recorded datasets serve as the basis for analysis and validation. The observations during the test drives are documented, including abnormal vehicle behavior states. Expert knowledge is then used to **analyze** the recorded data to identify the problem that caused the vehicle to behave abnormally. In addition to checking ECU diagnostic trouble codes, the test drive data is analyzed by the test engineers to identify known and unknown faults. Finally, the results and diagnosis of the faults found are documented in a comprehensive **report**. In addition, recommendations are provided to the development team for further development or refinement.

Definition 2.26 (Test Scenario) *A test scenario is defined as a particular route taken through a application, i.e. a specified progression of actions* [84].

Despite its important role in ensuring safety functions with highly valid test results, it has its

limitations in terms of cost, risk, time and knowledge. In particular, it is not feasible to perform several thousand representative test kilometers covering all critical situations. In addition to the high cost of testing, the tester's confidence in the safety-critical scenarios is low. Furthermore, it is impossible to verify safety and robustness under faulty conditions by FI while driving in an uncontrolled environment.

Recently, advances in virtual platforms have paved the way to overcome the above obstacles. In various engineering applications, virtual testing has demonstrated the ability to validate the SUT under safe, reproducible, and controllable conditions. This not only reduces costs, but also increases the coverage of safety-critical scenarios under faulty and fault-free conditions [85, 86]. The core idea of the virtual-based test drive is to replace the real driving environment with a virtual controlled environment [87]. It should be noted that the virtual test is characterized by different states of the SUT. In the case of non-real-time simulation, an executable model of the real ECU, i.e. a virtual ECU, connected to a model of the controlled system can be executed on the host PC, which is called model-in-the-loop (MIL). In real-time HIL simulation, on the other hand, the application software on a real ECU interacts with the simulated vehicle system, taking into account real-time constraints [88].

2.3.6 Resource Utilization Test

The resource utilization test method [59] has been recommended by ISO 26262-4-7 (Tables 5, 6 and 7). The main objective of this test method is to ensure that hardware resources such as CPU, memory and storage, real-time operating system are used efficiently under different operating conditions. Power consumption and bus load are some examples of issues that are tested when performing this method in dynamic element-level environments. First, due to the large number of ECUs, additional network management and power consumption requirements should be defined for each ECU. This means that the power consumption of each individual ECU should be evaluated. On the other hand, the bus loads are determined especially during the integration test before the real ECUs are available. Since all functions of the networked ECUs should work correctly within their specifications, a bus load test should be performed to ensure that a high bus load does not have a negative effect on the functions of the networked ECUs. Delays in message transmission or erroneous messages are some of the causes of additional bus load.

2.3.7 Test Process

The test process consists of a number of phases, i.e., test planning and control, test analysis and design, test implementation and execution, test evaluation, and finally test closure activities [89].

Test planning takes place at the beginning of a software development project, which serves as a guideline for the test activities and is documented in the form of a test plan. Notably, the responsibilities of the testers in performing the tasks are defined in the test plan. This plan not only estimates the time required, but also defines the resources, equipment, and tools needed. The test activities are monitored by test control to identify deviations between the actual status and the test plan. The main objective of test planning is to define the test strategy.

In the second phase, i.e., **test analysis**, the specifications of the test object are reviewed. TCs can then be created either from the specifications documents or from the results of the risk analysis. This phase defines also the prerequisites and requirements for TCs creation, as well as the structure of the test object. The TCs are developed in the test specifications and designed using the test techniques. However, if the documents are imprecise in defining the expected output or the expected behavior of the system, the TCs cannot be created efficiently. In this case, it is stated that the testability of the functional requirements is not sufficient. In addition to the preconditions and environmental conditions, the expected output or behavior is defined from the specification of the test object, the so-called oracle, before the test is executed. The developed TCs can be divided into two categories, i.e., negative tests and exception handling. Negative TCs are concerned with checking the specified behavior, output and response. On the other hand, in the case of exception handling, the developed TCs aim at analyzing the reaction of the test objects to invalid and unexpected inputs or conditions.

Definition 2.27 (Test Case) *A test case represents the parametrization of a test scenario in which all the necessary parameters for the simulation are defined* [81].

The **test strategy** is defined in the test plan to define the test framework and the predefined test methods. Traceability between the TCs and the corresponding requirements should also be established at this stage, and in some cases the risk analysis specification is used to establish traceability between the potential failure mode and the corresponding TCs.

The third phase is **test implementation and execution**, in which the TCs developed in the previous phase are executed and the results are recorded. In addition to verifying the traceability

between the requirements and the TCs, the test execution steps are described in this phase. In particular, the priority of TCs execution is taken into account. Each group of TCs is executed separately as test suites or test scenarios. Once the test harness is checked as part of the test environment, the test can be executed manually or with tools according to the predefined scenarios. Finally, the test results, including pass or fail results, are documented along with the test environment and input data. Any deviation between the expected output and the actual output is checked and documented as a failure. The possible causes of the failure are then identified. There are several causes of test failure, such as an incorrect test specification, problems with the test infrastructure or TCs, or incorrect test execution. TCs coverage should also be ensured and measured. As part of the qualification process, the severity of potential faults found during execution should be determined and used to prioritize fault resolution. TCs prioritization is useful when there is not enough time to run all TCs. The cases are then selected according to their priority and executed to find as many critical faults as possible. This type of testing is called risk-based testing. Other approaches to testing include model-based testing, statistical or stochastic testing, standards-based approaches, reuse-based approaches, checklist-based (methodological) approaches, and expert-based approaches.

The purpose of the **test evaluation and reporting** phase is to evaluate the test item against the test completion criteria defined in the test plan. To do this, an appropriate metric should be defined to determine whether or not the test completion criteria have been met. If the test completion criterion is not met, the TCs should continue to be executed. In addition to test coverage, the failure rate is another criterion for deciding whether or not to stop the test. At the end of this phase, a summary test report should be produced.

The final phase of the testing process is **test completion**. Three points in time are defined in this phase, i.e., the time of software release, the time of test completion, and the time of milestone or maintenance release. The test suits, including TCs, test logs, test infrastructure and tools, should be available for future use in the event of a program change.

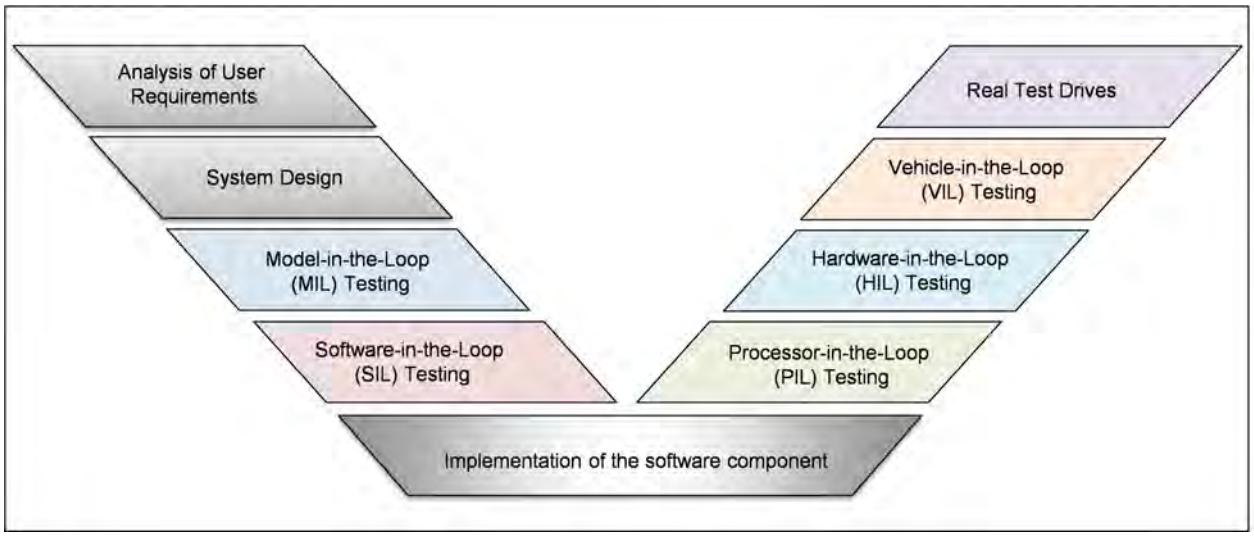


Figure 2.7: Testing platforms according to V-Model

2.4 X-in-the-Loop Platforms for Automotive Software System Validation

In order to guarantee the quality of the product and to meet the requirements of the functional safety standard ISO 26262 [15], comprehensive testing activities are necessary to ensure the safety and reliability of the product. In order to achieve this, several test platforms, known as X-in-the-loop testing [23] , have been defined in accordance with the V-model development approach. Some vital in-the-loop platforms are MIL, SIL, PIL, HIL, VIL, and real test drives [22, 90], as illustrated in Figure 2.7, each color represents a distinct test phase.

2.4.1 Model-in-the-Loop

Model-in-the-Loop (MIL) testing [91] is appropriate for a simulation environment that is entirely developed and executed by computer software. MIL takes place on the left side of the V-model, i.e., in the design phase, where the mathematical model of the SUT and its plant models are integrated and tested in a closed-loop simulated environment. This allows early verification and validation of the system without the need for physical hardware. The main phases of MIL test execution are model development, model integration, simulation scenarios, test execution, and test evaluation. The control system and controlled plant are modeled and simulated in the modeling environment.

This allows functionality to be tested and fundamental problems in the control logic to be identified without the need for physical hardware components. Therefore, no interaction or communication with physical devices or cables is required for MIL testing. In the case of automotive systems, the SUT typically represents the developed control algorithm (ECU), while the plant model represents the entire vehicle system, e.g., vehicle dynamics, powertrain, sensors, actuators, driver, and environment. Once the models are integrated, the test scenarios are defined to perform the test under different conditions, including normal operation, limit cases, and abnormal conditions. To validate robustness and reduce potential risks, the system behavior under abnormal conditions can be analyzed by FI.

The functional behavior of the system is also verified by simulating the control and system model in a simulation environment. Corresponding MIL test results are used as a reference for the next test phase in the V-shaped model. MATLAB/Simulink and SCADE are powerful tools for developing systems using a model-based design approach. They also support system design, simulation, automatic code generation, and continuous testing and verification of embedded systems. As a result, MATLAB/Simulink is the most widely used tool for performing MIL tests and enables reliable MIL simulation of ASSs.

2.4.2 Software-in-the-Loop

Software-in-the-Loop (SIL) testing [92] is the next step after MIL in the design phase of the V-model. SIL is responsible for testing the system's object code, which is either written by hand or automatically generated by MIL tools. Once the model has been verified, the code is generated from the model by the simulation tool and tested in a virtual environment, i.e., on a host PC. At this stage, the generated control code is also tested without hardware elements to verify how the software interacts with the simulated system based on various inputs, functions, and mathematical algorithms. Multiple versions of control algorithms can be validated without restrictions in the system environment, and many test runs can be performed. In addition, software faults can not only be detected and corrected at an early stage, but also the propagation of software faults to later stages of development can be prevented. In principle, automatic generation tools are required to generate and compile the control model code for implementing the SIL platform. MATLAB/Simulink is a well-known tool that enables automatic code generation from the model on the host PC. It is

important to note that both the control system and the plant model are at the same level. Although SIL checks the correctness of the control system object code before it is transferred to hardware, the real-time characteristics of the control system and the controlled equipment are not taken into account at this stage.

By reducing the cost and increasing the speed of the test process, SIL is superior to other methods for early validation of the control strategy developed without the need for real hardware or a real-time environment. However, because the control code and the plant are executed at the same simulation level, i.e., on the host PC, real-time conditions and physical communication modeling are not taken into account. In addition, the conditions of the real environment and the target machine differ from those of the simulation environment, which in turn leads to deviations in accuracy.

2.4.3 Processor-in-the-Loop

Processor-in-the-Loop (PIL) testing [88] represents the initial stage on the right side of the V-model. It employs a simulation environment and hardware devices. It is regarded as an efficacious simulation methodology that occupies a position intermediate between SIL and HIL. The code of the ECU from the SIL is implemented on the target processor, which interacts with the plant model at the simulation level. In other words, the use of PIL enables the validation of the behavior of the executable object code on the real processor within the target hardware. Conversely, it assesses the interplay between the real processor and the simulated system, encompassing simulated sensors, actuators, or communication interfaces. PIL enables the detection of faults associated with the execution of software on the real hardware, as well as the evaluation of the impact of the processor architecture on the software. Furthermore, the timing behavior of the software on the target machine can be analyzed to verify that the system behavior meets real-time requirements within the specified timing constraints. Additionally, the software is tested under the constraints of the simulated environment. PIL is particularly useful in situations where a HIL setup is not feasible. Furthermore, the implementation of PIL facilitates the early validation of embedded systems, thereby reducing the overall cost of a complete HIL system.

PIL enables the verification of the behavior of the ECU in a real hardware environment. As the simulation environment is employed as an offline simulator in the test sequence, the real-time behavior of the plant cannot be tested in PIL. In other words, despite the advantages in validating

real-time control, the execution of the controlled system is constrained to the non-real-time platform on the host PC.

2.4.4 Hardware-in-the-Loop

The limitations of PIL are overcome in HIL tests [93] by testing the behavior of the control (SUT) and controlled system in real-time. The control object is implemented in the real target processor, e.g., in the ECU. On the other hand, the code generated from the plant model is also executed in the HIL simulator, so that the real ECUs can communicate with the system in real-time in a closed loop. In HIL simulation, therefore, both the ECU and the plant model are implemented in real hardware, allowing verification and validation of the target system under real-time conditions. In some cases, real hardware elements can be connected to the HIL simulator, e.g., real sensors or actuators of the vehicle system. In this way, the interaction between the embedded software of the SUT and the actual hardware components of the system can be validated in a realistic environment. The HIL simulation platform enables the execution of various test methods according to ISO 26262, e.g., requirements-based tests, FI tests, virtual test drives, performance tests and back-to-back tests. Repetitive and systematic testing of software and hardware changes, known as regression testing, can be performed efficiently with HIL.

Another advantage of HIL is that the vehicle system can be validated without a real vehicle prototype, as the missing components can be modeled and simulated. Consequently, it is frequently employed as a substitute for a real test drive, which is costly for the tester and potentially hazardous for the driver. The HIL platform is strongly recommended by the ISO 26262 standard for software safety testing, part 6-Table 21.

Recently, the characteristics of HIL have led to a plethora of contributions in both academic research and the industrial field. Consequently, the investigation of the utilisation of HIL in diverse engineering challenges and application domains is currently a highly pertinent subject of research, including robotics, electric drives, power grids, railways, and automotive. The research focus determines the three modeling methods that form the HIL system: HIL simulation at the signal level, HIL simulation at the power level, and HIL simulation at the mechanical level.

There are several suppliers of HIL systems. dSPACE [94] and Speedgoat [95] are the best-known companies in this field. Note that the real ECU can be replaced by rapid control prototyping,

e.g., by the MicroAutoBox II, which acts as a real ECU and emulates its functionality.

The MicroAutoBox II is equipped with the capability to communicate with the HIL system via the CAN bus. All signals from the sensors and actuators can be transmitted in real-time via this bus. Conversely, the entire system model of the vehicle, including vehicle dynamics, the engine, sensors and actuators, the powertrain, the driver model and the environment, is modelled and simulated at the simulation level. Subsequently, the code is automatically generated and utilized in the HIL simulator. Finally, the software application is executed with the defined driving test scenario, allowing for the analysis, verification, and validation of the control system connected to the planned system in real-time. In order to achieve this, dSPACE provides a range of software tools with a user-friendly interface, including ModelDesk for system/component design and parameterization, ConfigurationDesk for code generation with system configuration, and ControlDesk for measurement, calibration, and diagnostics.

2.4.5 Vehicle-in-the-Loop

The Vehicle-in-the-Loop (VIL) testing method [96] is regarded as an advanced and comprehensive approach to testing automotive systems integration. The system employs a complete vehicle prototype with real physical systems and components connected via an on-board network. Driving scenarios are conducted in either the real world or a virtual environment in accordance with the VIL methodology. Consequently, it is employed at a subsequent stage of the V-model with the objective of attaining a high degree of realism in the vehicle's behavior under diverse conditions. The VIL test method can be employed to assess not only the safety and functionality of the entire vehicle, but also the performance and integration of embedded systems. By considering the interaction between ECUs and the physical hardware components of the vehicle, it is possible to evaluate control logic, sensor fusion, and actuator response in a realistic environment. In accordance with ISO 26262, VIL is recommended for use in the vehicle integration test, Part 4-7 (Table 7), for long-term testing as a user test under real conditions.

During the VIL test, the system is installed in a real vehicle in order to verify its functionality. Furthermore, it ensures that all physical components function in unison in a manner that is consistent with their intended purpose. Consequently, the VIL represents the final stage in the validation process, prior to the system's release to the general public. Typically, the VIL platform is em-

ployed to validate the target system against functional safety requirements, with due consideration of driver behavior. To circumvent the potential risks associated with a real-world test drive on public roads, a virtual simulation environment is employed, wherein dynamic traffic with pedestrians can be modelled and displayed in a three-dimensional visualization. This enables the simulation of realistic driving scenarios, including acceleration, braking, turning, and the impact of varying road conditions. Furthermore, the VIL method incorporates traffic simulation to account for the interaction of the vehicle with other vehicles. Furthermore, the VIL method considers a multitude of environmental variables to guarantee the vehicle's functionality under diverse circumstances. The vehicle's response to changes in adverse environmental conditions, such as precipitation, snow, and atmospheric pollution, is validated through the use of weather simulations. However, the greater the complexity of the VIL platform preparation, the higher the cost and time required, particularly for complex detection systems. Some technical applications for the use of VIL include the validation of autonomous driving systems and ADAS. In such cases, the integration and interaction between system elements and ECUs is of critical importance for the system's decision-making capabilities in complex driving scenarios.

2.5 Fault Taxonomy in Automotive Systems

A review of current literature reveals that faults can be classified into three main categories, i.e., hardware faults [97], software faults [98], and network and communication faults [99]. Each of these categories can be further divided into subcategories. faults are grouped according to the type of component that contains the fault, which can be a sensor, an actuator, a plant or a control system. In this context, the faults are referred to as hardware faults. In contrast, bit flips faults, runtime faults, corruption of processor registers, and target restarts are classified as types of software faults. Communication-related faults, which include delays, data loss, and improper delivery, are abnormal states of communication protocols that occur on the network path between components, applications, or subsystems. In addition, there are further fault categories that can be classified as transient, permanent, or incipient, depending on the duration and type of occurrence. For further details [100]. In the context of signal-based system behavior, i.e., time series data, a number of distinct fault types have been identified that manifest themselves in the form of measurement

deviations, stuck-at values, or scaling from the true values [101, 102, 103, 104]. Sensor and communication faults encompass a range of issues, including gain, delay, offset, hard-over, stuck-at, spike, noise, packet loss and drift. An overview of these faults and other fault types is presented in Figures 2.8.

It is important to note that a number of factors can contribute to the occurrence of the aforementioned sensor and communication faults. Based on the sources [105, 106], the direct causes of sensor failures in the automotive sector include dirt or damage to the sensors, aging, corrosion, vibrations, electromagnetic interference, improper calibration, and weak batteries.

A gain fault occurs when the sensor's output signal is not properly scaled and deviates from the expected correlation, resulting in the output signal being amplified or attenuated incorrectly. This type of fault results in the generation of inaccurate measurements, with the output signal exhibiting either a high or low value. Similarly, an offset fault is caused by the distortion or deviation of the sensor output due to a constant fault, which in turn leads to systematic faults in the sensor measurements. One potential cause of this type of fault is an internal bias voltage or bias current in the sensor. In the case of a delay fault, the response time of the sensor is delayed or has a latency compared to the expected time. In safety-related applications, this fault is considered critical due to its negative impact on synchronization and timing in control systems. When the sensor reading remains at a fixed value rather than the dynamic output, this is known as a "stuck-at" fault, which results in the loss of function and the inability of the system to respond to dynamic conditions. The occurrence of a stuck-at or freeze-at output is attributed to a mechanical obstruction within the internal components of the sensor. In contrast, a hard-over fault is initiated when the sensor output surpasses its maximum threshold and remains fixed in an extreme position. In the absence of a delay fault, the control systems exhibit extreme reactions to changing conditions. A spike fault is defined as a sudden and significant increase (or decrease) in sensor readings. One of the causes of false alarms in the system is the spike fault, which is defined as an irregularity in the sensor output. Random fluctuations or faults in the measurements are referred to as noise faults and are also known as erratic or precision faults. The primary causes of this type of fault are electrical noise, environmental conditions, or limitations in sensor technology. Noise faults can result in a reduction in measurement accuracy and the inability to obtain useful information from the measurements. The occurrence of drift faults can result in the loss of calibration and accuracy, thereby reducing

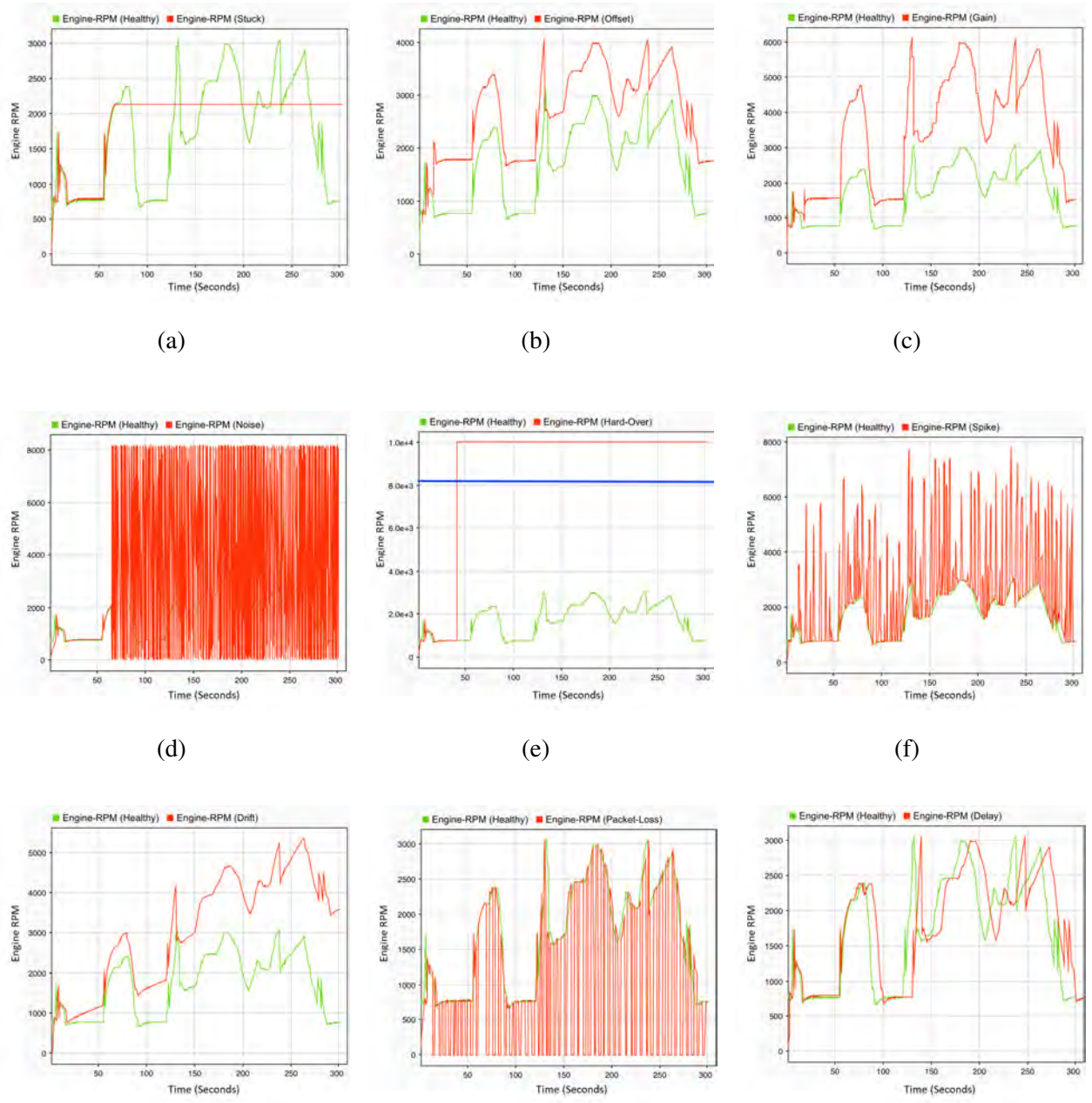


Figure 2.8: Fault types. (**a**) Stuck-at fault, (**b**) Offset fault, (**c**) Gain fault, (**d**) Noise fault, (**e**) Hard-over fault with maximum threshold, (**f**) Spike fault, (**g**) Drift fault, (**h**) Packet loss fault, (**i**) Delay fault.

Fault Type	(dv) Value	(ov) Value
Healthy Signal	1	0
Stuck-at Fault	0	0 or 1, and it varies on time
Offset/Bias Fault	1	fixed constant value
Gain Fault	Greater than 1	0
Noise Fault	1	random value
Hard-Over Fault	0	higher than maximum threshold
Spike Fault	1	value varies on time
Drift Fault	1	value increases on time
Packet Loss Fault	0	0
Delay Time Fault	0	last cycle value of x(t) based on time given

Table 2.1: Value of dv and ov for all fault types.

the reliability of measurements over extended periods.

Given the significant impact of random and systematic hardware faults on the safety of the ASS, this study examined the signal behavior of sensors and communication faults with regard to their influence. The mathematical representation of the fault types mentioned can be shown in the Equation: (2.1) and Table 2.1:

$$y(t) = d_v x(t) + o_v \tag{2.1}$$

where $y(t)$ is a faulty or manipulated signal value, d_v represents the gain value and $x(t)$ is the healthy or standard signal value. o_v represents the offset/bias value.

In contrast, the fault location indicates the place where the fault is likely to occur, for example within an element, a function, or the communication between components and the subsystem. The primary potential points of failure in the vehicle were analyzed in [36]. The authors classified the potential points of failure in a vehicle network into the following categories: functional specifications, network, sensors, actuators, ECUs, gateways, a power supply, vehicle subsystems, and a data acquisition system.

ASSs are composed of numerous heterogeneous components, subsystems, and systems that interact with each other. As the complexity of ASSs increases, the probability of faults also rises. Consequently, both systematic and random faults should be considered during the development process.

In accordance with the ISO 26262 fault classification, faults can be divided into five categories: safe, single-point, multi-point, residual, and latent faults. Dependent faults are defined as those in

which two or more faults occur simultaneously, thereby violating functional safety requirements.

Definition 2.28 (Multipoint Failures)) *A multipoint fault is a fault that leads directly to the violation of a safety objective and results from the combination of several independent faults* [1].

Two classes of dependent faults can be distinguished: common cause faults and cascading faults. The dependent failure analysis identifies the single causes or single events that could result in a violation of a safety requirement or safety objective. In the context of cascading failures, the failure of one component can lead to the failure of another component. Consequently, cascading failures are regarded as dependent failures. In contrast, common cause failures are the consequence of a singular event with a specific cause that results in the occurrence of two failures in distinct components.

2.6 Fault Detection and Diagnosis approaches

In this section, an overview of the current approaches used to develop FDD models is presented. Specifically, model-based approach, signal-based approach, knowledge-based approach, and data-driven approach are presented in subsections 2.6.1, 2.6.2, 2.6.3, 2.6.4, respectively.

Depending on the target tasks of the engineering applications, different models can be developed, including models for fault detection, fault diagnosis, fault identification, fault isolation, and fault prediction.

Definition 2.29 (Fault Detection) *Fault detection is defined as the process of determining the occurrence and timing of an event in a system* [55].

Definition 2.30 (Fault Diagnosis) *Fault diagnosis involves determining the type, size, location, and time of a fault. It includes both fault detection and fault identification* [55].

Definition 2.31 (Fault Identification) *Fault identification aims to determine the extent and approximate time behavior of a fault* [55].

Definition 2.32 (Fault Classification) *Since the goal of fault identification is to identify the type of fault, which is a classification problem, classification-based methods are used to develop a fault identification model. In this study, both fault identification and fault classification refer to the same process* [107].

Definition 2.33 (Fault Isolation) *The function of fault localization is to accurately determine*

the cause and/or origin of a fault [55].

Definition 2.34 (Fault Prediction) *Fault prediction is the estimation of the current number, future occurrence and probable consequences of faults* [13].

2.6.1 Model-based Approach

For dynamic systems, the model-based approach [108], also known as physics-based models, is considered the best option for developing an FDD model. Three different methods have been proposed in the literature, i.e., parity space-based [109], observer-based [110] and parameter estimation-based method [111].

The fundamental tenet of this methodology is to ascertain the analytical redundancy of the developed mathematical model in comparison to the initial data of the real physical system. faults can be identified and classified on the basis of the generated residual signals. Consequently, the fault is defined as the discrepancy between the value predicted by the simulation model and the values actually observed in the real system. The mathematical model represents the dynamic behavior of the system in the absence of faults.

In particular, in the parity space model, the relationships between the system variables are defined as parity equations based on the mathematical models. Once the residuals have been calculated over time, the fault is identified if the residuals are found to be significantly greater than the expected value, in accordance with the defined criteria. One potential application of this method is the process of identifying the root cause of a fault in order to facilitate its isolation. The primary tenet of the observer-based fault detection model aligns with the parity space, with the exception of the system representation. Once the system model has been developed, the system's behavior is represented in the form of a state space, which contains the inputs, outputs, variables, and parameters. The state estimator, which provides the current state, is used to determine the discrepancy between the estimated and the measured state, which represents the fault. Finally, the parameter estimation-based model is designed to identify the key sensitive system parameters whose change indicates the occurrence of a fault. To this end, parameter estimation techniques such as maximum likelihood, Bayesian inference, and least squares are employed to estimate the values of the identified parameters. Once the residuals between the estimated and expected parameters have been calculated, the fault can be detected based on defined thresholds or criteria.

In the automotive industry, some studies have demonstrated the advantages and possibilities of the approach by proposing new techniques for FDD with high accuracy [112, 113]. However, the requisite expertise to construct an accurate mathematical model and the increasing complexity of a modern ASSs present challenges to the implementation of this approach [114].

2.6.2 Signal-based Approach

The signal-based approach, as outlined in [115], is dependent upon the analysis of fault symptoms within the signals. In particular, the healthy signals are analyzed in order to determine whether there is a deviation in the signal characteristics, such as specific frequency, amplitude and ripple characteristics. A fault is present if any of these characteristics deviate from the normal state. Consequently, one of the most crucial prerequisites for the implementation of signal-based methods is the availability of system measurements in the form of time series data. A plethora of methods for performing the aforementioned procedures have been presented in the literature. These include waveform analysis, frequency domain analysis, envelope analysis, threshold analysis, and statistical process control. The selection of appropriate techniques is contingent upon several factors, including the characteristics of the signals, the type of system, and the specific fault scenarios to be addressed. In recent times, hybrid techniques have been put forth as a means of integrating several techniques and expertise in order to benefit from them and overcome their limitations.

Although the signal-based approach is a widely used method for real-time monitoring of industrial systems, especially in steady state, it has some limitations. The assumption that the captured signals contain significant information about the faults necessitates the involvement of expert knowledge about the symptoms of healthy systems to detect the presence of faults when analyzing the measured signals [116]. In this approach, the extracted features of the signals are divided into three categories: the time domain [117], frequency domain [118] and time-frequency domain [119].

2.6.3 Knowledge-based Approach

Knowledge-based methods [120] for FDD rely on explicit domain knowledge about the SUT. Rules, logic, and test engineer expertise are used to identify anomalous behavior. Fuzzy logic systems, expert systems, and causal reasoning are some of the techniques used in the knowledge-based FDD

approach. The fuzzy logic system is a widely used technique for dealing with uncertainty and imprecision in complex systems. Through fuzzification, design of expert rules and membership functions, application of fuzzy operators, and defuzzification, an FDD model based on fuzzy logic can be developed. The main advantage of this method is its ability to adapt to changes in system behavior over time, especially in a dynamic environment. In addition, the interpretable decision model developed leads to improved confidence in the FDD process in complex systems. Similarly, expert systems use domain-specific knowledge together with rules and heuristics to determine the presence of a fault. The core idea of rule-based systems is to use explicit rules to make decisions or draw conclusions from inputs. The main components of expert systems are the inference engine, the rule-based system, the historical data, and the user interface. Finally, causal reasoning is about analyzing the cause-and-effect relationships within a system to identify anomalous behavior. In this way, the causes of observed anomalies can be efficiently determined. However, this requires a deep understanding of the relationships between system components. By monitoring the system variables, the deviation from the normal state can be detected by the model. In addition, the causal relationships can be used through graphs or networks to analyze fault propagation through interconnected components. This makes the method useful for applications where the test objective is to analyze the root causes of detected failures.

A knowledge-based approach does not necessitate the pre-classification of training sets. For further insight, one may consult the work of [121]. However, the knowledge-based approach is limited and insufficient for the construction of an FDD system due to the lack of specific, qualitative expert knowledge and the complexity of modern vehicles [122]. In addition to the extensive human intervention required, the non-linear relationships between the signals in the complex system make it difficult to detect and classify faults.

2.6.4 Data-Driven Approach

In the past decade, the advent of advanced computational resources has facilitated the pervasive implementation of a data-driven approach to the problem of FDD [123]. At the core of this approach lies the concept of extracting knowledge from historical data and constructing nonlinear relationships between input and output classes with the objective of uncovering patterns between the data samples. In contrast to other FDD approaches, neither expert knowledge nor a precise

mathematical model is required, which makes the data-driven approach attractive for a wide range of research areas. The development of FDD methods has grown rapidly in various technical fields.

The advent of sophisticated sensor technology, the rapid evolution of computational methods, the accessibility of substantial computational resources and raw data from measured signals, and the simplicity of model realization with its high efficiency have collectively prompted researchers to prioritize data-driven approaches for FDD models development [124]. The core idea of this approach is to use knowledge constructed from high-quality historical data to detect and classify abnormal behavior. According to [120], FDD methods can be divided into two categories, i.e., data-driven statistical analysis and data-driven non-statistical analysis. The most popular methods of the statistical approach include PCA [125], ICA [126] and PLS [127]. ML techniques based on pattern recognition [128] are frequently employed in the non-statistical approach due to their adaptive learning and non-linear approximation capabilities.

2.7 Machine Learning Methods for Fault Detection and Classification

2.7.1 Machine Learning Methods

The ML techniques are divided into four categories depending on the tasks to be performed, i.e., clustering, classification, regression and anomaly detection. The categories are unsupervised learning methods [129], semi-supervised learning methods [130], supervised learning methods [131], and reinforcement learning methods [132].

Supervised learning requires the labeling of the dataset as a prerequisite for the development of the model. In other words, both the inputs and the outputs of the model should be known in advance. The learning process then establishes a relationship between the features of the samples and the corresponding output classes for decision making. It is therefore considered a suitable option for classification and regression tasks. When the output of the model represents discrete values, such as a class or category, it is called a classification model, while in the regression task the output is represented as a continuous value. Some examples of ML techniques that use a supervised approach are Support Vector Machine (SVM) [133] and RF [134]. Unsupervised learning, on the other

hand, is about analyzing and finding the relationships or connections between the data samples whose labels are not available. Therefore, unsupervised learning is mainly used for clustering and grouping tasks, where data samples with the same behavior are grouped into the same clusters. K-means clustering is a well-known technique from this approach that is used for clustering tasks. Semi-supervised learning was introduced as a middle ground between supervised and unsupervised learning. The main goal of this approach is to overcome the limitation of the labeling process, which requires a significant amount of time and effort. Therefore, the semi-supervised approach is used when the dataset is partially labeled. In this approach, both classification and clustering tasks can be performed according to the objectives of the learning process. Finally, reinforcement learning uses the principle of continuous interaction with the environment and the reward system to develop the target model. More specifically, decision making in this model depends on the feedback the agent receives as a result of its response to the current environment. Some applications of this approach include robotics, autonomous vehicles, and energy management.

Random Forests (RF):

Random Forests (RF) [134] is an ensemble learning technique because it combines the predictions of several individual decision tree (DT). This improves the overall performance and generalization of the developed model. It is worth noting that each tree is trained by RF on a random subset of the data, which in turn reduces overfitting and improves generalization. Tree construction, bagging, DT training and prediction are the main steps to develop an FDD model based on RF. First, a number of DT (N) are collected and used to construct the RF based on the partitioning of the feature space, resulting in a set of decision rules. Then, the bootstrap aggregation technique is used to randomly select and place a subset of the original training set. This ensures diversity among the trees and reduces the problem of overfitting. In the training phase, each tree is trained by recursively partitioning the data based on the selected features until the predefined criteria are met. For example, the maximum depth or the minimum number of samples in a leaf node are the criteria for terminating the training process. Finally, a majority decision mechanism is used for the final prediction decision, taking into account the majority of all separately generated trees.

Mathematically, the prediction output can be represented by the Equation (2.2).

$$T = ModeT_1(x), T_2(x), ..., T_i(x) \tag{2.2}$$

where the predicted output is represented by T, Mode is the majority voting operation, and $T_i(x)$ represents the prediction of the i-th DT in the forest.

RF is considered a powerful technique that can be applied to a variety of tasks from different domains. Classification, regression, and anomaly detection are some applications where RF is used with complex datasets and a large number of features. In addition, it is able to extract features with non-linear relationships and is very robust to noise and unbalanced data.

K-means:

The core of the K-means algorithm [135] is based on partitioning a dataset into different non-overlapping subgroups or clusters. The main steps in the implementation of the method are initialization, assignment, center updating, and iteration. In the initialization phase, the number of clusters in the feature space is randomly selected, along with the initialization centroids of the selected clusters. The optimal number of clusters can also be calculated using a special method, e.g., the elbow method. The elbow method calculates the sum of squared error for different numbers of clusters. The optimal k-value then represents the points where the first sudden drop in squared error value occurs. Once the optimal k-value is selected, the data are assigned to the corresponding cluster whose centroid is closest to it. The euclidean distance is used as an evaluation measure to correct the assignment process so that the samples are grouped in the cluster that is closest to the centroid and has a minimum centroid distance. Subsequently, both the k-selection and the distance calculations are repeated iteratively until the convergence criteria are met. Specifically, training is complete when the specified number of iterations is reached or the centroids no longer change. Mathematically, the sum of the squared distances can be represented in [136]. K-means has been widely used in various applications such as social tagging, shape recognition, and wireless sensor networks. However, the main limitation of the method is its sensitivity to the initial centroid positions, which can lead to convergence to local optima.

2.7.2 Deep Learning Methods

As a subfield of ML, DL [137] has gained prominence in the development of complex FDD models, especially when vast amount of data are involved [138]. This is due to the ability to outperform traditional statistical ML methods and overcome their limitations in terms of feature extraction, computational cost, and dimensionality reduction. Furthermore, the rapid development of graphics processing units (GPUs) has facilitated more efficient processing of large amounts of data, which in turn contributes to the outstanding performance of DL-based approaches. Consequently, there has been a notable increase in interest in the development of FDD systems using DL techniques over the past decade. In this context, various DL architectures, including the deep belief network (DBN), restricted boltzmann machine (RBM), CNN, recurrent neural network (RNN), and AE, have been proposed for different application domains. A comprehensive overview of DL architectures with a focus on current challenges and future developments can be found in [139]. Recently, data-driven approaches including those based on DL and ML, have been recently employed during the software development lifecycle in a variety of phases. These include software requirements engineering, software architecture and design, software implementation, software quality and analysis, as well as software maintenance [140].

In the testing and analysis phase, DL and ML are not only concerned with the generation and selection of TCs, but also with the detection and analysis of faults and anomalies. However, despite the remarkable achievements in various fields, the lack of representative datasets is one of the biggest challenges for DL and ML methods [141, 142]. In addition to obtaining a sufficient amount of high-quality data, obtaining datasets with different scenarios that reflect real-world operating conditions can also be a challenge. Furthermore, the dynamic environment with changing conditions and the need to meet real-time requirements is another complicating factor when applying the data-driven approach with real-time system validation. In the automotive industry, in particular for safety-related real-time systems, it is of the utmost importance to ensure that data-driven approaches comply with the applied development standards, for example ISO 26262 [15]. Consequently, the availability of datasets represents a limitation for the development process of DL-based FDD in real-world applications.

Denoising Autoencoder (DAE):

Autoencoder (AE) [143] is one of the unsupervised DL techniques in which the effective representations of the input data are learned using the encoder-decoder method. Basically, autoencoder is used for dimensionality reduction, so that the information of the input data is compressed in the intermediate layer of its structure. In addition, AE has been shown to be able to extract the features from the input data, that form the basis of the training process, for either classification or clustering tasks. Most importantly, AE has achieved remarkable success in learning compact and meaningful representations of large amounts of input data. This, in turn, has played a crucial role in the development of an effective anomaly detection model based on unlabeled historical datasets. The key components of the AE structure are the encoder module, the decoder module, and the loss function. In the encoder, the received input data is converted into a compressed representation using multiple layers of neurons. The essential features extracted by the encoder are stored in the middle layer, the so-called latent code. In this way, the original high-dimensional input data is transformed into a low-dimensional representation.

On the other hand, the extracted features in the latent code are used to reconstruct the original input data in the decoder part using linear and nonlinear activation functions. At the core of the AE training process is the idea of optimizing the structure using loss function to minimize the computational error between input and output. As an extension to AE, the DAE [144] was introduced to overcome the challenge of noisy data during the learning process. Due to its ability to learn robust representations of data in the presence of noise, it has been widely used in engineering applications such as FDD, image denoising, and speech signal denoising. In addition to the above architecture of standard AE, a corrupting layer is added after the input layer to corrupt the input data by adding a certain level of noise. Typically, Gaussian noise is used for this purpose. As a result of the DAE training procedure, the noisy input data is presented and effectively reconstructed without noise. Figure 2.9 shows the architecture of the DAE, including the encoding and decoding modules.

Mathematically, the autoencoder part of the DAE is represented by the following equations:

$$Z = f_{enc}(WN + b) \tag{2.3}$$

where f_{enc} represents an activation function of the encoder, W is an encoder's weight, and b is an

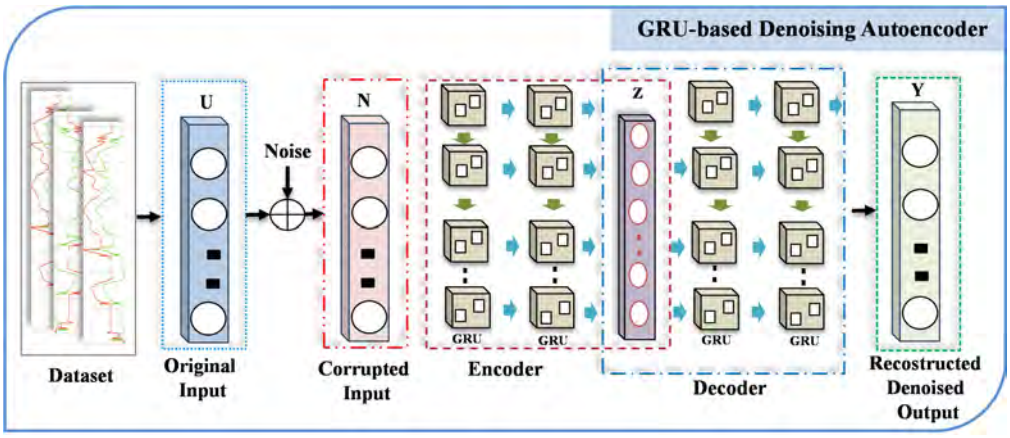


Figure 2.9: Denoising Autoencoder

offset vector.

On the other hand, the decoder part is mathematically represented by the following Equations:

$$Y = f_{dec}(W'Z + b') \tag{2.4}$$

where f_{dec} denotes an activation function of the decoder, and W' and b' represent the decoder's weight and offset vector of the decoder, respectively.

Mathematically, the loss function $L(U, Y)$ is calculated according to Equation (2.5):

$$L(U,Y) = ||Y - U||^2 \tag{2.5}$$

Long Short-Term Memory Network (LSTM):

The study in [145] represents one of the earliest investigations into the utilization of LSTM networks. The defining feature of the LSTM architecture is its capacity to learn data with long-term dependencies. The LSTM cell enables the retention of past information, overcomes the inherent limitation of the RNN, i.e., the disappearance of the gradient and gradient explosion. Subsequently, due to its high performance in feature learning and classification results compared to other deep neural networks, LSTM has been applied to a variety of complicated sequential problems, especially in the case of dynamic information of time sequences. For example, in the field of fault monitoring, LSTM was used to develop an FDD model for the Tennessee Eastman (TE) process in [146]. Similarly, in the field of unmanned aerial vehicles (UAVs), the detection of sensor anomalies

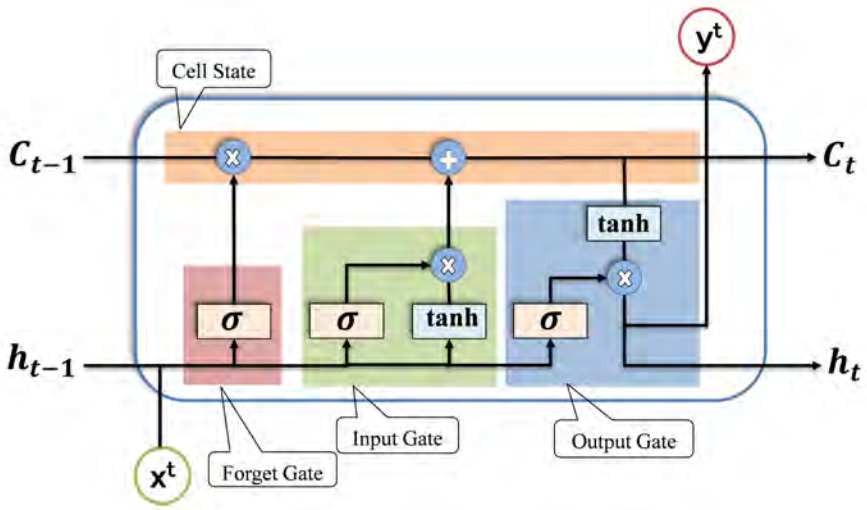


Figure 2.10: Internal structure of LSTM cell

with LSTM was investigated in [147]. In [148], an improved method for the diagnosis of bearing damage in electrical machines is presented, which is based on the integration of LSTM and CNN techniques into a unified structure. It is noteworthy that several studies have been conducted in the automotive field to investigate novel methods for FDD. These studies have explored the use of LSTM alone or the integration of LSTM with other DL techniques, targeting diverse automotive systems. Potential sources of faults include faults in engines [149], EV batteries [150], and fuel cell vehicles [151].

The LSTM is comprised of the chain units, also known as the cell network. The internal structure of the cell is formed by three different gates, i.e., the forgetting gate, the input gate, and the output gate, which are connected to the cell state in a specific manner. This is illustrated in Figure 2.10. The decision as to which information is to be eliminated from the cell state is determined by the forget gate. In contrast, the input gate is responsible for deciding which new information should be stored in the cell state. To this end, the sigmoid is calculated to determine which values should be updated. Subsequently, tanh is employed to generate a vector of new candidates to be added to the cell state. Finally, tanh and sigmoid functions are combined and added to the cell state.

Mathematically, Sigmoid and Tanh are represented in Equation (2.6) and (2.7), respectively. Equation (2.8) applies to the cell state update, where g_t and d_t represent Sigmoid and Tanh output,

respectively. C_t is the output of current state and C_{t-1} represents the output of previous state.

$$g_t = \sigma(W_g.[h_{t-1}, x_t] + b_g) \tag{2.6}$$

$$d_t = tanh(W_d[h_{t-1}, x_t] + b_d) \tag{2.7}$$

$$C_t = (f_t \times C_{t-1}) + (g_t \times d_t) \tag{2.8}$$

The value of the gate depends on h_{t-1} and x_t. h_{t-1} is the output of the previous state, where $f_t \in 0, 1$. f_t is further multiplied by the current state value. If $f_t = 1$, the state of the cell is fully maintained, otherwise it partially depends on the value.

Finally, the output of the cell is determined by the output gate. With Tanh of the cell state, the output is calculated and then multiplied by the output of the Sigmoid. Mathematically, Sigmoid in the output cell is shown in Equation (2.9), and the output of this gate is shown in Equation (2.10)

$$O_t = \sigma(W_O.[h_{t-1}, x_t] + b_O) \tag{2.9}$$

$$h_t = O_t \times tanh(C_t) \tag{2.10}$$

Convolutional Neural Networks (CNNs):

In the present era, among the multitude of ML algorithms, such as the SVM and DT algorithms, the CNN is regarded as the optimal network for FDD. In comparison to traditional manual methods, CNNs possess a valuable attribute, i.e., the capacity to automatically extract features. This represents a pivotal step in the learning algorithm and results in a reduction of time and effort in the training process. In addition to the success of CNN in specific applications with two-dimensional data, such as image recognition and speech recognition, a 1D-CNN has also proven successful in the field of FDD. It shows high performance in terms of accuracy and training time. The CNN algorithms and their applications for FDD, i.e., the detection, identification, and estimation of fault magnitudes, were analyzed in a comprehensive manner in [139, 152]. Other proposed studies to use the 1D-CNN for FDD of rotating machines can be found in [153, 154, 155].

CNN is a DL model developed on the architectural foundation of the artificial neural network (ANN) and incorporates additional layers to address the shortcomings of ANN. Feature extraction and classification represent the two primary phases of a CNN [156]. In the extraction phase, an

input layer, convolutional layers, and pooling layers are organized in a network configuration. This network's objective is to autonomously extract and learn features from raw data. In the classification phase, the features derived from the final pooling layer are transmitted to the fully connected layers. The classification tasks are then performed on this basis, taking into account the learned features from the preceding layers.

Mathematically, convolution and pooling are expressed in Equation (2.11) and Equation (2.12), respectively.

$$f(y)_i = AF(\Sigma\ w(i,j) \times y(i) + b(i)) \tag{2.11}$$

$$x_i = Pool_{(m,n)}(f(y)_i) \tag{2.12}$$

where AF is an activation function like Sigmoid and ReLU, $y(i)$ is the ith signal features, $w(i,j)$ and $b(i)$ represent weight and bias, $f(y)_i$ is the output feature extraction and (m,n) represents the nth pooling area with size m.

3 Problem Analysis and Research Questions

The primary objective of this chapter is to conduct a comprehensive analysis of the research problem at hand and to identify the specific challenges to be addressed in this study. To this end, besides highlighting the main challenges of current practices in failure analysis process during HIL tests, the research questions to be answered are derived and stated. To illustrate the problem domain and scope, an overview of the current concepts and approaches of real-time validation during the system integration phase of ASSs according to ISO 26262 is presented in Section 3.1. Furthermore, the types of faults to be detected and identified as a result of HIL tests are presented in Section 3.2. In order to provide a comprehensive and understandable analysis of the problems that arise in this context, the validation process of the gasoline engine system is considered as an illustrative example in Section 3.3. Then, in Section 3.4, the conventional techniques used for failure analysis are presented, focusing on the main challenges in this field. Finally, in light of the aforementioned challenges, the research questions to be addressed by this work are defined in Section 3.5.

3.1 Validation Process According to ISO 26262

This section presents the fundamental methodology employed for the validation of ASSs in accordance with ISO 26262. First, the motivation and objectives of the functional safety standard in the automotive domain are presented. Subsequently, an overview of the standard's structure is provided, with a particular focus on product development at the system level. The structure is then utilized to discuss the concept of system integration and testing level, including the primary test methods. Finally, the current test platforms for implementing the test methods are presented.

In order to standardize the development process of safety-related ASSs, the functional safety standard ISO 26262 has been introduced as a uniform and generally accepted development approach. Some examples of safety-related systems mentioned are Adaptive Cruise Control (ACC), Power Assisted Steering (PAS), and Electronic Stability Control (ESC). The primary objective of

the standard is to guarantee the functional safety of E/E systems by defining the requirements specifications and recommendations for the development process. ISO 26262 is structured in ten sections, each corresponding to a phase of the V-model, and encompasses all activities throughout the development lifecycle, including production and operation. This is illustrated in Figure 3.1.

Definition 3.1 (Safety-Critical Systems) *Safety-critical systems are defined as systems whose failure could result in loss of life, significant damage to property or environmental damage* [157].

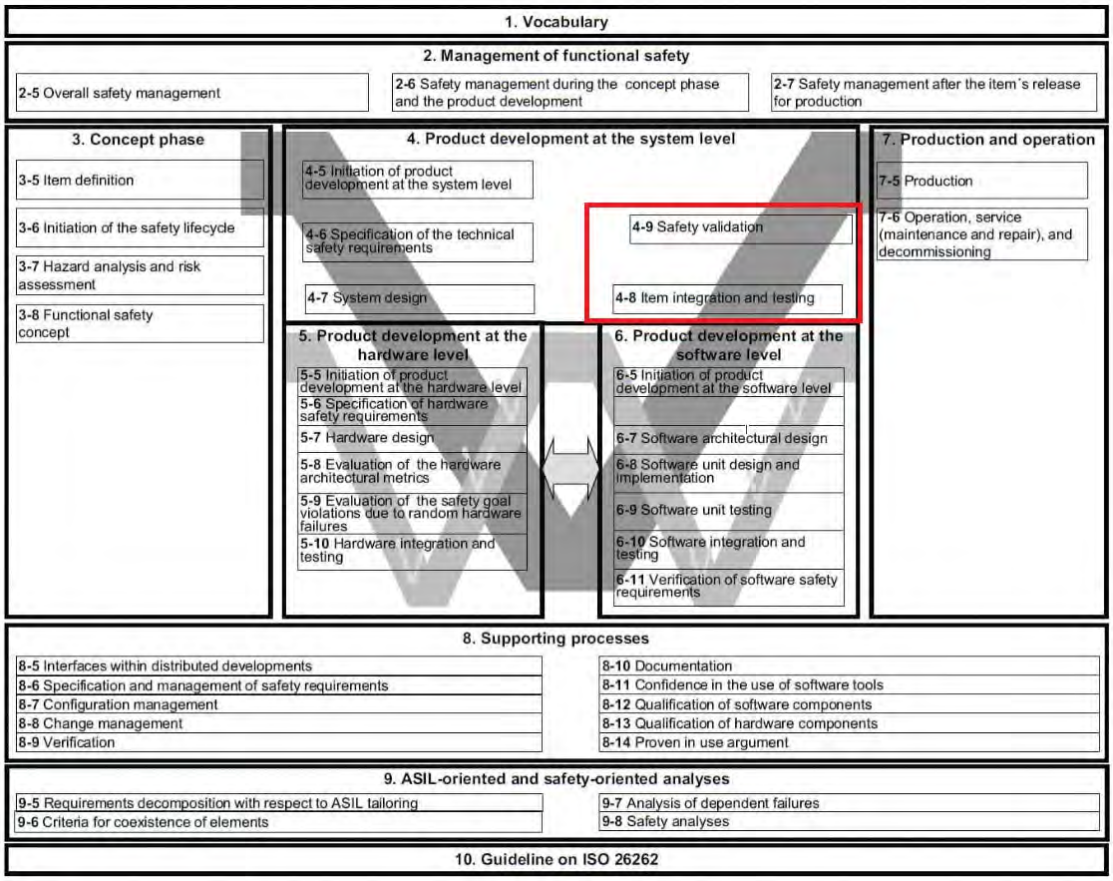


Figure 3.1: ISO 26262 [1]

A review of the standard reveals that the product development stages have been delineated into three distinct categories: system level (part 4), hardware level (part 5), and software level (part 6). In each of the aforementioned levels, the testing process is conducted on the right-hand side of the V-model. During the validation process, the SUT should be tested at different integration levels, commencing with the basic software modules and concluding with the entire operational vehicle system, encompassing both software and hardware elements. For instance, the product

development at the software level (part 6) encompasses three levels of testing: 6-9 software unit testing, 6-10 software integration and testing, and 6-11 verification of software safety requirements.

In accordance with the ISO 26262 structure depicted in Figure 3.1, the purview of this study encompasses two integration phases of product development at the system level (Part 4), specifically 4-8 System and item integration and testing, and 4-9 Safety validation. In these phases, the developed embedded software is validated in the actual target environment to ensure the functionality and safety of the developed product. At this level, the environmental conditions and the bus communication between the SUT (real ECUs) and other vehicle systems with real elements are considered in the validation process. The integration of the item's elements is conducted in three stages. Initially, the software and hardware are integrated. Subsequently, the integration of systems from other technologies and external measures or systems is undertaken. Finally, the overall system integration is completed, leading up to vehicle integration.

The primary objective of these levels is to guarantee that the interactions between system components occur in an optimal fashion, aligning with their intended functions. Furthermore, this should demonstrate that the safety objectives have been met without compromise under various operating conditions. In other words, system integration levels are designed to verify that the system design has been correctly implemented by the entire system and that all safety requirements have been met in accordance with their specifications and ASIL classification. A variety of testing methods have been proposed by ISO 26262 for the purpose of detecting faults during the system integration phase. Some of the aforementioned testing methods include requirements-based tests, back-to-back tests, interaction/communication tests, FI tests, tests derived from field experience, and long-term tests as user tests under real-life conditions. The aforementioned methods permit the performance of two types of tests at the system integration level: systematic and non-systematic tests.

Systematic testing can identify systematic faults that result from deviations in the specifications. Requirements-based testing and back-to-back testing can identify software faults resulting from inadequate requirements, ineffective design, or flawed implementation practices. In general, the TCs employed in these tests exhibit a consistent structure and procedure, whereby an expected output is compared with the actual output of the SUT. The results of the test can be scored as pass or fail, depending on the execution of the TCs [90]. In contrast, non-systematic or exploratory

testing, such as tests derived from field experience and driving tests, is designed to uncover unknown, unexpected faults in the target system through experimental test execution. This approach is exemplified by the work of [158], who describes the process of detecting unexpected deviations in system behavior through the observation and analysis of system behavior. It is noteworthy that this type of testing necessitates domain knowledge to enable the observation and analysis of system behavior in order to detect unexpected deviations that may not be covered by predefined TCs in systematic testing. In the context of driving tests, the SUT is validated under both normal and abnormal conditions in a virtual or real driving environment. Based on realistic conditions and a variety of driving scenarios, a substantial quantity of data samples is collected and analyzed in order to identify any deviations in the system's behavior.

A number of test platforms have been recommended by ISO 26262 for the implementation of test methods. According to the V-model, MIL, SIL, PIL, HIL, VIL and real vehicle [22] are the main recommended platforms. In order to benefit from and avoid the limitations of the pure simulation platform and the real test drive, HIL simulation has recently been introduced as a reliable, safe and flexible platform [159]. For further insight into the role of HIL simulation in the development of safety-related systems in accordance with ISO 26262, refer to [4]. Notably, it is possible to connect physical vehicle components or systems to the HIL simulator, thus enabling the assessment of the impact of these components or systems on the ECUs. For instance, in the virtual test drive platform, real steering, mechanical loads, pedals, and other real sensors and actuators are connected to the HIL system to provide dynamic test benches. Different HIL systems have been developed based on the test object. In particular, when testing individual ECUs, component HIL is used with a simulation model of the controlled system (plant). In contrast, system HILs and networked HILs are employed for the testing of connected real ECUs with partial or complete networks within the vehicle, as outlined in [160].

3.2 Sensors and Actuators-related Faults

This section outlines potential sensor-related faults that can occur in automotive systems, including the type and locations. It is imperative that these faults be identified and addressed during the validation process to ensure the reliability of the system.

In the cited source [55], it is explained that the deviation of the behavior of the components from the normal state, which leads to the failure of an element, is referred to as a fault. Due to the environmental and working conditions, the hardware components, i.e., sensors and actuators, in an ASS are inevitably at risk of failure. In addition, control units, networks, gateways, a power supply, vehicle subsystems, and a data acquisition system may also be potential sources of fault in a vehicle network [158].

It is noteworthy that, according to the ISO 26262, abnormal conditions that could lead to the failure of a specific component or object are designated as "faults". Accordingly, in our study, the term "abnormal conditions" was employed to describe the system behavior in the presence of faults in the system components, specifically sensors and actuators. In contrast, the term "normal condition" is used to describe the system's behavior in the absence of faults, or in other words, in a "golden run" scenario. It is also worth noting that unexpected sensor-related faults in the complex ASSs could propagate through the components and subsystems, known as fault propagation [13], causing a significant failure in the output of the functions. Failures resulting in fault propagation through the vehicle functions, especially in ADAS and autonomous driving, lead to violation of safety objectives and severe consequences. In the case of autonomous vehicles, decision making is based on the data collected by the multiple sensors, which not only observe the operating status of the system, but also identify the environmental situation. Failures in these components could lead to functional safety risks in autonomous vehicle applications [161]. For this reason, fault diagnosis plays an important role in analyzing the nature or causes of the underlying safety risks. By providing insight information for subsequent decision making, not only can such risks be avoided, but the intended functionality can be maintained to ensure safety.

In order to perform a fault diagnosis, the signs, patterns, and symptoms extracted from the collected data are analysed in order to identify the nature of the faults. Despite the vital role of fault detection in identifying the existence of faults, it does not permit the system to interpret these faults and their impact on the vehicle's decision-making processes. Consequently, the identification of the nature and location of the detected faults is of paramount importance in order to provide a reasonable interpretation of the condition and situation. This, in turn, allows for the development of appropriate strategies to maintain functionality and safety aspects in the event of a fault occurring.

In the literature, different types of sequential data-related faults are referred to as data-centric

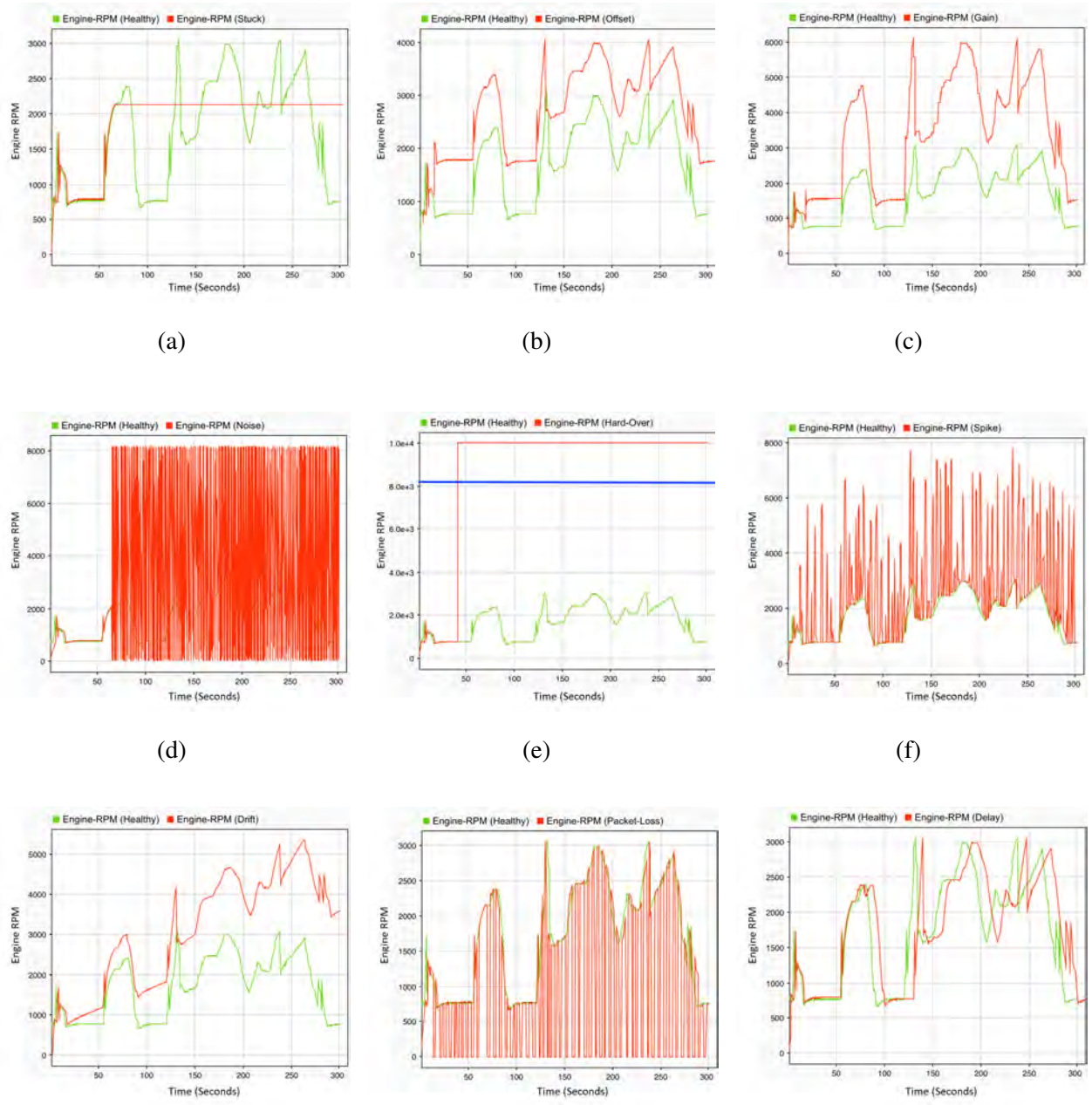


Figure 3.2: Fault types. (**a**) Stuck-at fault, (**b**) Offset fault, (**c**) Gain fault, (**d**) Noise fault, (**e**) Hard-over fault with maximum threshold, (**f**) Spike fault, (**g**) Drift fault, (**h**) Packet loss fault, (**i**) Delay fault.

and system-centric faults [162, 163, 164]. Examples of sensor faults include gain, offset/bias, noise, hard-over, spike, stuck-at, packet loss, delay and drift as shown in Figure 3.2. Various factors can lead to faults within or between system components. The authors in [165] identified several direct and indirect causes for these faults. Direct causes of these types of faults include dirty or damaged sensors, faulty vehicle components, vibrations caused by dangerous internal motors, and sudden acceleration. Conversely, anomalies in signal behavior can be caused by factors such as bumps in the road and unsafe driving practices, although these factors do not necessarily lead to faults.

3.3 Real-Time Validation of Gasoline Engine: Illustrative Example

In this section, the gasoline engine is used as a case study to illustrate the activities of performing the real-time validation process during the integration phase of the V-model. Figure 3.3 shows the phases of the real-time validation process of the SUT based on the HIL simulation platform. More detailed information can be found in the following subsection, which describes the individual phases of the aforementioned methodology. In particular, the system development phase is presented in subsection 3.3.1. The design and creation of test artifacts is described in subsection 3.3.2. The structure of the HIL test platform is presented in subsection 3.3.3. Test execution steps with the results are demonstrated in 3.3.4. Finally, subsection 3.3.5 presents the test evaluation and failure analysis process.

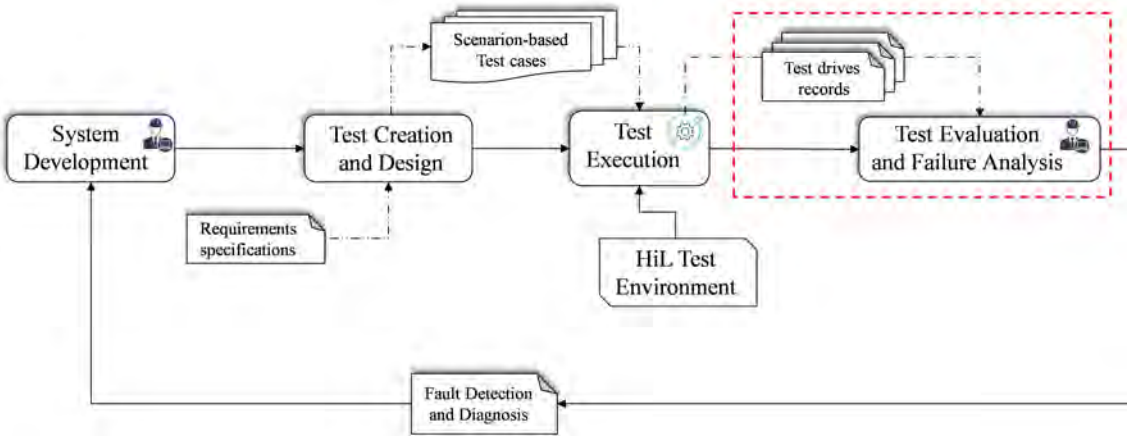


Figure 3.3: System integration test process based on HIL simulation

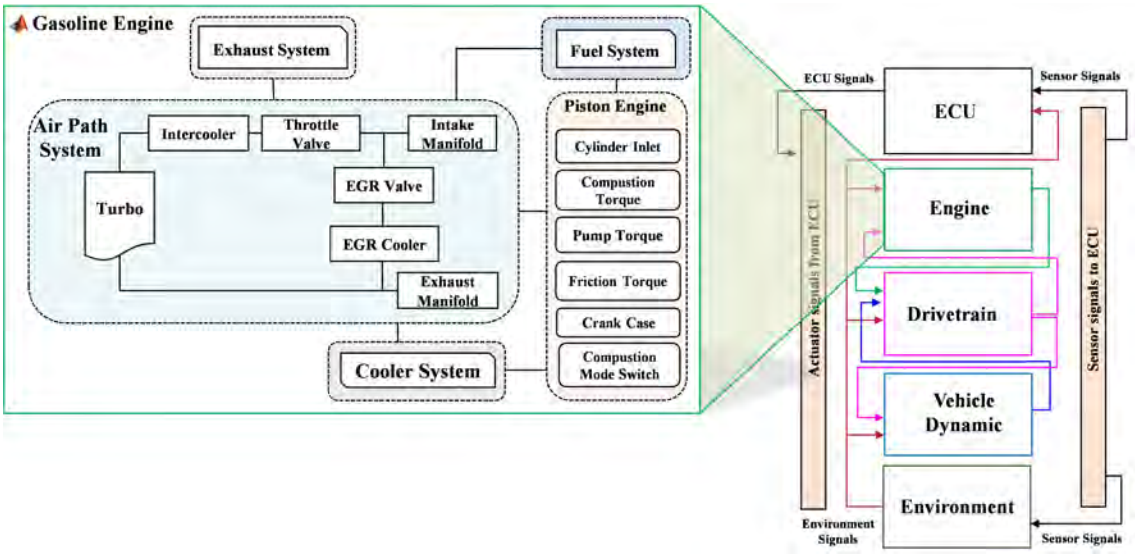


Figure 3.4: System architecture of the case study.

3.3.1 Gasoline Engine System Development-Architecture and Functionality

The Automotive Safety Integrity Level (ASIL) of the ISO 26262 standard categorizes faults in the engine management system into classes C to D, which are considered very critical and serious. As a consequence, faults in such systems should be subject to rigorous analysis and mitigation during the development process. In this study, a high-fidelity system model of a gasoline engine provided by dSPACE [166] is used as a case study. Created in the MATLAB/Simulink environment, the behavioral model includes all subsystems and components to provide a comprehensive representation of the detailed system characteristics. As illustrated in Figure 3.4, the inclusion of an environment model is essential for a realistic simulation and capturing the comprehensive characteristics of the system behavior. Specifically, the powertrain, vehicle dynamics and the environmental system are simulated along with the engine system. Longitudinal driving characteristics, including vehicle resistances, transmission models, vehicle resistances and driver characteristics, are modeled for both the powertrain and the vehicle dynamics system. In addition, the environment model includes a maneuvering model, a driver model and a road model.

Definition 3.2 (Maneuver) *Manoeuvres are defined as any action performed by dynamic objects, such as vehicles. A manoeuvre can be instantaneous or have any duration. Unless otherwise defined, manoeuvres can be combined to form more complex manoeuvres* [81].

The control system is represented in a separate model and then implemented in the actual ECU. The ECU model is employed to perform the control logic. The gasoline engine is modeled in detail, including all of its physical characteristics. The principal components of the system include the air path system, fuel system, piston engine system, exhaust system, and cooler system. In the form of a closed loop, the sensor signals are transmitted from the environment sensors to the corresponding ECU, which in turn analyzes the signals and makes the decisions. Subsequently, the command signals are transmitted from the ECU to the corresponding actuators, thereby enabling the desired behavior to be achieved. It is important to note that in this phase, the internal and external system parameters are set according to the system specifications. This enables the engine to be parameterized, including the number of cylinders, piston displacement, and injection maps.

3.3.2 Test Creation and Design

The creation of the test artefacts is comprised of four distinct steps. These include the design of the driving environment, the specification of the testing conditions, the creation of test scenarios with TCs, and the configuration of the acquisition system. Despite the system complexity of the case study, the software tools provided by dSPACE [167] permit both configuration and setup of the selected system within the user environment. The ModelDesk, ConfigurationDesk, MotionDesk, and ControlDesk tools are employed to facilitate the completion of these steps.

In addition to configuring the vehicle, ModelDesk enables the creation of the test environment, which encompasses the road, driving maneuvers, and traffic. In particular, the road networks can be constructed with their specific characteristics, including lanes, lines, and traffic signs; intersections; height, inclination, and surface. To define the movement of dynamic objects in the driving environment, such as vehicles and pedestrians, ModelDesk is employed to design driving scenarios.

The MotionDesk tool enables the visualization of HIL simulations as 3D online animations, thus facilitating the visualization and comprehension of the simulated system behavior within 3D scenes. Furthermore, weather conditions such as rain, snow, smoke, and sunshine can be configured.

ControlDesk can be employed to configure experiments, including instrumentation, online parameterization with access to simulation platforms and connected bus systems, diagnostics, and measurement execution. Furthermore, as part of the data acquisition system, the target sensors

and communication signals are selected, accessed, and acquired in real-time. A multitude of sensor signals are available for the purpose of capturing the simulated system behavior, including the crank angle sensor, battery voltage, accelerator pedal position, ignition and starter demand, EGR mass flow, engine speed, intake and exhaust manifold pressure, fuel pressure, coolant temperature, and railbar. The selected actuator control signals include the control of the fuel metering unit per cylinder, the injection angle per cylinder, the injection time for direct injection, and the control switches.

The model-based design concept enables the generation of model code for both control and controlled systems through the use of software tools. In this study, ConfigurationDesk is employed to generate the model code, which is subsequently deployed into the target real-time simulation system. In order to configure the driving scenario, the driving cycle can be selected from the list provided by dSPACE, as this represents the desired behavior. Figure 3.5 presents the selected driving cycles, specifically the Highway scenario, which represents the target system behavior as vehicle speed over time.

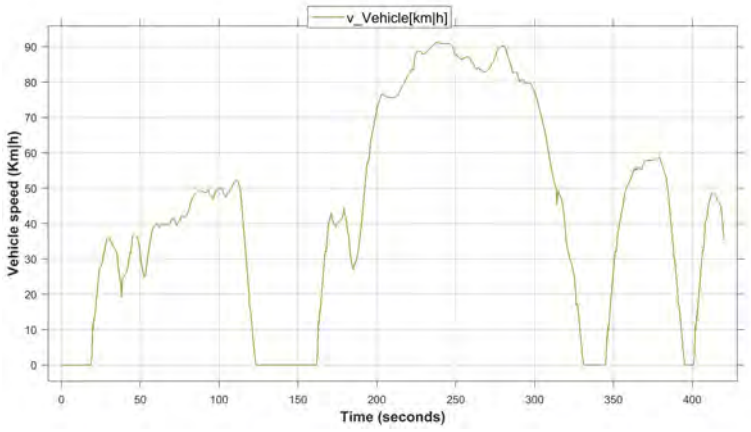


Figure 3.5: The selected driving scenario for HIL tests as a desired system behavior

3.3.3 HIL Test Environment

HIL simulation is a widely utilized methodology for the verification and validation of real-time embedded systems across a multitude of fields, including automotive, railroad, and aerospace. The system offers efficient, rapid, realistic, and highly accurate simulation with repeatability. The two primary components of the HIL simulation platform are the real-time simulator and the real ECU.

The HiL simulator is connected to the real ECU via the CAN bus, allowing the embedded real-time control system to interact with the mathematical models simulated and used in the HIL simulator. In certain cases, it is also possible to connect real vehicle components such as sensors, actuators, steering wheel, and pedals to the HiL simulator via electrical interfaces. The development of the control model and the system model takes place at the simulation level with the aid of model-based development methodology. In the initial stages of software system development, the Simulink and SCADE tools are frequently employed to design, simulate, and verify the software. The system model encompasses the system components with bus communication, the environment models, and other vehicle simulation models.

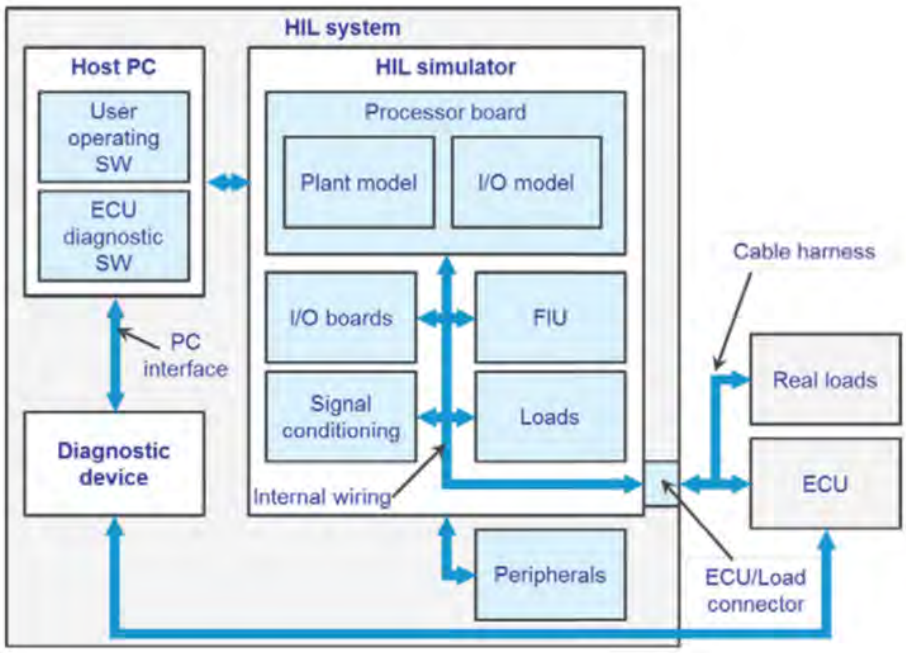


Figure 3.6: HIL system architecture [4]

The utilization of real sensors connected to the HIL simulator, or alternatively, a simulated sensor model, enables the real controller to obtain sensor data from various sources and transmit the positioning command to the simulated actuator in the HIL simulator, thus enabling the execution of the action in accordance with the control logic. The software tools on the host PC, which are connected to the HIL simulator via the Ethernet bus, are employed to configure the HIL test bench. The host PC is responsible for running the modeling environment, defining the configuration, setting up and using the real-time application, and monitoring and analyzing the test run on the HIL sim-

ulator. The DS6001 processor card is the fundamental component of the HIL simulator and serves as the primary processing unit for the real-time application. The processor cards are equipped with interfaces to I/O cards and the host PC.

The structure of the HIL system is depicted in Figure 3.6. It can be observed that, contingent upon the specific test objective, mechanical components and real loads may be incorporated into the system. For instance, when validating the ECU of a steering system, a physical steering system with all the necessary drive elements is connected to the HIL simulator. Furthermore, when developing sophisticated automotive functions such as ADAS, real environmental sensors are incorporated to assess the impact of integrating the real ECUs with other vehicle components.

3.3.4 Test Execution

One of the main features of the HIL system is to enable two driving modes during the validation process, i.e., manual and automatic driving. In the automatic driving mode, the selected predefined scenario is executed by the simulator without human intervention, in accordance with the configured settings and the defined TCs. In contrast, the manual mode enables the user to engage in real-time manual driving within a virtual environment. In this manner, the tester's behavior is incorporated into the validation process. In order to achieve this, the HIL simulator is connected to a real steering wheel and pedals. Consequently, once the test configurations have been specified, the driving scenarios can be executed manually or automatically. In the absence of faults, the standard system behavior, including the signals from the sensors and actuators, can be captured in real-time, thereby generating a healthy dataset, as illustrated in Figure 3.7

The time sampling rate of the recording system is set to 0.001 sec for driving cycle 400 sec. The engine temperature [degC], engine speed [rpm], rail pressure [bar] and vehicle speed [Km|h] are selected in both modes as system variables to detect and analyze the failure. It is worth noting that the system variables selected for the evaluation phase were those that play a critical role in determining the system state, i.e., healthy or faulty. On the other hand, the output signals of the SUT (real ECU) were not considered. The reason behind this is that the ECU output signals represent commands to the actuators that contain digital values, i.e., "0" or "1", which do not contribute significantly to failure analysis.

To illustrate the impact of faults on system behavior, the fault types described in section 3.2

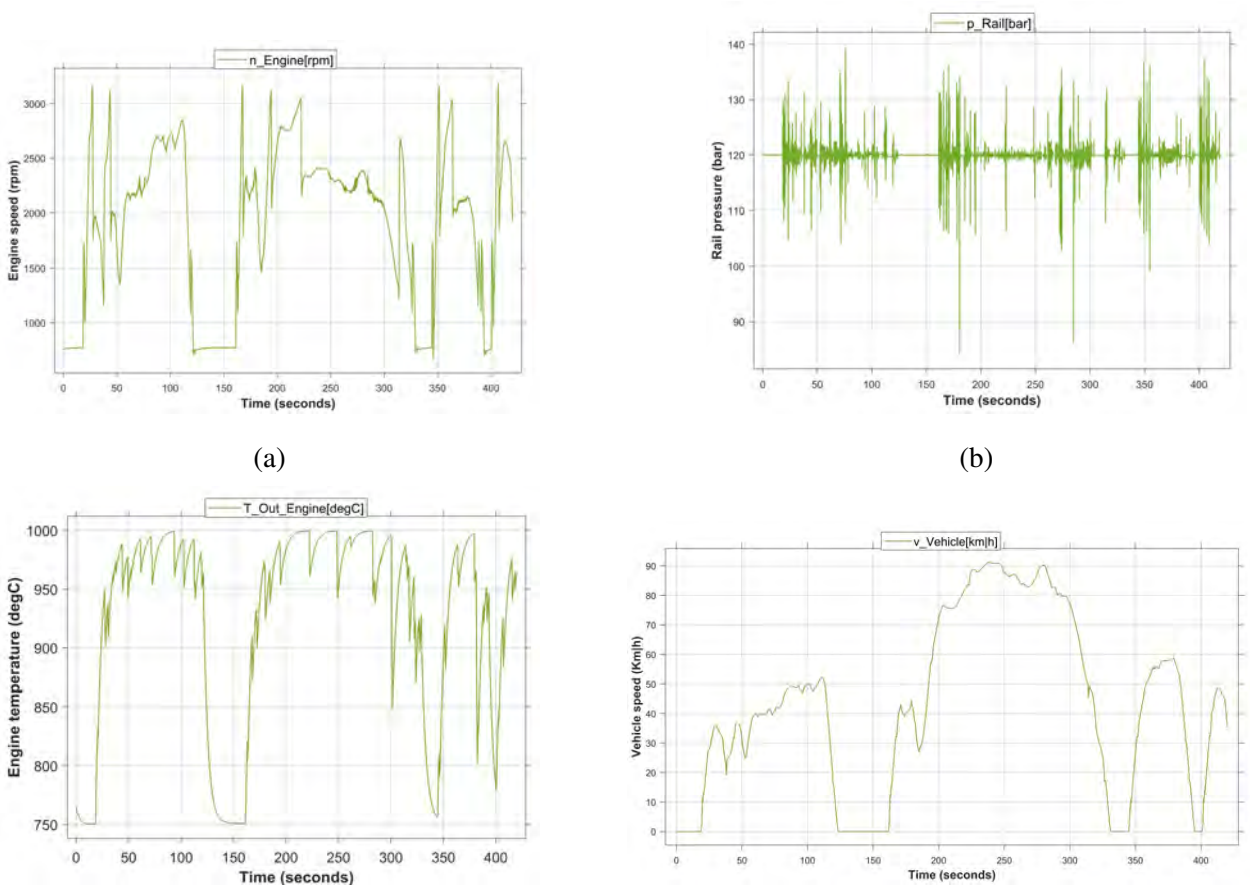
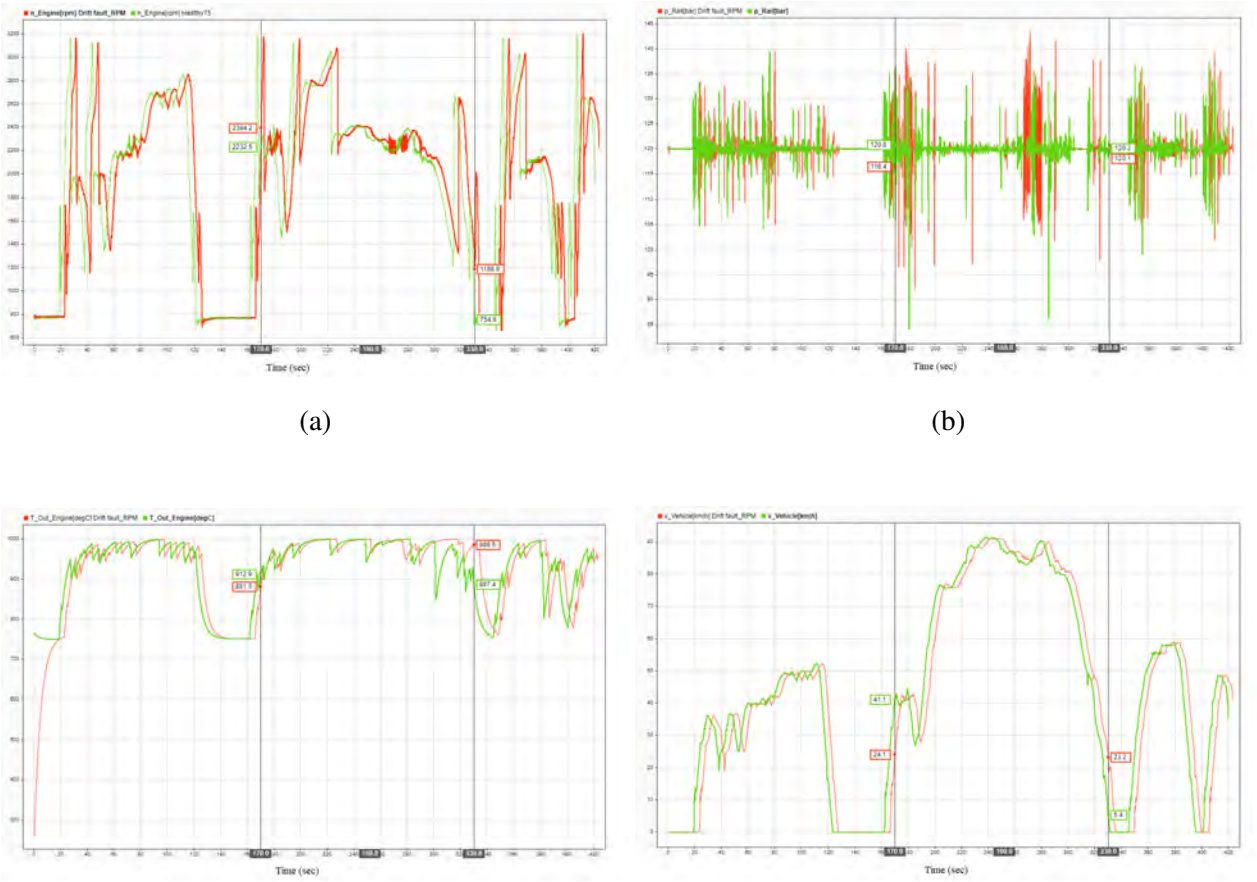


Figure 3.7: System behavior under fault-free conditions.
(**a**) Engine speed.(**b**) Rail pressure (**c**) Engine temperature (**d**) Vehicle speed

are introduced individually and simultaneously into the target components, namely the accelerator pedal position (APP) sensor and the engine speed sensor. For transient faults, the faults are introduced during real-time execution between 170 and 330 seconds, while for permanent faults, they are introduced throughout the entire driving cycle. The effects of simultaneous fault samples are also demonstrated as a combination of two fault types. For this purpose, two different faults are injected simultaneously into the APP sensor and the speed sensor. Figures 3.9 and 3.10 illustrate the system behavior for single and simultaneous faults, respectively.

In consideration of the user behavior in the validation process, Figure 3.11 shows the system behavior without faults captured by conducting the driving scenario by the user in real-time.



(a) (b)

(c) (d)

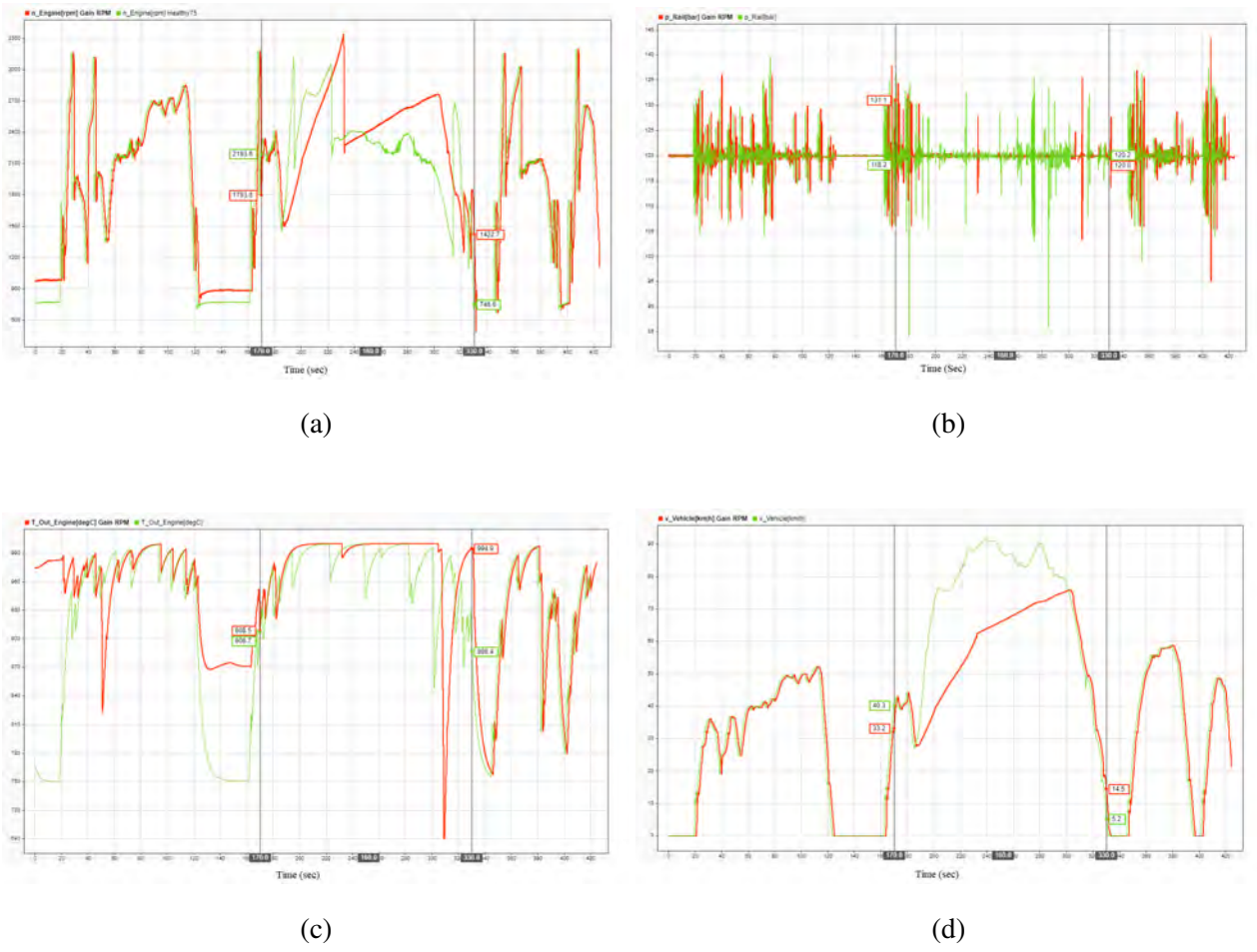
Figure 3.8: System behavior under single fault conditions (drift fault)

(**a**) Engine speed under drift fault.(**b**) Rail pressure under drift fault (**c**) Engine temperature under drift fault (**d**) Vehicle speed under drift fault

3.3.5 Test Evaluation and Failure Analysis

The final phase of the HIL test is the evaluation of the test results, during which the deviations in system behavior are identified. To illustrate the current limitations of failure analysis methods, the output of the test execution, as detailed in Section 3.3.4, will be discussed and analyzed in this section. In particular, the system behavior under single faults and concurrent faults from manual and automated driving is presented.

Figure 3.8 and Figure 3.9 illustrate the system behavior under single faults, as determined by an automated virtual test drive. In the same figures, the standard system behavior, including healthy samples, is presented in green under fault-free conditions.



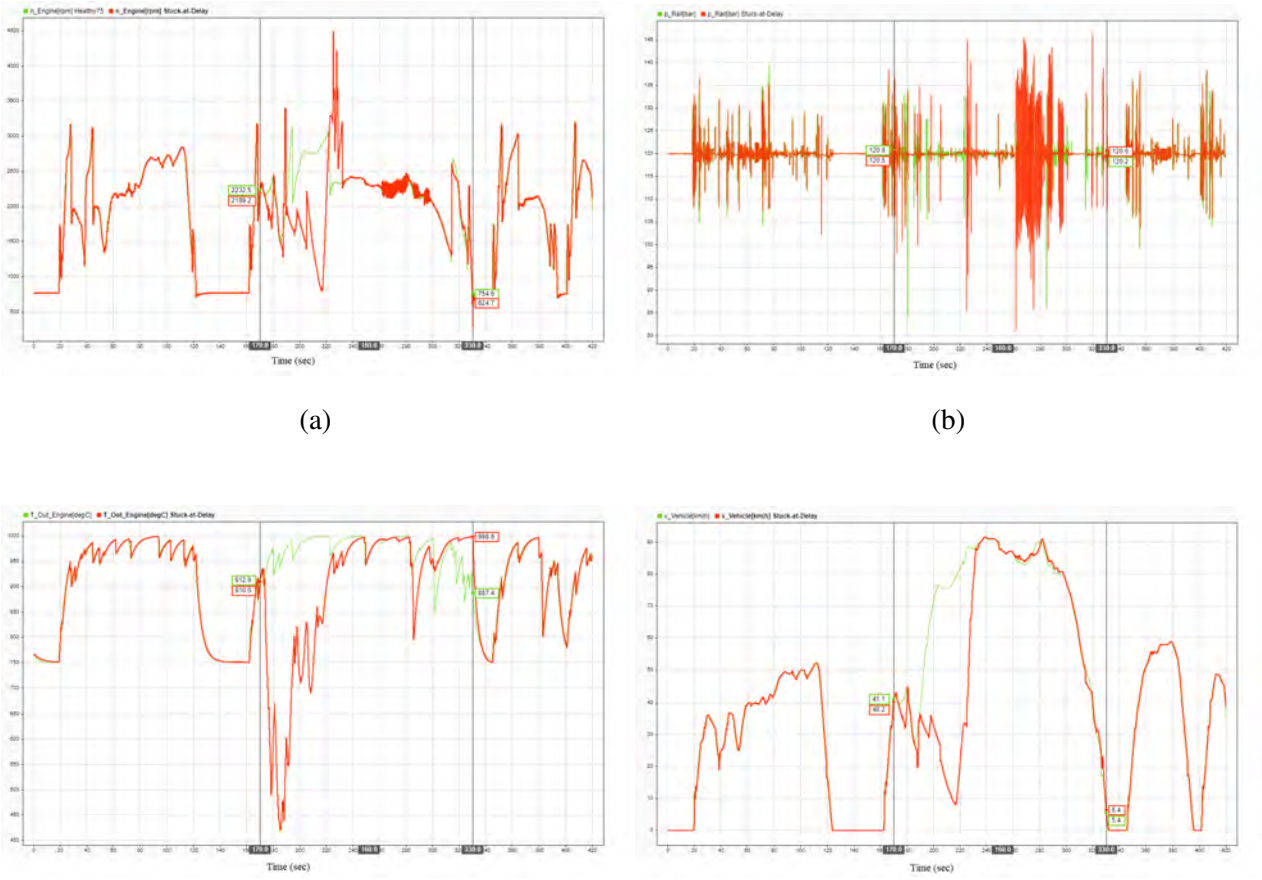
(a) (b)

(c) (d)

Figure 3.9: System behavior under single fault conditions (gain fault)

(**a**) Engine speed under gain fault.(**b**) Rail pressure under gain fault (**c**) Engine temperature under gain fault (**d**) Vehicle speed under gain fault

The aforementioned figures demonstrate that the captured dynamic system behavior is represented as multivariate time-series data. Consequently, the state of the system cannot be determined by examining a single variable. The interdependencies between the representation of non-linear signals and dynamic operations present a complicating factor for the analysis process based on expert knowledge. Furthermore, although current industrial tools are capable of accurately identifying deviations in system behavior, they lack the capacity to provide a reasonable interpretation of the detected anomalies. Moreover, the utilization of these tools may result in the omission of alarms in the event of a system failure. This is due to the fact that the fault does not necessarily result in the failure of the intended functions [71, 168]. Such faults are designated as "dormant faults." In



(a) (b)

(c) (d)

Figure 3.10: System behavior under simultaneous fault conditions (stuck-at and delay faults). (**a**) Engine speed under stuck-at and delay faults.(**b**) Rail pressure under stuck-at and delay faults(**c**) Engine temperature under stuck-at and delay faults (**d**) Vehicle speed under stuck-at and delay faults

other words, the correct state of the system is not altered by the occurrence of faults unless the faults propagate through the components until the system exhibits a failure. The discrepancy between the measured values and the true specified value can be corrected by the system's protection, redundancy, or remain latent in the system. In accordance with ISO 26262, faults that do not result in a contravention of the safety objectives are designated as safe faults. A comparison between Figures 3.8 and 3.9 reveals that the fault occurred in the same component at 170 seconds of the drive cycle, resulting in a deviation in the system's behavior. Nevertheless, the system state under the fault occurrence in Figure 3.8 remains in the correct state, performing the intended functions and without causing safety goal violations. Conversely, in 3.9, the amplitude of the fault results

in the termination of an intended function and the violation of safety objectives in safety-relevant systems. The faults described in the second case are designated as critical faults.

Definition 3.3 (Safe fault) *is a fault whose occurrence does not significantly increase the probability of a breach of a safety objective* [1].

In addition to the single faults mentioned above, the analysis of the system behavior in the case of simultaneous faults is considered to be a major challenge. In this case, two different types of faults can occur simultaneously at different locations. Identifying the causes and timing of concurrent faults based on certified tools or expert knowledge is extremely complicated. Figure 3.10 shows the system behavior under simultaneous fault occurrence, i.e., stuck-at and delay faults. Starting from time 170 sec, two factors cause the deviation and contribute to the fault occurrence at the system level.

Finally, the uncertainty in the data due to the manual driving by the user plays a negative role in performing an accurate fault diagnosis during the validation process. As shown in Figure 3.11, the system behavior was captured based on manual virtual test drives. In this case, the tester is not bounded to prespecified test scenarios to detect unexpected unknown faults. This requires in-depth knowledge of the recordings inspection process, which adds to the difficulty of fault diagnosis.

Definition 3.4 (Recording) *A recording represents discrete, time-stamped signal values that are recorded during the tests. A recording can contain several signals* [34].

From the recorded behavior to be analyzed, it is evident that the behavior is nonlinear, dynamic, and multimodal, lacking a clear set of rules. Moreover, the resulting behavior is shaped at the system level by a multitude of variables, all of which should be taken into account during the analysis process. In addition to the aforementioned challenges, the sheer volume of data samples generated by the test drives is considered to be another complicating factor in the analysis process. In this example, 400 seconds of driving time in three HIL sets, namely healthy, single fault, and simultaneous fault, generates 8,400,000 samples, with each set consisting of seven variables and 400,000 samples. Consequently, the greater the number and duration of driving cycles, the greater the amount of data to be analyzed. For instance, for 100 records and one hour of driving, the analysis process generates 2.5 billion data samples. In addition to the time and effort required, this necessitates the input of an expert with sufficient knowledge over the course of several days. It is also important to note that the manual analysis process is further complicated by the presence of

unknown fault types and the simultaneous occurrence of faults.

In conclusion, the conventional approach to failure analysis, which relies on Diagnostic Trouble Codes (DTCs) to identify specific faults, is not an optimal solution. DTCs lack the capacity to provide detailed contextual information, which hinders their efficiency. Moreover, in the context of complex systems, the necessity for expertise and an understanding of the vehicle system structure and functions, coupled with the necessity for manual inspections to interpret the results, represents an additional challenge. The greater the extent of records of complex system behavior, the greater the cost, effort, time, and difficulty of the manual knowledge-based analysis process by experts. Furthermore, the certified industrial tools are unable to accurately identify critical faults with reasonable interpretation. Consequently, in order to overcome the shortcomings of the aforementioned approaches, it is necessary to develop an intelligent strategy based on historical data for the analysis of virtual test drive recordings during the development of ASSs.

3.4 Challenges of Current Failure Analysis Methods during Validation Process

In accordance with the fundamental approach to system integration testing outlined in the previous sections, this section examines and elucidate the current challenges associated with state-of-the-art methods. In particular, the current limitations of the traditional failure analysis process used in the industry are elucidated. The current techniques and methods for FDD in the literature are then discussed. Finally, five challenges to be overcome in this work are presented.

The methodology of knowledge-based analysis is predicated upon the expertise and a profound comprehension of the target system. This enables the analysis of the captured system behavior and the identification of faults. In other words, the subjective evaluations are conducted by experienced test drivers and engineers. In particular, the test drivers identify faulty driving behavior during the test drive, and the test engineer manually reviews the recorded data after the test drive. This process is described in detail in [34]. In order to achieve this objective, data processing techniques such as data visualization, statistical analysis, and anomaly detection algorithms are employed. Nevertheless, although this approach plays an important role in the analysis of the records, it is associated with significant drawbacks. As the conclusions and interpretations of the evaluation are contingent

upon human experience, the process is susceptible to fault in certain instances and may result in inconsistencies in fault detection. Moreover, in modern complex systems, multivariate, multimodal dynamic behavior is captured, involving several hundred sensors, actuators, and controller signals, as well as system parameters and environmental conditions [169]. Consequently, the vast quantity of uncertain data renders knowledge-based manual inspection a challenging, time-consuming, and resource-intensive endeavour. Moreover, comprehensive coverage of the occurrence of all potential faults is not feasible, particularly in the case of simultaneous fault occurrences.

Over the past decade, the advent of more powerful computational resources has facilitated the widespread application of a data-driven approach to the problem of FDD. At the core of this approach lies the concept of extracting knowledge from historical data and constructing nonlinear relationships between input and output classes in order to uncover hidden patterns. In the current approaches, several DL architectures have been proposed to solve various application problems in different domains [170, 171]. These include AE, GAN, CNN, DBN and RNN, which have already been the subject of numerous studies. Depending on the task to be solved, the methods are divided into anomaly/fault detection, fault classification, fault clustering and fault regression. For instance, [172, 173] explored the potential of various AE variants as an unsupervised anomaly detection method for the in-vehicle CAN.

To identify the type of faults and the faulty component of the system, several frameworks have been proposed using the variants of RNN, i.e., GRU and LSTM, as the classification problem [174, 175]. CNN techniques with one and two dimensions have been used to diagnose faults in various vehicle systems, for example in vehicle dampers [176] and vehicle engines ([177]. Fault clustering is the process of discovering the hidden relationship between unlabeled data and grouping the data samples into multiple clusters. To accomplish this task, ML algorithms, such as K-means, are typically combined with DL methods, as proposed in [178]. Despite the observable performance of current intelligent approaches for FDD tasks, the developed FDD models still have several aspects that require further investigation and exploration. The main challenges addressed in this work are outlined below:

1. One of the most crucial aspects of the FDD model's development is the dataset. In theory, fault-free data can be obtained by recording the system's behavior in an fault-free mode [179]. However, obtaining representative data that accurately captures the system's reactions under

fault conditions is challenging in terms of difficulty and cost [180]. In the automotive sector, publicly available real-world data containing faulty behavior is rarely available due to the confidentiality of test data [181]. Moreover, even with limited data availability, the ratio of faulty to healthy data is low and unstable, which in turn leads to the problem of unbalanced data. One reason for this is the difficulty of simulating the open set of fault modes under complex working conditions [182]. Furthermore, the implementation of hardware faults in real applications is not feasible [183]. A multitude of studies have been conducted in the literature to investigate FI methods at the model level. The objective was to analyze and capture the system behavior under fault conditions and to collect the corresponding dataset [184]. However, the majority of previous studies do not take into account the real-time constraints. Moreover, the studies do not consider the impact of modifying the system model during the implementation of the fault mode on the real-time requirements of the control task. The execution of the additional blocks necessitates additional time, which affects the real-time performance of the control task [185]. The corresponding research question of the mentioned challenge is formulated in RQ1.

2. A further challenge in the development of an FDD model for ASSs is the multitude of independent techniques on which the majority of proposed studies are based. However, it has not been demonstrated that a single DL technique is sufficient for all FDD requirements. Consequently, there is currently no standard for selecting an appropriate technique from the extensive range of available DL algorithms to develop a robust and accurate model. In recent years, several studies have been conducted to demonstrate that hybrid methods are superior to stand-alone methods. Additionally, extensive efforts have been made to investigate FDD based on hybrid DL methods [186, 187, 188]. To achieve optimal performance of FDD in a complex system, combining different DL techniques as a hybrid method represents a comparatively efficient solution. In this way, the advantages of each method can be utilized while circumventing the limitations. Consequently, research on developing an FDD model based on hybrid DL methods is still at the initial stage and requires further contributions in this field (RQ2).

3. Considering the realistic operating conditions of the developed target system with noisy data,

analyzing the test recordings to detect and identify unknown faults, both single and simultaneous faults, is a challenge. In the models currently presented in the literature, only the known fault patterns in the target system are considered to solve the FDD problem. This approach results in the development of a model that cannot detect unforeseen faults or misidentifies the detected faults. This limitation was highlighted in [181]. Consequently, developing a novel method to improve the applicability and robustness of FDD algorithms for unknown faults in the presence of noise is a major challenge that needs to be addressed. The research question derived from this challenge is presented in RQ3.

4. The current FDD models consider a single fault without accounting for the potential occurrence of simultaneous faults. In a complex system, however, a combination of several independent faults can occur simultaneously at different points, leading to multi-point faults [189]. Furthermore, the dataset is collected under the real operating conditions of the complex system, i.e., under noisy and unbalanced data [35, 190]. However, the majority of FDD model development studies have been conducted with noise-free experimental data under normal conditions. Therefore, the fourth challenge is to develop an intelligent solution that addresses the problem of detecting and identifying simultaneous faults in noisy and unbalanced data. This is discussed in greater detail in Section RQ4.

5. The aforementioned challenges must be overcome in order to develop a robust FDD model for the validation process of ASSs. Once the target model is developed, it must be implemented and integrated into the testing process and environment. The final challenge in this thesis is to develop a real-time virtual test framework that takes into account the user behavior and abnormal state of the systems to evaluate the developed FDD models. The detailed description of this challenge is given in section RQ5.

In light of the aforementioned challenges, this thesis proposes a ML-based FDD approach that leverages historical datasets generated through real-time FI. Besides, the generalizability and scalability of the proposed work is validated by developed real-time virtual test drive based on HIL simulation. The main aims of the proposed research can be summarised as follows:

- Develop a generic framework as a means to capture and analyse the system behavior under

random and unpredicted fault conditions that may occur in the sensors and actuator components.

- Generate a representative faults dataset considering real-time conditions and most types of faults that could occur in the sensors individually and simultaneously.
- Develop a data-driven intelligent FDD of single and simultaneous faults under noisy and unbalanced data conditions to be used during system integration testing of ASSs.
- Test and validate the developed methodology in terms of reliability and applicability using a real-time virtual test drive framework and various case studies.

3.5 Research Questions

In Section 3.4, five different challenges that arise from current failure analysis approaches are discussed and analysed. Based on the mentioned challenges, five research questions to be addressed in this study are derived and formulated as the following:

RQ1. How can the system behavior under faulty conditions be captured in real-time without changing the system architecture to generate representative dataset?

Overcoming the challenge of the unavailability of a representative dataset is still an open issue, and the collection of fault data should be further explored. In this study, we attempt to bridge this gap by proposing a novel FI framework for analysing the system behavior under faults and collecting a representative real-time dataset, including faulty samples of critical faults. The real-time representative dataset generated, including critical faults, is the input for the next research questions. This research question will be addressed in Chapter 6.

RQ2. How can the diagnosis model of single sensor faults be improved while maintaining comprehensive class coverage?

The aim of the RQ2 is to investigate the effectiveness of hybrid DL algorithms in developing an intelligent FDD that can improve the accuracy compared to single stand-alone methods. It aims to detect and identify the faults at system-level, as a classification problem, in sensors

and communication signals during the HIL test process. This will overcome the limitations of individual methods and exploit their advantages. Chapter 7 discusses and answers this research question in detail.

RQ3. How can a robust model be developed for diagnosing unknown simultaneous sensor faults under noisy conditions?

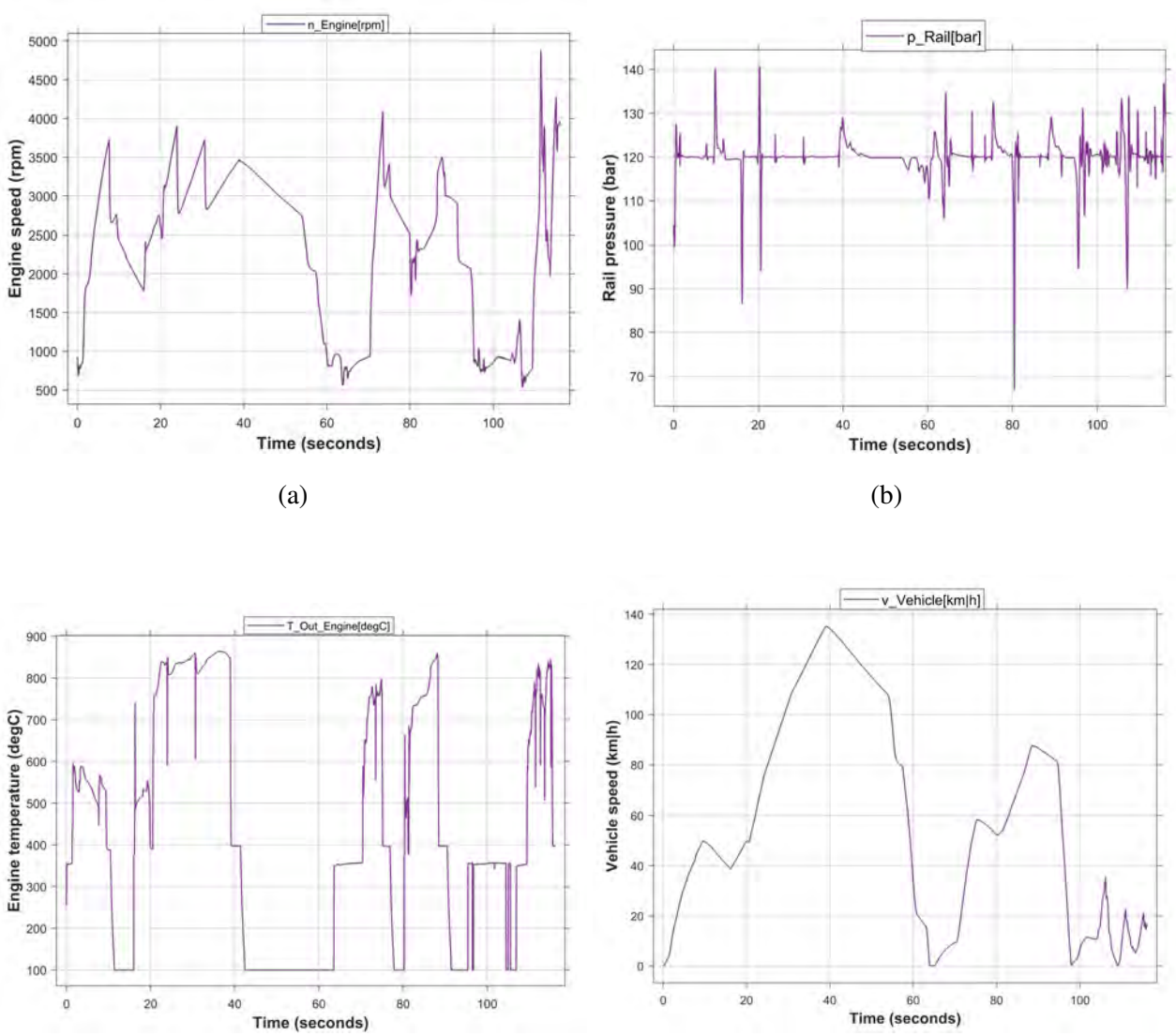
This research question will be addressed in this study by developing a novel intelligent method for unknown fault detection and clustering during the development process of ASSs. Specifically, the objective is to develop a DL-based model capable of detecting and clustering concurrent unknown faults during the system integration and test phase with real-time HIL simulation tests. This RQ will be discussed and addressed in Chapter 8.

RQ4. How can a known simultaneous FDD model be developed for the recordings analysis considering the noisy and imbalanced data?

To fill the gap in the literature on concurrent FDD of ASSs, this study aims to develop a novel model based on the ensemble DL approach considering the noisy and imbalanced data conditions. In this study, the applicability of the ensemble learning based classifier for FDD under unbalanced and noisy conditions will be investigated. The objective of the target model is to facilitate the automated analysis of test drive records, thereby enhancing the real-time validation of ASSs throughout the V-cycle development process. This research question is discussed and answered in detail in Chapter 9.

RQ5. How can a demonstration prototype be developed to evaluate the SUT and validate the proposed concept based on real-time HIL simulations?

This research question will be addressed through the development of a real-time virtual test drive framework (Chapter 10). The aim is to develop a case study to validate the developed models in research questions RQ2, RQ3 and RQ4. As evaluation metrics, quantitative evaluation measures, i.e., precision, recall and F1-score, are used to assess the performance of the proposed model of RQ2 and RQ4. On the other hand, for unsupervised learning with unlabelled data, the Mean Square Error (MSE) and Davies-Bouldin (DB) score are used to evaluate the developed model of RQ3.



(a) (b)

(c) (d)

Figure 3.11: System behavior with manual test drives. (**a**) Engine speed (**b**) Rail pressure (**c**) Engine temperature (**d**) Vehicle speed

4 Related Work

In the previous chapter, the problem was analyzed and the corresponding research questions were formulated. The goal of this chapter is to present the state-of-the-art methods related to this work in the field of FDD. In particular, this chapter focuses on describing the main contributions and drawbacks of related work and highlighting the differences of our proposed methodology compared to the state-of-the-art. In order to achieve this objective, the chapter is divided into several sections, each of which corresponds to a research question defined in Section 3.5. Section 4.1 discusses related work that addresses the problem of analyzing system behavior under faults and generating representative faults dataset (RQ1). Section 4.2 presents the related approaches for detecting and classifying individual faults (RQ2). Similarly, Sections 4.3 and 4.4 discuss the main contributions and limitations in the area of fault diagnosis for unknown and known concurrent faults, respectively (RQ3 and RQ4). Section 4.5 provides an overview of the related work, highlighting the main contributions and limitations of current test platforms (RQ5). Finally, Section 4.6 concludes the chapter with a summary.

4.1 Research related to Fault Effect Analysis and Representative Dataset Generation

4.1.1 Faults effect analysis in the automotive software systems

Model-based techniques for the analysis, verification, and validation of system behavior in the early stages of system development have attracted significant attention from academic and industrial researchers [191]. One of the earliest applications of a FI methodology that uses models to simulate hardware fault behavior and analyze the system in the presence of faults is described in [192, 193]. The executable model of the system was run using existing commercial tools such as Simulink or SCADE. In the work of Moradi et al. in [185], a model-implemented hybrid tool is presented

that uses Simulink as a modeling environment to combine the advantages of hardware-based and software-based FI. The proposed framework is limited to injecting one fault in each model. In order to guarantee the real-time behavior of the system, additional slacking parameters are required, which must be determined manually. These define the slacking time and the number of additional blocks to be added to the system model. Our framework allows the injection of single and multiple faults, which also enables compound and simultaneous faults. Moreover, the faults in our study are injected programmatically without changing the original system model.

In the automotive industry, numerous authors have developed FI approaches for evaluating safety objectives and safety mechanisms in accordance with ISO 26262 during the early design phases [194, 195]. FI-based test simulation framework, designated as MOBATsim, was presented by Saraoglu et al. in [196]. The framework is based on Simulink behavioral models and permits the injection of hardware faults into the model, thereby enabling the evaluation of the safety of autonomous driving systems at different levels, including component, vehicle, and traffic levels. However, the proposed simulation framework is not designed to consider the real-time characteristics of the control task at the simulation level. In contrast, our FI framework is implemented in the real-time simulation platform, which enables accurate real-time analysis. In [197], Juez et al. investigated the applicability of a simulation-based FI framework, called Sabotage, using the Dyanacar vehicle simulator. The proposed work aimed to determine the most appropriate safety concept and to conduct an early safety assessment of a vehicle lateral control system in accordance with ISO 26262 at the simulation level. Although the authors conducted a comprehensive safety analysis of the entire vehicle system, they incorporated model blocks into the system model to represent potential failure modes. However, this approach is not effective for complex systems and may result in a violation of real-time system performance. The proposed framework is based on the programmatic manipulation of sensor signals, thereby ensuring that the system remains in real-time.

Beyond the aforementioned work, several publications over the past few years document model-driven FI tools for safety and reliability assessment of ASSs, like Kayotee [198], ErrorSim [199], AVFI [200], FIEEV [201], SIMULTATE [202], and EQUITAS [47]. While a substantial body of research has been conducted to develop simulation-level FI methods and tools for a variety of domains, much of this research has encountered difficulties in accurately representing fault effects under real-time constraints. In our research, our proposed framework provides comprehensive fault

coverage along with highly accurate simulated behavior for complex system behavior analysis.

Although HIL simulation has been used for the design and development of new automotive control units [93], academic researchers have put a considerable effort into investigating the development of automotive control software based on HIL. For example, a portable electronic environment system called the micro-HIL system has been proposed by Palladino et al. in [203]. The study evaluates strategies for the engine control software using the Fiat 1.6-liter petrol engine. In addition, the functions of the engine control software are diagnosed on a CAN bus. A new concept for developing ADAS based on vehicle HIL simulation is proposed in [204]. In the field of railroads, Conti et al. in [205] analyze the railroad braking system under degraded adhesion conditions based on the HIL approach. They emphasize the advantages of the proposed approach in terms of test cost and repeatability, especially for the analysis of system behavior under normal and degraded adhesion conditions during railroad braking.

The analysis of the behavior of complex ASSs under abnormal conditions has attracted considerable attention in the last decade, along with the advances in HIL real-time simulation for embedded control development and automated testing. In the existing literature, a number of methods have been proposed for addressing this problem. For instance, Poon et al. in [206] conducted a study to demonstrate the capability of a HIL platform with FI for testing electric vehicles (EVs) drive systems. In the proposed study, the fidelity of the real-time simulation was validated by comparing the real EV drive system and the real-time simulation under three different operating and fault conditions. However, the focus of our study is on the development and design of an effective real-time FI framework with high fault coverage for complex software system analysis. In the mentioned study, FI is limited to specific fault modes in the drive systems and is used to validate the fidelity of the proposed HIL platform. Yang et al. in [207] proposed a multiprocessor HIL FI strategy for the simulation of various faults in a traction control system (TCS). Three fault scenarios are employed in the HIL simulation, i.e., an open-switch fault of the power transistor, a stuck fault of the three-phase current sensor and a broken rotor bar fault of the traction motor. Although the proposed FI method was developed with the assistance of a physical traction control unit (TCU) and a real-time simulator, the FI device was designed on an FPGA based on logical operators to satisfy the time restriction. However, to inject the faults, our framework is based on the manipulation of signals accessed on the CAN bus according to the user's specifications in terms of location,

time, and type. In a similar context, an enhancement of the FMEA methodology through the integration of a HIL approach has been proposed in [208] to the evaluation of risks in railway traction drives and the analysis of their behavior. The objective of the proposed research is to enhance the FMEA methodology in order to facilitate a quantitative analysis using FI, with the aim of generating fault modes. However, for the implementation of faults modes, the system model has been extended, which in turn has an impact on the real-time behavior of the system. In our framework, this issue has been addressed by treating both the plant model and the control system model as a black box without modification. In the context of automotive sensor networks, Elgharbawy et al. in [209] proposed a real-time functional robustness verification framework for multi-sensor data fusion algorithms applied to radar and camera sensors in ADAS. In order to simulate sensor faults, such as latency, detection faults, and incorrect one-to-many object labeling, HIL co-simulation with runtime-implemented FI was employed. The study is limited to the investigation of certain critical driving situations by focusing only on image and distance sensors to verify the robustness of the fusion algorithms. Whereas in our framework, all sensor signals accessible via the CAN bus can be manipulated, thus increasing the coverage of fault locations for the analysis objectives.

Other areas where the FI approach can be applied include online condition monitoring and fault diagnosis in a real-time environment based on HIL simulation. For instance, recent research in [210] proposes the use of a short-circuit fault model to facilitate online switching between healthy and fault states of induction machines. This approach is intended to enable the determination of the trend of change in fault properties and fault severity. Although the modeling of the fault source in the proposed work reduces the modeling complexity, it is limited to one fault mode, specifically the stator interturn short-circuit fault. This fault is triggered through changing the asynchronous motor parameters and changing the operating conditions. In addition to the healthy condition, our proposed framework is also capable of implementing nine types of faults in the online simulation. Garramiola et al. in [211] have employed the FI approach to develop a hybrid methodology for diagnosing sensor faults in railway traction drives using the HIL platform. This is accomplished by introducing gain and offset sensor faults into the DC link voltage and catenary current sensors through the use of an FI signal, thereby enabling the analysis of the dynamic response, robustness to drift, and sensitivity of the fault reconstructions. In the aforementioned study, the FI approach was investigated with regard to its potential for use in the development and verification of fault

diagnosis systems. This differs from the subject of our study, which focuses on the use of the FI approach in real-time experimentation. Nevertheless, our study focuses on the development of real-time FI as an experimental technique during the system development stages.

The automotive community has recently become increasingly interested in safety verification during design at the component and system levels. This is due to the importance of such verification in ensuring safety properties and identifying safety faults. A number of researchers have proposed the use of FI frameworks and tools as a means of verifying the functional safety of vehicles in order to address this issue. For instance, a retargetable FI framework at the vehicle level has been proposed in [212], which is capable of automatically injecting various faults into the processor, memory, or I/O at run time. The proposed framework has been validated and demonstrated using an experimental HIL test for autonomous driving, namely EcoTwin truck platooning. In contrast to our proposed framework, our framework is based on signal modification as the foundation for FI, whereas this framework has been developed using a software-based FI approach. Furthermore, the target components for FI in our framework are sensors and actuator control signals, whereas the target components of the SUT in the mentioned study are processor registers, memory, input/output (IO), and operating system (OS) kernel. In addition, Park et al. in [213] have proposed an FI method for software (SW) unit/integration testing during the ECU software development process of automotive software based on the Automotive Open System Architecture (AUTOSAR). In the proposed study, potential software faults in AUTOSAR-based automotive software, including data faults, program flow faults, access faults, asymmetric faults, and timing faults, were defined and injected using the proposed tool. This study demonstrates that our proposed approach can be applied to real automotive applications and compares its performance with other FI tools. Although there are significant differences between our proposed framework and the existing research, the proposed research analyzes and compares various aspects of functional integration testing in the software unit/integration test phase. In addition, hardware faults occurring in the sensor and actuator control signals can be injected into our proposed framework. Furthermore, to enable effective and accurate system-level testing, the entire vehicle system model was considered in our study.

4.1.2 Representative real-time dataset generation under critical faults

A number of potential solutions have already been proposed in the scientific literature to address the issue of the unavailability of fault datasets. Some of the proposed solutions are based on publicly available online datasets, while others require the creation of data in accordance with the specifications of a real prototype or simulation system.

A standard gas turbine dataset provided by NASA was used by Rengasamy et al. in [174]. The dataset is employed for the development of a DL-based model for prediction and diagnosis tasks. Similarly, an intelligent model for fault detection, isolation, identification, and prediction based on DL methods was developed in [214] using an industrial dataset provided by Audi AG. The objective of this research is to develop an autonomous driving system. The classification of faults depends on the source and cannot be flexibly extended under identical operating conditions. However, the use of published standard data is advantageous. An alternative approach would be to statistically inject the faulty sample into the data based on normal distributions with one standard deviation. Other researchers have opted to use a real prototype. For instance, in [215], a real engine was utilized to capture the system's behavior under ordinary and irregular circumstances with the objective of developing a dependable FDD for automotive engines. The developed model demonstrated a high degree of applicability and robustness to environmental conditions such as noise, as evidenced by the close alignment between the collected data and the real industrial application data. The main significant limitation is the potential for harm to the target physical device if a fault is introduced. Yang et al. in [216] proposed an EG-DEG device, which is a self-powered sensor that generates electrical signals, to record the flow of fluids. An intelligent particle detection and fluid analysis system was developed based on the generated data. Similarly, for IoT applications, Alsaedi et al. in [217] proposed a new representative IoT/IIoT dataset to be used for intrusion detection system training and evaluation. The proposed medium-scale dataset testbed enables the generation and capture of new data features, namely IoT/IIoT telemetry data, operating system data, and network data.

A dataset comprising sensor-related data was collected and used to develop a DL-based FDD system for hydraulic systems, as detailed in [218]. This system was developed using a hydraulic machine test bench. In order to provide a representative sample of faulty data, three types of sensor

faults, namely constant, gain, and base, were injected into the collected dataset. The generated dataset lacks the necessary context to reflect the working environment in real-time. Furthermore, the dataset does not provide a scenario in which multiple faults are present simultaneously. In a similar context, Bafroui et al. in [219] proposed a FDD model for a vehicle transmission system, based on data collected from a laboratory experimental test bench. Additionally, several studies have been conducted to develop an intelligent solution for detecting unknown faults in test drive recordings. These studies are based on data collected from real vehicle prototypes [158]. However, such experiments are not only costly but also pose a risk to the driver and the test equipment. Moreover, some faults are not amenable to investigation, which may result in the entire physical system being at risk.

A simulation platform was employed to generate representative datasets, thus overcoming the limitations of using real physical systems. A number of researchers have proposed solutions to the problem of data unavailability based on modeling and simulation techniques. An example of applying the FI method with a simulated system in the MATLAB/Simulink environment to generate a faulty dataset can be found in [220]. In the aforementioned work, four driving scenarios were carried out to generate faulty data in addition to the normal data. Similarly, Biddle et al. in [221] addressed the problem of the unavailability of faulty data in their work. To do this, they injected five types of faults into the simulated system: erratic, hard overshoot, drift, spike, and stuck. The simulation was conducted by modeling the fault modes within the simulation environment and subsequently introducing the faults through the injection of the fault parameters. Consequently, the data collected was used to create an ML-based multi-faults architecture for the multi-sensor FDD. Nevertheless, the real-time conditions are not taken into account, despite the capacity to reproduce the test under critical conditions with a high degree of confidence. Consequently, the practical applicability of the target model is becoming increasingly constrained. Moreover, the simulation data generated by a simulation platform is employed to validate the currently developed FDD models. This approach fails to account for real-world conditions such as external influences, noise, and uncertainties. In the context of autonomous driving, the Carla simulator has been employed to generate object detection datasets, as evidenced in [222]. The datasets include realistic driving images. In order to enhance the efficacy of the developed intelligent object detection system, the principal characteristic of the generated datasets is the incorporation of abnormal weather and lighting

conditions. However, the focus of the generated datasets is on images and does not include faulty sensor readings or system behavior under actuator faults. A visual simulation system, designated TrainSim, has been proposed as a means of generating synthetic datasets for the development of DL-based models within the railroad sector [223]. A set of simulated sensors is employed to generate synthetic datasets comprising a diverse array of realistic railway scenarios accompanied by labelled data. In the course of the work, cameras, LiDARs, and inertial measurement units have been employed in order to generate a multitude of labelled images. However, the potential of time series data has not been fully exploited.

To overcome the limitations of pure simulation, real-time HIL simulation was introduced as a solution to generate a dataset covering different critical conditions. This approach is discussed in detail in [224]. An example of the application of intelligent fault detection models for vehicle air brake systems is the development described in [225]. The data collection is conducted using a HIL platform and encompasses real-time simulation data generated by executing a virtual test drive under normal conditions. The dataset comprises the wheel speed of the four wheel. The present study is confined to the investigation of two single-fault scenarios, yet the realistic vehicle operating conditions resulting from real-time constraints are accounted for. In [226], two FI simulation platforms were utilized to implement the FDD strategy for high-speed traction systems. The proposed platform considered six types of faults, injecting two types at three points: sensing, traction motors, and traction inverters. Notwithstanding the considerable outcomes achieved by the proposed framework in comparison to the state-of-the-art, the dataset of the work is constrained to specific types of faults, with a focus on the system operation under the individual faults. Similarly, in [227], it has been developed an integral diagnostic strategy for electric drives in railroad applications based on a sensor-related fault dataset generated from a HIL simulation. In order to capture the system behavior under fault conditions, the original system model is extended with additional simulation blocks, which simulate the fault modes. This may result in the violation of the SUT's real-time constraints. Furthermore, the manual preparation of TCs and scenarios for the detection of abnormal system states is a time-consuming process, and the more complex the system architecture, the more time is needed to prepare TCs and scenarios. Modeling faults as described in [228] requires more time and effort as the system architecture becomes more complex.

This thesis presents representative real-time datasets of ASS based on a FI framework and a

HIL simulator. While the currently available datasets for ML-based FDD neither represent real environments nor fulfill the time constraints. The shortcomings of datasets with balanced classes have been addressed by incorporating transient faults, which result in imbalanced data, to reflect the actual operational conditions.

4.2 Related Work to FDD of the Single Faults

Several studies have been conducted with the objective of developing a FDD model based on the collected representative datasets. These studies have proposed the use of either real-world or simulation platforms to collect measurement data in different system operation modes, including both faulty and non-faulty modes. Safavi et al. in [229], for instance, have proposed a fault detection, isolation, identification, and prediction system based on DL algorithms for sensor faults in autonomous vehicle systems. In the study, the real-world data from three sensors, i.e., the accelerator pedal, the steering wheel angle and the brake pedal, collected and recorded in a CAN bus on highways were used to train, test, and validate the proposed FDD system. In particular, the collected autonomous driving dataset contains about 35 min of real driving scenarios, namely city and freeway, at three different recording sites: Ingolstadt, Munich and Gaimersheim. A faulty dataset is generated by capturing the system behavior in the faulty mode using the FI method in addition to the healthy real dataset. The proposed fault detection system exhibited an accuracy rate of 99.84%, while the fault identification system demonstrated a range of 73.00% to 100.00%. In contrast, our proposed methodology considers eight different types of faults, thereby increasing the coverage of faults in both the sensor and the communication. Another distinction pertains to the methodology employed in the FI to gather the fault data. This study examines the impact of faults on system performance, taking into account real-time constraints and the simulation of the entire vehicle system. Similarly, in [230], the author presents an anomaly detection approach that involves the use of an ensemble classifier to analyze a test recording of a vehicle. The analysis is based on the data recorded during a test drive on the road. Although the proposed system is capable of effectively detecting both known and unknown faults, which are robust to different fault types and driving scenarios, it is limited to a detection problem with only two classes, namely binary classification of inputs into healthy or faulty. In contrast, our proposed method goes beyond the detection task

to investigate the identification of the type of recognized faults as a classification problem with nine different classes (healthy and faulty). Furthermore, the utilization of real data from a vehicle prototype as the foundation for the training dataset is costly and carries a significant risk for the driver, particularly if faults are introduced during operation for the validation of the proposed system. In contrast, our proposed method addresses this issue through the use of HIL simulation with real-time FI.

In order to overcome the limitations of the use of a real vehicle prototype, some researchers have proposed the use of real automotive system test benches as a solution for the acquisition of training datasets. For instance, in [231], a probabilistic fault classification method for time-series data was proposed for the classification of unknown faults. A combination of Weibull-calibrated OSVM and Bayesian filtering was employed for the purposes of fault class modeling and fault classification, respectively. To illustrate the efficacy of the proposed methodology, an internal combustion engine was utilized as a case study. In parallel, a residual dataset was generated using a real engine test bench, which was modelled to encompass seven distinct types of engine faults, including sensor faults, leaks, and air filter blockage. However, our proposed method employs a real-time simulation platform (HIL system) to inject faults and capture system-level effects. In contrast, a highly accurate mathematical model is required to operate in a realistic environment, which increases the cost in terms of complexity. In a similar context, Kaplan et al. in [232] employed an EV prototype to evaluate their proposed LSTM-based FDD approach. In our research, this problem is addressed by injecting the fault in real-time using a HIL system. To address both single and simultaneous fault diagnosis problems of automotive engines, a novel framework has been proposed by Zhong et al. in [233] based on the combination of three phases: feature extraction, probabilistic rejection engine (PCM), and decision threshold optimization. A real sports car, comprising a control unit, an integrated oxygen sensor, and a microphone, was employed to record the dataset in a variety of fault conditions. Nevertheless, the use of high-fidelity simulation (HIL) with rapid control prototyping (RCP) was employed in this study to circumvent the prohibitive costs and risks associated with the collection of real data from a real vehicle.

Another method for obtaining a representative dataset of healthy and faulty operating modes is the use of a simulation platform. For instance, in [234], the researchers propose that the MATLAB/IPG CarMaker co-simulation platform can be utilized to model sensor faults and subsequently

collect the resulting dataset. This approach enables the development of an architecture for detecting, isolating, and identifying multiple faults in autonomous vehicles. Although the proposed FDD architecture with SVMs exhibited high accuracy (94.94% and 97.01%), the faults were introduced at the model level without consideration of real-time limitations.

Over the past decade, the capabilities of HIL simulation have attracted the attention of academic and industrial researchers alike. The capacity of the HIL platform to simulate a complex nonlinear system with high fidelity enables effective and accurate analysis, verification, and validation of an ASS in real-time, rather than the real hardware elements of the vehicle. Consequently, the utilisation of HIL systems is not only focused on the advancement of control software modules but also on the verification and validation of the developed systems [235]. The development of intelligent FDD of complex software systems has attracted considerable attention, extending beyond the advances in real-time simulation with HIL. As previously documented in the literature, a number of methodologies have been developed to address this issue. For instance, in [236], an integral sensor fault diagnosis method for railway traction drives was proposed using the HIL system. This was accomplished by developing a model that is capable of detecting, isolating, and evaluating sensor faults, including gain and offset faults, through the integration of model-based and data-driven techniques. In order to simulate the system's behavior under faults, the system model has been augmented with additional model blocks that affect the real-time properties of the control task. In contrast, in our proposed work, all the faults under study have been injected programmatically into the automotive sensor signals without any modification of the original system model. Furthermore, in our proposed method based on DL classification, nine distinct classes have been considered, whereas in the aforementioned study, the classification model is constrained to three classes and relies on manual threshold setting. In reference to the study by Raveendran et al. in [237], a multi-class classification model based on data obtained from the wheel speed sensor was proposed for the same platform. The aim of this model was to enable the identification of faults in air brake systems of heavy-duty vehicles. This model was developed using data obtained from HIL simulation and was evaluated on the aforementioned platform. In comparison to the RF-based model developed in the study, the model proposed by our research group, which was developed using hybrid DL techniques, demonstrated a superior classification performance relative to the traditional ML method. Furthermore, the methodology presented in this investigation relies on the comprehensive vehic-

ular system model operational within the HIL simulator and connected to the control system. To simulate the system behavior under fault conditions, a real four-wheel drive system was connected to an HIL simulator and data were collected during the test scenario. The data were collected to provide insight into the system behavior under fault conditions. Finally, an FDD method based on data-driven pattern recognition techniques, specifically SVM, has been proposed by Namburu et al. in [238]. The proposed method is focused on automotive engine faults. Eight engine faults with varying degrees of severity are inserted into the engine system model simulated in the CRAMAS real-time simulator. Although their proposed model exhibits high diagnostic accuracy with regard to the detection, isolation, and estimation of fault severity for a significant number of potential fault points, it is significantly distinct from our proposed method. The objective of the proposed study was to classify faults according to their type, rather than their location. In the study, fault instances were introduced as incremental modifications at various locations with the objective of replicating the operational functionality of the engine under different scenarios and fault conditions. Moreover, although our proposed study considered eight distinct fault types and employed a comprehensive vehicle model with high-fidelity simulation, the available data is limited to the evaluation of the presented approach without regard to fault types or realistic driving scenarios.

In summary, the conventional approach to acquiring representative dataset is to obtain the faulty dataset from the automotive industry. However, it is unlikely that manufacturers share their highly confidential faulty data. Conversely, real-world test drives offer comprehensive testing, yet the cost of testing escalates with the number of test miles. Consequently, this type of recorded dataset remains limited and expensive. Moreover, the probability of risk to the test driver increases in direct proportion to the definition of the relevant scenario and critical driving situation. The challenge is to gain access to confidential industrial data and to obtain fault data from real test drives.

The novel aspect of this study is the development of a FDD methodology for HIL real-time testing of an ASS during development, prior to operational implementation. In particular, this methodology is designed to be used in the system integration test phase, which is an essential stage of the overall development process of an ASSs. The study encompasses the entirety of the vehicle system models. A representative fault dataset is generated by programmatically injecting nine different fault types into the HIL system in real-time, without modifying the original system model. A novel intelligent FDD model based on hybrid DL techniques, namely LSTM and CNN,

is developed through this approach. The selected techniques were selected for three main reasons: their ability to process large amounts of data, the automatic extraction of features from multivariate time series data, and their capacity to develop a precise fault detection model with the incorporation of prior knowledge. In addition, the expenditure incurred in data acquisition is diminished, high-fidelity simulation is ensured, and a model is constructed for the analysis of test drive data at an elevated degree of accuracy by utilising data from an real-time FI within the context of a HIL simulation, while taking into consideration real-time constraints and a comprehensive vehicle system model.

4.3 Related Approaches Tackling FDD Problem of Unknown Faults

In the contemporary manufacturing and industrial sectors, the deployment of ML-based FDD is a crucial strategy for enhancing reliability and ensuring safety. Nevertheless, the most significant obstacles in the development of FDD models are the scarcity of labelled data and the occurrence of previously unseen faults.

The lack of labeled data has motivated researchers to explore the possibilities of learning features from unlabeled data using unsupervised DL methods. Among these techniques, AE was introduced as a powerful method for reducing the dimensionality of the data and for extracting and learning features from unlabeled data. This approach was first proposed by Shao et al. in [239]. Consequently, several research works have been conducted based on historical time series data [240] to investigate the applicability of AE for FDD in various applications. For instance, a novel network architecture based on a hybrid LSTM-based AE and a 1D-CNN was proposed by Mallak et al. in [218]. In order to accurately detect and classify the unprecedented faults of sensors and components in hydraulic systems, the study focused on unsupervised and supervised learning methods. The experimental results demonstrated that the use of pearson autocorrelation can yield superior outcomes in calculating signal difference, thereby enhancing the detection performance. The study considered a balanced dataset derived from hydraulic test benches with three distinct fault types: stuck-at, gain, and offset. In the same field, Wang et al. in [241] performed a behavior analysis of two DL-based FDD methods, namely, independent CNN and AE-based DNN with SoftMax

classifier. This study sought to identify an optimal DL architecture that would provide high classification accuracy while reducing computational time without the need for data pre-processing. To develop and validate the FDD model, a simulated MMC HVDC transmission network under seven conditions was employed as a case study. The experimental results indicate that the DNN based on AE exhibits superior accuracy compared to the CNN, while the CNN requires less training and testing time

To address the challenge of identifying unknown faults, Han et al. in [242] proposed an intelligent rotating machinery diagnostic model based on out-of-distribution (OOD) detection-assisted reliability. To guarantee the dependability and safety of the model in the face of unforeseen faults, the innovation of this work lies in the creation of an ensemble of five individual deep-based learners with reliable analysis. In this manner, uncertainties can be identified during the decision-making process, thus avoiding the generation of untrustworthy diagnoses in practical applications. To validate the proposed approach, two case studies are considered: wind turbine and gearbox systems. In a similar context, Park et al. in [243] propose a novel robust CNN-based diagnostic model for the reliable operation of permanent magnet synchronous motors (PMSMs) under variable conditions. The research revolves around the concept of transforming fault-related information present in the motor current signals into 2D instantaneous residual current images. In light of the rapid development of autonomous vehicles and ADAS, there has been a significant focus on the use of ML techniques to develop FDD for such systems. In the automotive domain, the historical test recordings of real-world test drives have been used to develop a robust ML-based model that is capable of detecting both known and unknown faults under different driving scenarios, as demonstrated in [158]. In order to ensure the model's robustness in the face of data variability across different fault types, a ML classification approach was employed. To validate the proposed methodology, road test recordings were utilized in which faults were introduced during the driving process.

In addition to single faults, the simultaneous occurrence of multiple single faults is one of the most significant complicating factors in the FDD process, as evidenced by Wong et al. in [244]. Furthermore, in multivariate systems, the acquisition of data containing representative simultaneous faults with a large number of possible combinations remains a significant challenge, as evidenced by the findings of Chang and colleagues [245]. Consequently, the development of FDD strategies to overcome this challenge has received considerable attention in various fields over the past decade.

In this context, Zhong et al. in [233] proposed a probabilistic committee machine (PCM)-based intelligent concurrent FDD framework for automotive engines. The proposed strategy was trained and validated using a real automotive motor. A total of ten types of single faults and four reasonable combinations of concurrent faults were explored, with the data derived from three real engine signals. In consideration of the noisy operational process, a total of 2,000 and 800 samples of single faults and simultaneous faults, respectively, were utilized. The proposed method demonstrated notable success in terms of FDD performance when compared with single probabilistic classifiers. Nevertheless, by examining the potential for other techniques to enhance the accuracy of the results, it may be possible to achieve further improvements.

Likewise, considerable effort has been invested in developing methodologies and frameworks for concurrent fault detection in the domain of building management systems, with a particular focus on HVAC systems. For instance, hybrid classification chains based on multiple labels with RF can be successfully employed to detect and categorize single and simultaneous faults with an accuracy of 99.5%, as demonstrated by Wu et al. in [246]. In particular, the study focused on the actuator, wherein the HVAC system was considered to have six individual faults and seven concurrent faults. Although the proposed method demonstrates superior performance relative to other methods, it ignores the noise conditions and time constraints. In [247], the authors argue that the problem of multi-class classification of concurrent faults can be addressed using a multi-label DL network, specifically a stacked sparse autoencoder, in the context of a solid oxide fuel cell system. The proposed method's distinctive feature is that it does not necessitate the acquisition of independent and simultaneous fault samples. In contrast, the feature extraction process is automated in both normal and unknown conditions. Nevertheless, the study is constrained to four potential combinations of simultaneous faults using non-real-time simulation data, despite the demonstrated superiority of the proposed model for independent and simultaneous faults.

In a recent publication, Liang et al. in [248] proposed an innovative approach to the simultaneous FDD problem by integrating GAN and continuous wavelet transform. The data is transformed into two-dimensional time-frequency images, and a target model is constructed based on adversarial learning. The semi-supervised approach enabled the use of a limited number of labeled samples to address the challenge of the unavailability of representative faulty data. Moreover, the impact of the training data on the evolution of the model, specifically the number of labeled samples and

the image size, was investigated using a real dataset from a transmission test platform. The efficacy of the proposed system was evaluated under 15 distinct conditions and compared to other related works. The results demonstrated that the classification accuracy of 99.16% on unlabeled data represents an enhancement over the existing FDD methods documented in the literature.

The current body of research on simultaneous FDD for ASS development is inadequate, as evidenced by the aforementioned studies. In contrast to our proposal, existing models for simultaneous FDD do not take account of real-time conditions or the effect of noise in the training dataset. Furthermore, the current classification approach is limited to specific target system conditions. This study addresses the aforementioned shortcomings by developing an innovative hybrid DL methodology. This methodology is capable of detecting and clustering both single and simultaneous faults during the system integration phase of the V-model. This is accomplished through the utilization of real-time HIL simulation.

4.4 Related Work to Detecting and Classifying the Concurrent Faults

4.4.1 Fault detection and diagnosis in automotive domain

In recent years, with the advent of more sophisticated ASSs designs, FDD strategies have gained increasing popularity in research. Consequently, a plethora of methodologies have been developed within the automotive domain for the purpose of fault detection, isolation, identification, and prediction of sensor states.

Jiménez et al. in [249] proposed a scheme for the detection and isolation of a faulty fuel injector. The proposed ANN-based FDD scheme, which employs an FPGA, was validated in real-time. The results demonstrated an accuracy approaching 100%, indicating a remarkable performance in classifying tasks. In a similar context, a ML-based system for fault detection and fault-tolerant control was proposed in [225] using a HIL platform for real-time simulation. A comparative study of six ML models revealed that the RF model outperformed the others by 91.99% accuracy. Another area of interest in recent times has been FDD for EVs based on DL techniques. For instance, a single fault diagnosis based on LSTM for an EV induction motor was proposed by [250]. A

dataset comprising faults was generated through an injection process that employed the use of short-circuit and open-circuit faults based on a simulation model of an EV system developed in MATLAB/Simulink. The superior accuracy of LSTM in comparison to alternative techniques is illustrated by the validation results of the proposed system utilizing an EV prototype. In addition, a data-driven fault classification algorithm has been developed in Jung et al. in [215] to address the issue of unknown fault classes and inadequate training data. In the aforementioned work, a Weibull-calibrated OSVM classifier combined with Bayesian filtering was developed to cover seven different single types of engine faults. The classification efficacy of hitherto unobserved faults in continuous datasets is exemplified by a real internal combustion engine. Nonetheless, the generation of the residual in the proposed approach necessitates the use of a high-precision mathematical model, which in turn increases both the cost and complexity.

Associated with the development of FDD for autonomous vehicles, Biddle et al. in [221] have demonstrated in their study that using SVMs from ML techniques can ensure high accuracy in detecting, isolating, and identifying multiple faults in multi-sensors with an efficient computational burden. The MATLAB/IPG CarMaker co-simulation platform, which considers five individual sensor faults, namely drift, hard-over, erratic, spike, and stuck fault, was employed to assess the efficacy of the proposed algorithm. However, the real-time constraints of the system behavior in the presence of the faults have not been considered in the generation of the data. Moreover, an analysis of the robustness of the developed model against sensor noise has not been performed, leaving room for further refinement.

Modifying the conventional DL architecture has been demonstrated to enhance fault detection performance in several areas, as evidenced by the findings presented in [251]. As an illustration, a DL-based fault detection method has been put forth in [252] to guarantee the safety of a rail vehicle system. The bidirectional LSTM-DAE network has been modified in the proposed work with the aim of overcoming the challenge of the unavailability of datasets under a faulty state. This is necessary for the determination of the added noise level. While the autoregression model, LSTM and BiLSTM-DAE exhibited superior performance relative to other models, the state of the system during simultaneous faults was not considered.

Although there has been significant progress in the development of real-time test platforms, the creation of an intelligent system capable of detecting, isolating, and identifying faults during

the development process remains in its infancy. For instance, Scharoba, et al. in [253] propose a system for the detection of anomalies in proximity using ML techniques to automatically evaluate test runs in embedded systems and identify any faulty behavior. The identification of anomalies within the behavior of the test object is achieved through the use of historical test records as a reference point. In the aforementioned study, a drive controller under development was utilized as a case study to assess the efficacy of the proposed framework. Notwithstanding the evident benefits of the proposed anomaly detection method, the study in question does not address the identification of single and simultaneous faults.

In accordance with the aforementioned findings from previous studies, the developed intelligent methods encompass a range of systems within the automotive field. Nevertheless, the diagnostic challenge of simultaneous fault occurrence in the presence of noisy and unbalanced data remains unaddressed, despite the impressive accuracy demonstrated by the proposed models. In this study, this gap in the literature is addressed by proposing a novel method for detecting and classifying simultaneous faults during real-time testing of an ASS during the V-cycle development process.

4.4.2 Detection and diagnosis of concurrent faults in automotive and other domains

The availability of the datasets containing the fault behavior provided the opportunity to employ a data-driven approach to the problem of concurrent fault diagnosis. For instance, a hybrid FDD framework, with data center cooling systems as the target application, was proposed by Asgari et al. in [254]. The proposed strategy employs a two-phase approach, utilizing single-class SVM and NARX for the detection phase and two DL techniques, 2D-CNN and LSTM, for the diagnostic task. A total of seven distinct fault types related to pumps and fans, as well as their respective combinations, were considered. Furthermore, the impact of incorporating noise into the training dataset with varying standard deviations, specifically 0.1, 0.5, 2, and 3, was examined. The experiments demonstrate the robustness and capability of the proposed model, with 100% accuracy in detection and diagnosis, based on F1-score and accuracy as evaluation metrics. However, the real-time application of this approach is constrained by the runtime computational limitations of the one-class SVM. Furthermore, the system behavior under normal and faulty conditions was simulated using a

simulation model of the target system, without consideration of real-time constraints. In their proposed study, Li et al. in [255] demonstrated that multi-label classification based on DL techniques can provide significant results for FDD of solid oxide fuel cell (SOFC) systems.

The significance of the aforementioned work lies in the fact that the simultaneous faults data samples are not required; rather, only the faulty data with single faults are necessary. PCA techniques were employed for the purpose of feature extraction. Moreover, a multi-class SVM technique was employed for the classification of nine distinct fault classes. Nevertheless, the validation results for concurrent faults with an F1-score of 84.93% are deemed unsatisfactory, despite the high performance of single fault classification with an F1-score of 96.4%. The dataset was generated in a simulation environment without consideration of real-time constraints, in addition to the high computational time of 26.1 seconds for classification. Consequently, it is essential to assess the viability of implementing the proposed model in real-time scenarios.

In the domain of automobiles, Wong et al. in [244] presented one of the first examples addressing the diagnostic problem of simultaneous engine faults using a probabilistic committee machine. The intelligent FDD system demonstrated diagnostic efficacy with accuracies of 92% and 81.49% for single and simultaneous faults, respectively. The authors selected a 4-cylinder in-line engine as a case study and utilized three distinct signal patterns from a real engine, encompassing 15 types of faults, to train and validate the model. In a similar context, but for an alternative application domain, a fault detection system based on fuzzy logic was developed for a continuous stirred tank heating system. This system is presented in [256]. Furthermore, wavelet transform techniques have been employed to address the issue of noisy data in the measurements. Consequently, the developed model has been demonstrated to be robust to noise during the multiple fault diagnosis process, with an accuracy of 100%. Furthermore, wavelet transform techniques have been employed for the processing of data exhibiting noise in the measurements. Consequently, the developed model has been demonstrated to be highly robust to noise during the multiple fault diagnosis process, with an accuracy of 100%. Nevertheless, the development of such a system requires a profound comprehension of the domain and the physical response, which can be challenging in complex software systems. In a similar manner, the authors of [257] propose a residual deep neural network coupled with the IIR Wiener filter denoising method in the context of noisy seismic data. The proposed method exhibits high performance in data denoising and reconstruction, as well as in the detection

of anomalous signals within the recordings. Consequently, less computational resources, effort, and time are required for the detection and denoising process. It should be noted that the diagnostic functions of fault identification and localization are not covered by this method. In [214], Safavi et al. proposed an architecture for a health monitoring system that considers multiple faults and sensors in autonomous vehicles. To address the task of fault detection, isolation, and identification, multiclass DNNs and 1D-CNNs were employed. The efficacy of the proposed methodology has been substantiated through the utilization of real-world sensor measurements. In this study, four types of sensor faults have been considered: drift, hard-over, erratic, and spike faults. The model's resilience to noise has not been evaluated. However, the proposed system demonstrates robust performance, with a detection accuracy of 99.84% and an identification accuracy ranging from 73% to 100%. Furthermore, the erroneous data was generated in a static manner using a normal data distribution with a single standard deviation.

It is evident that the detection and identification of concurrent faults in the presence of unbalanced data and existing noise in the measurements has not been sufficiently studied despite the emergence of novel FDD models with remarkable performances. Additionally, it is crucial to consider the range of fault types and the generation of faulty data in relation to the real-time system behavior under faulty conditions. In light of these limitations, the proposed research represents a novel contribution to the field of fault diagnosis by developing an efficient DL-based single and simultaneous FDD model that is capable of accounting for the noisy and unbalanced data of HIL tests. Moreover, datasets representative of the target model were gathered using an in-situ FI framework and a high-fidelity vehicle system.

4.5 Research Related to Real-Time Validation Platforms for Automotive Systems

Recently, in order to overcome the shortcomings of real test drives in terms of safety, cost, and effort, virtual testing has attracted the attention of researchers from different fields [258, 259]. The scope of this approach is not limited to validating the developed control strategy [260, 27] but also extends to developing human-machine interfaces [261]. Moreover, quality assurance aspects of the developed overall system, such as ADAS, can be effectively ensured while keeping costs and

complexity low [262, 263].

A variety of platforms have been employed to implement the virtual driving system, including those based on pure simulation tools, real prototypes, or real-time simulation systems. For instance, Saraoğlu et al. in [264] explored the adaptability of a model-based development approach, proposing a simulation framework for the analysis of safety characteristics of autonomous vehicle functionalities. In consideration of the traffic model and driving scenarios, this work employed the FI method to validate the target system's functional safety requirements. Additionally, the Simulink environment was employed to analyze the propagation of faults between system components. The method is limited to non-real-time executions, despite the demonstrated applicability of the proposed framework for systematic safety evaluation. Furthermore, the simulation framework does not consider manual driving, thereby failing to capture the actions of the driver. In a similar context, a methodology for the validation of decision and control strategies has been developed in [265]. The focus of this work is on automated driving control units. In addition to the selection of the controller, the scenarios and vehicle types are highly modular and adaptable. Furthermore, a three-dimensional model of the driving environment was constructed, taking into account the route with various traffic circles and junctions. Similarly, Sievers et al. in [266] proposed an integrated tool chain for autonomous vehicle testing, which enables the validation process to be conducted in real-time. The efficacy of the proposed tools has been validated through the assessment of sensor-supported control units at various stages of testing. To illustrate the utility of the proposed tools, a camera raw data injection setup was employed as a case study. Nevertheless, the proposed methodology is constrained to the validation process under normal operating conditions. In contrast, our study evaluated the safety and reliability of the SUT in the context of fault precision, considering dangerous conditions.

A number of studies have been conducted with the objective of enhancing the VIL simulator in order to address the limitations of simulation-based non-real-time testing in evaluating the SUT under real-life operating conditions. Consequently, for a variety of automotive applications, as mentioned in [267], the integration of real components with the SUT can be effectively validated in terms of real driving and a virtual environment. For instance, Park et al. in [268] proposed a modular VIL topology for validating ADAS applications. A real vehicle was employed to assess the real-time stability of ADAS, in addition to the 3D virtual environment and sensor emulation.

The results demonstrate the benefits of the proposed topology with respect to saving maintenance time and costs, and providing a highly realistic driving experience. In [269], the authors proposed a testing methodology called Hybrid Testing for ADAS, which aims to exploit the advantages of real road-based testing and pure simulation-based testing. In comparison to traditional ADAS testing methodologies, the study presents the advantages of the proposed methodology, which combines the virtual environment of simulated vehicle components with the real vehicle prototypes. The use case of a lane change maneuver was employed to illustrate and elucidate the architecture and structure of the co-simulation framework. In particular, the aforementioned study considered the ADAS functions, namely the trajectory planning algorithm, as a single SUT. The aforementioned functions were executed in the MicroAutoBox. Nevertheless, the safety level for the operator during a malfunction was quite low. Moreover, in the event of malfunctioning components, it is not possible to assess the performance of the target system. In contrast, our study offers an effective approach to assess the reliability and safety attributes of the SUT, taking into account irregular environmental circumstances and malfunctions in system components.

This has created a necessity for test platforms that can bridge the gap between real tests, SILs, and VILs with respect to cost, safety, reliability, reproducibility, and flexibility. To fulfill the aforementioned requirements, real-time HIL simulation has been proposed as the optimal solution. In various applications, as mentioned in [228], HIL simulation has achieved remarkable success. In particular, several attempts have been proposed in the last decade for the development of real-time driving simulators. These include the work of [270, 271, 272]. For instance, in order to provide effective and accurate real-time validation of motion controllers, a signal HIL simulation platform focused on EVs was proposed in [273]. The experimental platform considered comprehensive target system characteristics by modeling vehicle dynamics in the longitudinal, lateral, and yaw directions, in addition to the possibility of manual driving. Furthermore, three driving scenarios were employed to illustrate the efficacy of the proposed study: a normal road, a low-adhesion road, and a slick road. However, in contrast to our proposed study, the platform does not consider anomalous environmental conditions or malfunctions due to defective system components. With the same objective but for a different application, Chen et al. proposed a test platform based on HIL simulation in [30]. The objective of the proposed study is to validate the functionality of real ECUs for autonomous vehicle development. The platform is comprised of three distinct layers: system

modeling, multi-sensor simulation, and a virtual test environment layer. This architectural configuration is in accordance with the proposed platform architecture. Similarly, as in this dissertation, the interaction of multiple agents and the modeling of scenarios are represented in 3D visualization, with the real ECU connected to the simulated system via the CAN bus in a closed loop. In this work, both self-driving and manual driving modes to consider user behavior are enabled.

With regard to the issue of validating safety-related systems, including ABS and ESC systems, Tumasov et al. in [274] proposed a HIL test bench that enables virtual test driving. The proposed work is distinguished by the integration of a real vehicle component, namely a hydraulic cylinder of the braking system, along with the real ECU and the dynamic model. The efficacy of the proposed platform is evaluated by comparing the outcomes of virtual and field tests of the same maneuver, taking into account the requirements of stability control. However, in the event of a malfunction in the physical components, there is a significant risk of damage to the system. In contrast, our proposed work involves the simulation and implementation of the physical components with fidelity to accurately analyze the effects of failed components in real-time. Finally, in [275], the applicability of the HIL system for the development of an integrated driving simulator (IDHIL) for the evaluation of cooperative eco-driving systems has been demonstrated . The HIL simulator, the network simulator, and the vehicle simulator represent the core components of the proposed work, which employs ASM, MicroAutoBox, and dSPACE tools. Furthermore, two use cases were employed to illustrate the simulator's efficacy: a normal hybrid EVs simulation and a connected hybrid EV eco-speed demonstration. Nevertheless, while our work modelled a generic environment encompassing abnormal conditions and critical scenarios, the environment in the study is constrained to the predefined normal conditions of the dSPACE tool, namely the defined scenarios in ModelDesk. Another distinction is the applicability of our proposed framework for the simulation of component faults, such as sensor and actuator faults, to validate the response of the SUT in the event of a fault.

In conclusion, there has not yet been an adequate investigation of the development of a generic validation framework that meets the criteria of flexibility, cost-effectiveness, comprehensive coverage, repeatability, and reliability. The proposed validation framework fulfills these requirements and represents a novel contribution to the field of study. The proposed validation framework employs real-time HIL simulations, a comprehensive vehicle model with high accuracy, a virtual driving environment with extensive coverage of anomalous driving scenarios, both manual and

automated driving, as well as real-time FI, all in real-time.

4.6 Conclusion

This chapter presents the main contributions of the proposed studies in the related researches to the scope of this thesis. It focuses on providing an overview of the main contributions and limitations of the related works. According to the research questions formulated in this study, the corresponding related works were presented in several sections. In the first Section (4.1), the state-of-the-art for tackling the problem of unavailability of faulty data for developing a data-driven FDD model was presented. In the second Section (4.2), the main drawbacks of current DL-based methods for single-sensor FDD in automotive systems are presented. Besides, the limitations of the proposed solutions in fault diagnosis of unknown and concurrent faults are presented in this Section 4.3 and Section 4.4, respectively. Finally, in Section 4.5, the state-of-the-art of current test platforms for automotive system validation has been presented. What can be concluded from this section is that the concept presented in this thesis is innovative and has not been presented in research before in the filed of ASSs development.

5 Overall Concept

This chapter presents the solution concept for the problem statement presented in Chapter 3. The proposed methodology addresses the shortcomings of the conventional failure analysis process during real-time validation of ASSs. In particular, the evaluation process of test drive records during the system integration test of the V-model is improved in an efficient manner, thereby reducing the time and effort required. This chapter presents the details of the phases of the proposed methodology in the context of the real-time validation process.

5.1 Machine Learning-assisted Failure Analysis Methodology for HIL Test

This section presents an overview of the proposed methodology to address the challenges identified in Section 3.4. In particular, the section commences with an introduction to the fundamental concepts of the validation process utilising HIL simulation (subsection 5.1.1). Subsequently, the proposed solution for the first challenge is presented in subsection 5.1.2. Since the RQ2, RQ3, and RQ4 fall within the scope of developing a ML-based FDD model, the corresponding solutions are presented in subsection 5.1.3. The proposed solution for challenge 5 is presented in subsection 5.1.4. Finally, the conclusion of the chapter is presented in subsection 5.1.5.

The overall concept comprising the extended phases of the HIL-based real-time validation process is depicted in Figure 5.1. In addition to the fundamental phases of conducting HIL tests, the extended processes are depicted, which permit an intelligent analysis of the test records. To facilitate the mapping between the research questions and the proposed solution, the extended blocks in the methodology have been mapped to the corresponding research questions and chapters. The proposed methodology can be implemented in four stages: (1) performing the system integration test steps, (2) collecting representative fault datasets based on a real-time FI framework, (3) developing ML-based FDD models, and (4) validating the developed models on a virtual test drive platform.

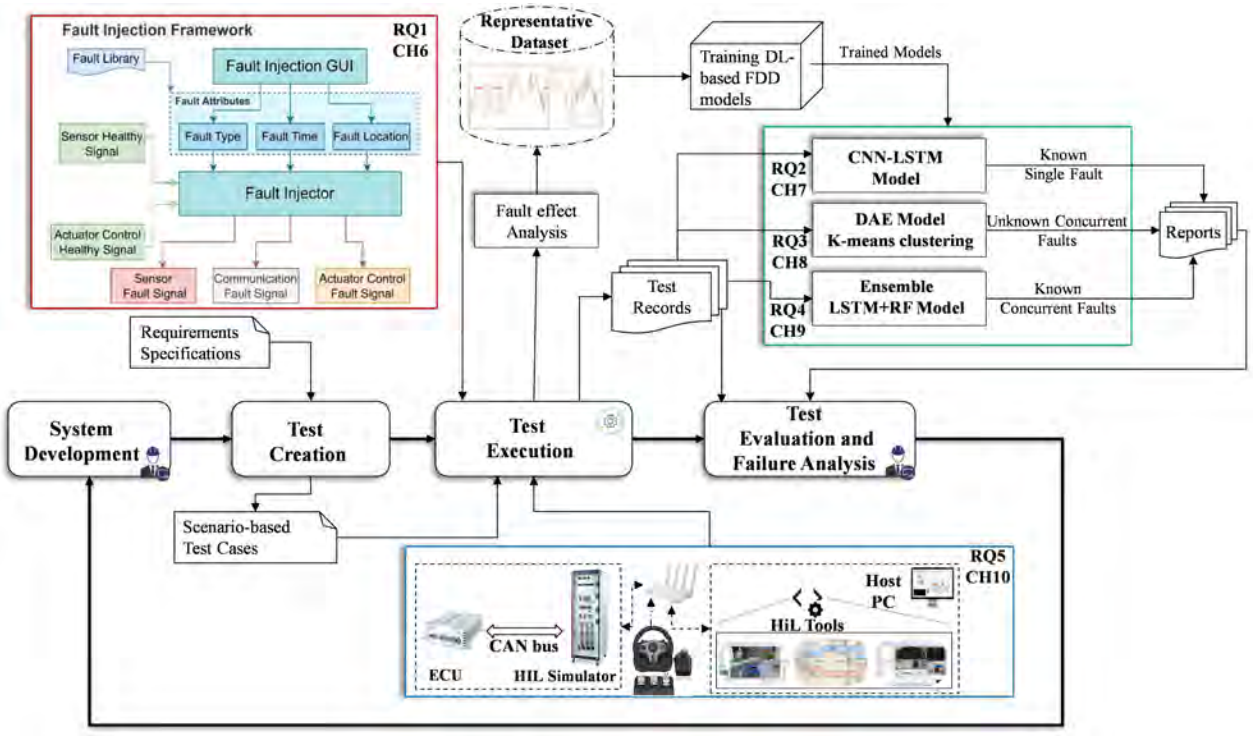


Figure 5.1: Proposed methodology for intelligent analysis of HIL tests records

5.1.1 System Integration Test based on HIL Simulation

In accordance with the functional safety standard ISO 26262-4 clause 7 [15], the system integration testing phase of the V-model is regarded as a crucial stage in ensuring the correct behavior of the SUT in its environment. Such testing serves to validate and analyze the response of the SUT to its physical environment under different operating conditions. Consequently, potential faults that could lead to risks in the real world during the operational phase can be mitigated. In other words, if the test is performed, it should provide sufficient proof that each component of the system interacts properly and carries out only the intended functions. Additionally, there should be a reasonable degree of confidence that the system does not exhibit any unintended functions that could potentially violate a safety objective [15].

The advent of HIL simulation has facilitated the cost-effective, safe, and flexible execution of the validation process. By enabling real-time validation prior to the actual production of the target system, faults can be identified and rectified at an early stage of the development process. This results in a significant reduction in in the costs associated with testing. Moreover, the potential

for safety risks to arise during real-world testing is mitigated by the provision of a comprehensive range of test scenarios under critical conditions in a secure environment.

It can be observed in Figure 5.2 that the SUT developed in the development phase is subsequently transferred to the validation phase, i.e., test creation. During this phase, the test scenarios are created and executed on the HIL platform. In light of the aforementioned validation methodology, the phases have been presented in detail in Section 3.3. The test scenarios are created based on the requirements specifications and use cases, in which the target functional and non-functional aspects to be validated are described. In addition to the typical and extreme operational conditions, the defined scenarios encompass safety-critical scenarios and unusual edge cases. In this manner, a multitude of facets of dependability, including safety, reliability, robustness, and performance, as well as functional correctness, can be guaranteed.

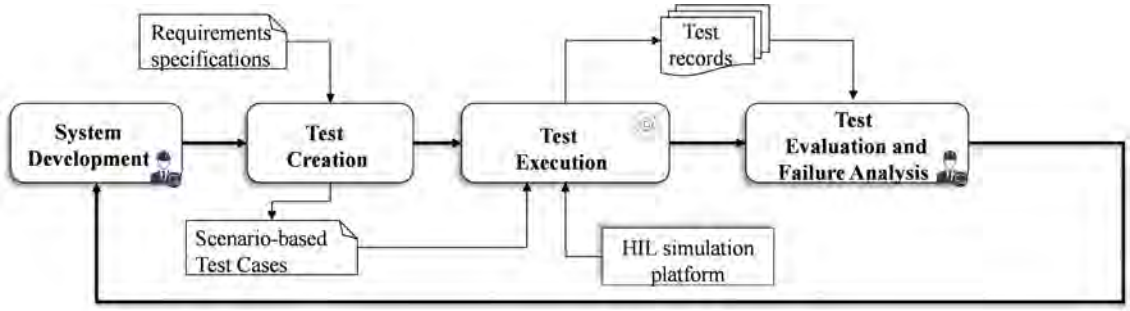


Figure 5.2: System integration test process based on HIL simulation

The designed test scenarios and TCs are executed in a simulated and controlled environment, namely the HIL test bench. Prior to the execution of the test, it is necessary to define the setup and configuration of the hardware, software, and simulation tools. In particular, the SUT, HIL, real hardware components and their communication, and acquired systems should be configured. During the execution phase, a number of activities are carried out, including the transmission of sensor and command signals between the controlled system and the SUT, the monitoring of the SUT's output, and the logging of the data. The captured behavior is represented as time series data, which are known as HIL records. These records contain the SUT and HIL output signals.

Based on the requirements of the intended system performance and the safety criteria, the test results are evaluated with the objective of determining deviations in behavior and analyzing the nature or causes of the underlying detected issue. This stage is designated as test evaluation and is

conducted by an expert. Specifically, the detected failures are analyzed to determine the location, type, and severity of the root causes. In order to fulfil this objective, the tester should perform a number of steps, including data pre-processing, the investigation of discrepancies in the system behavior, fault detection and fault type identification. The results of the evaluation are documented in a test report, which contains findings, issues, observations, and faults. This report is then sent back to the development team.

5.1.2 System Behavior Analysis and Data Collection using Real-Time Fault Injection Framework

FI is a highly effective methodology employed throughout the system development process to ascertain the safety and reliability of the SUT. This technique is based on the concept of introducing faults into the developed system in order to analyse and evaluate its response to abnormal conditions. The functional safety standard ISO 26262 [15] strongly recommends the use of FI as a testing method. As demonstrated by Pintard et al. in [276], the FI method can be employed on both sides of the ISO 26262 V-cycle. In contrast to traditional analytical techniques, such as FMEA and FTA, the FI enables the capture and representation of the dynamic behavior of the SUT under faulty conditions. In this study, the FI approach is employed to analyze the effect of potential hardware faults on system behavior and to determine the critical faults that lead to failure at the system level. As a consequence of the FI methodology being employed during the real-time validation process, a representative dataset including the critical fault is collected as a basis for training the intelligent models of Ml applications.

The real-time FI process in our proposed framework occurs at the signal interface between the control unit and the HIL simulator during real-time execution. Consequently, it is possible to execute the system model of the ECU and the plant in real-time as a black box without modification. Moreover, the injection process is executed in accordance with the programmatic specifications without the necessity of extending the architecture of the target system model with additional components. Figure 5.3 illustrates the integration of the developed FI framework with the HIL testing phases. The input to the FI framework is healthy data representing the system behavior under normal conditions. Subsequently, the target signals are subjected to manipulation in accordance with

the fault attributes as defined by the user. Prior to injection, three attributes in the fault injector must be configured: the type of fault, the location of the fault within the system, and the time at which the injection takes place. The term "fault type" encompasses a wide range of potential sensor and actuator faults, including gain, offset, hard-over, stuck-at, delay, noise, packet loss, drift, and spike faults. The target sensor and actuator signals are identified as locations where faults are injected. The temporal parameters of the FI, including its duration and injection point in time, are specified in accordance with the specific driving cycle or standard system behavior in question.

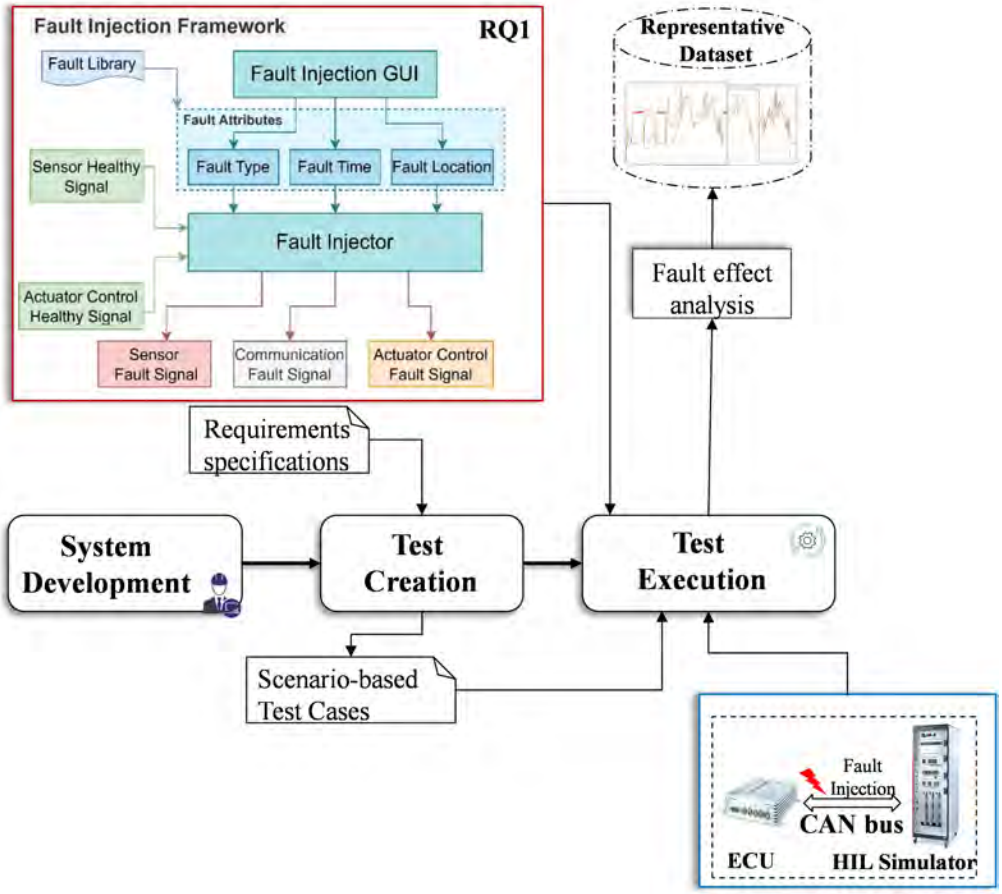


Figure 5.3: Representative Dataset generation and collection using real-time fault injection framework

Finally, fault effect analysis is conducted either manually, based on the expertise of the analyst, or automatically, based on the functionality of the automation tool and the evaluation functions. Once the effects of single and concurrent faults have been analyzed and the critical faults have been determined, the collected healthy and faulty datasets are stored as a result of the test execution.

Two types of data are collected, i.e., time series signal data and textual log data. This enables the collected data to be employed for a variety of purposes, including fault detection, diagnosis, prognosis, and the identification of the root causes of failed TCs.

5.1.3 Machine Learning-based FDD Models for Recordings Analysis

The representative dataset collected by the real-time FI framework is subjected to pre-processing prior to its utilization in the development of target models based on a data-driven approach. Specifically, the data is transformed into a format suitable for ML and DL model development, such as CSV, MAT, or XML.

It is notable that a significant proportion of the information contained in the generated dataset is either redundant or of no value. As an illustration, each CSV file within the dataset contains data pertaining to the recording process, as well as the data samples. As a result, the collected data must be subjected to preprocessing in order to eliminate outliers and superfluous information. The preprocessing phase encompasses three discrete steps. The first is the removal of non-useful information and outliers. The second is data scaling and normalization, and the third is data splitting. Subsequently, the CSV files are converted into Excel format, after which the training variables are selected and the non-useful data is removed. This ensures the availability of useful data samples for the development of target ML and DL models. Subsequent to this, a normalization function is applied to all variables, given the distinct range of values exhibited by each variable. Consequently, all variable values are scaled within the range [0, 1]. Finally, data is divided into three distinct subsets for the purposes of training, validation and testing. It is important to note that when developing ML models based on supervised learning, the process of labeling the data should occur prior to data splitting. During this process, the fault classes are assigned to the corresponding data for the classification task. The general division of the dataset is as follows: 80% is reserved for training, while 10% is designated for validation and 10% for testing.

To address RQ2, RQ3, and RQ4, three distinct DL models were developed to detect and identify faults in various scenarios, including known, unknown single, and simultaneous faults. These models are illustrated in Figure 5.4. To this end, a variety of DL/ML architectures were employed in the development of the models. In particular, a hybrid CNN and LSTM architecture was developed to detect and classify single known faults, while a combination of GRU and DAE with K-means

clustering was used to detect and group unknown single and concurrent faults. Finally, an ensemble LSTM-RF architecture was developed to detect and identify known concurrent faults under noisy and imbalanced data.

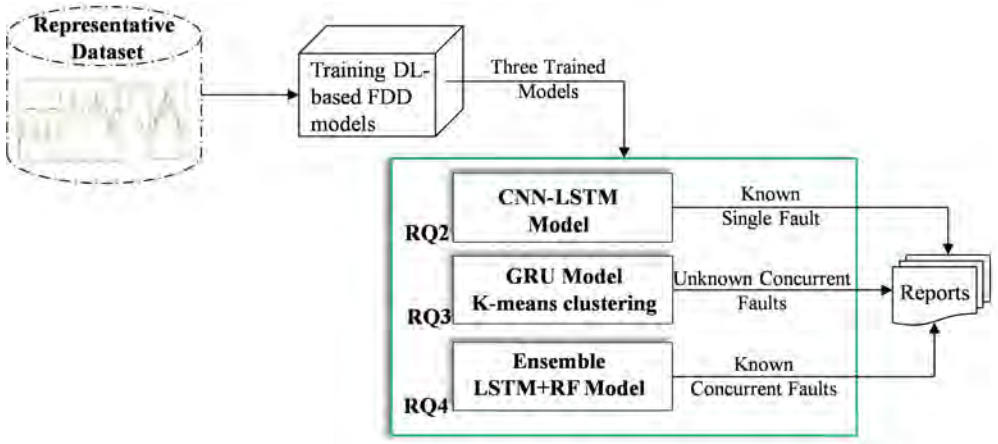


Figure 5.4: ML-based FDD models for recordings analysis

During the training and validation phase, the architectures of the models are optimized by tuning the hyperparameters of the models according to the validation criteria. Consequently, the phase culminates in the achievement of the optimal convergence state for the target models, defined by the highest performance in terms of accuracy and loss values. The developed models are then implemented in the deployment phase during the execution of virtual test drives. The robustness and effectiveness of the proposed models are validated based on various metrics, including accuracy, precision, recall, and F1-score, using different driving scenario datasets. Moreover, the superiority of the proposed approach is demonstrated by comparing the results with those of the state-of-the-art FDD methods.

5.1.4 Models' Employment on Real-Time Virtual Test Drive Platform with HIL

Recently, the ability of the HIL system to provide a realistic simulation has enabled the execution of virtual test drives, which consider the behavior of real hardware components, such as ECUs, sensors, and actuators. In a controlled environment, the system models executed in a real-time simulator interact with software control algorithms and physical components in a manner that is

consistent with the real-world scenario. Furthermore, the real driving system, comprising steering, gear, and pedals, is employed to enable the test drive in consideration of the user's behavior (see Figure 5.5). In a closed-loop configuration, the sensed signals generated by sensor components, which simulate real-world sensor data, are transmitted to the ECUs. Conversely, in accordance with the control algorithms, the ECUs transmit the corresponding commands to the target actuators, such as electric motors, pumps, or valves. In such instances, the responses and behavior of the vehicle's physical components can be simulated in real-time. In general, the communication between the ECUs and HIL simulators is conducted via industrial communication protocols, such as CAN, FlexRay, and LIN. Notably, in HIL setup, the embedded software of the SUT in the target machine is connected to a HIL simulator. The simulator can be connected to either real components, such as sensors, actuators, and control units, or to the actual hardware system, such as a real engine or steering system.

In accordance with the specifications outlined in the target system requirements, a series of driving scenarios is constructed, including acceleration, braking, cornering, lane changes, and emergency maneuvers. In addition to the specifications and conditions, each scenario should define an expected output, which is referred to as desired behavior. The virtual driving environment allows for the examination of critical conditions in a controlled, safe, and reproducible manner, considering the time constraints. Through the use of 3D-visualization, a variety of test drive routes, including those on highways and in urban areas, can be covered. These routes take into account the road topology, including surface irregularities. In addition, the environmental conditions, such as rain, snow, fog, and lighting, are considered. It is noteworthy that the test scenarios can be executed either automatically or manually via a driving system and virtual environment.

During the execution of the test, the internal system variables, such as the signals from the ECUs, sensors, and actuators, are recorded via data acquisition systems. In other words, the actual system behavior is captured in real-time as multivariate time series data, which are known as test drive records. The captured behavior is non-linear, dynamic, and multi-modal. Subsequently, the recorded data is subjected to analysis by the test engineer with the objective of identifying and rectifying any deficiencies in the system. In order to achieve this, in addition to verifying the diagnostic trouble codes of the ECUs, based on the expertise of the domain and the system, the divergence of the anticipated outcomes in the recordings is identified and analysed. Finally, the

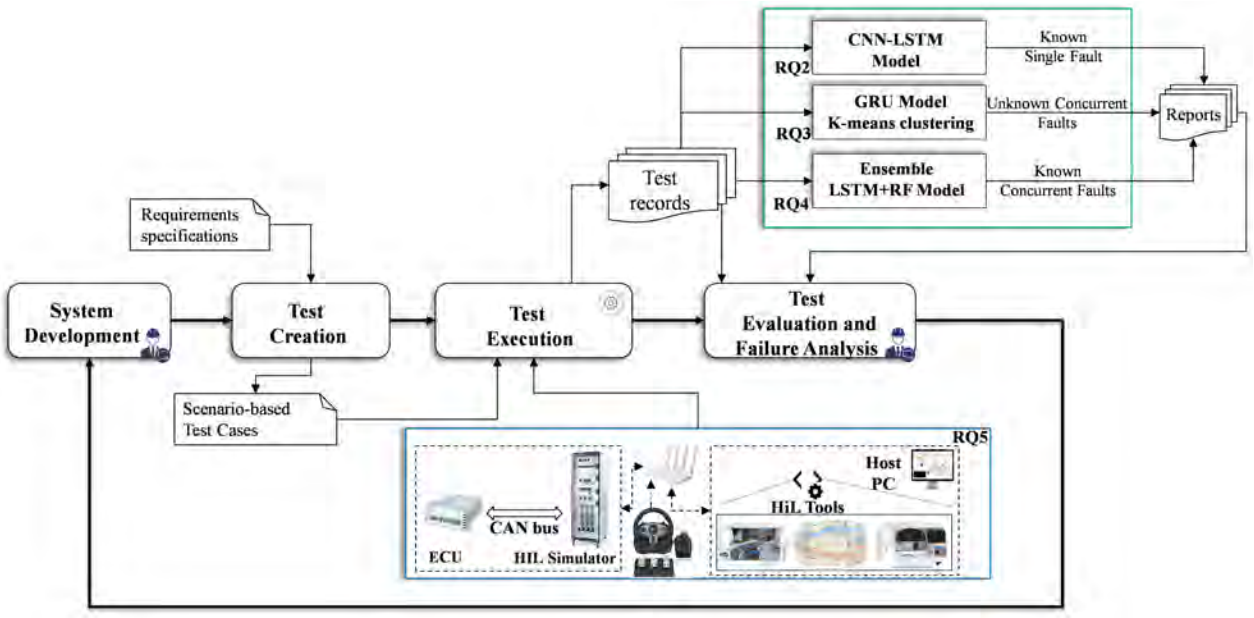


Figure 5.5: Models' employment on real-time virtual test drive platform with HIL

findings and observations, including the identified faults, are documented in the form of a test report, which also includes recommendations or suggestions for improvement. Subsequently, the testing report is transmitted to the development phase, where the model is modified and adjusted in order to achieve the desired performance.

Following the execution of the tests, several hundred test records are transmitted to the developed ML models (subsection 5.1.3) with the objective of detecting and identifying the faults encountered. The inputs to the models are the time series data contained in the test records. On the other hand, the output of the models is the test reports, which include the occurrence time and the type of the detected fault. Specifically, following the pre-processing of the sequential data, the non-linear features are automatically extracted by the hidden layers of the developed DL architectures. The extracted features are then passed on to subsequent layers in the output, allowing for classification according to the corresponding fault classes. Consequently, three reports are generated as the output of the developed models, which contain the time and type of occurrence of the faults in the records. It is noteworthy that the analysis reports generated by the models encompass a range of fault scenarios, including single, simultaneous, known and unknown faults. Moreover, the actual operational conditions of the data, including noisy conditions, are taken into account. Finally, the identified clusters/fault classes are subjected to a second review by the test engineers in

the fault analysis phase. This review is conducted to confirm the cause of the failures that occurred. The pertinent data from the analysis of the records, findings, and observations are transmitted to the developer for the purpose of rectifying the issue and modifying the model as necessary.

5.1.5 Conclusion

In conclusion, the objective of this thesis is to address the limitations of conventional methods employed during the system integration testing of ASSs development, specifically in relation to test recordings analysis. The central concept is to enhance the process of analyzing test drive recordings by developing an intelligent model based on historical data pertaining to faulty and fault-free instances. The study specifically addresses the problem of FDD of test drive records. The study encompasses both random and systematic unexpected faults, which may occur simultaneously. In addition to investigating the issue of identifying known fault types, the study also addresses the detection and categorization of unknown faults in the presence of noise. The findings of this study contributes to the development of advanced techniques for recordings analysis of the real-time testing process in various application domains. Moreover, the outcomes facilitates further advancement in the data-driven FDD approach during the system integration testing phase of the V-model, benefiting from the pure non-real-time simulation phase and real-world application phase. Consequently, during the development process, engineers are supported by more efficient and effective identification and remediation of faults, resulting in a high level of safety and reliability.

6 Fault Injection Framework for Dynamic Behavior Analysis and Representative Real-Time Dataset Generation

The main focus of this chapter is on answering the research question RQ1 derived and formulated in Chapter 3, i.e. Section 3.5. Specifically, this chapter presents and describes a novel approach to solving the problem of unavailability of a representative dataset subject to critical faults. To this end, an overview of the chapter is provided in Section 6.1 followed by the proposal of a novel real-time FI framework integrated with the HIL platform in Section 6.2. To demonstrate the applicability of the proposed framework in terms of analyzing the system behavior and generating representative datasets, a case study from the automotive domain is considered. Furthermore, the implementation steps and the experimental setup for the selected case study are defined in Section 6.3. The results of applying the proposed approach to the case study are analyzed and discussed in Section 6.4. Finally, the conclusion of the chapter is stated in Section 6.5.

Note: It is important to highlight that the proposed methodology and the corresponding implementation outcomes presented in this chapter have been published in [277, 278].

6.1 Chapter Overview

The application of a data-driven approach has been observed at various stages throughout the system development lifecycle due to its capacity to derive insights from historical data. Nevertheless, despite its superiority over other conventional approaches, such as those based on models or signals, there remains a significant challenge in the form of the unavailability of representative datasets. It is, therefore, imperative to develop novel solutions that generate representative fault data that accurately reflect real-world operating conditions for a variety of engineering applications. This chapter

presents a novel approach based on HIL simulation and an automated real-time FI method. The objective of this research is to develop a methodology that can be employed to generate, analyze and collect data samples in the presence of single and concurrent critical faults. The generated dataset is utilized during the system validation phase of the V-cycle development model to develop ML-assisted test strategies. The developed framework is capable of generating not only time series data but also textual data, including fault logs, in an automated manner. To illustrate the framework's capabilities and benefits, a case study employs a high-fidelity simulation model of a gasoline engine system with a dynamic entire vehicle model. The outcomes illustrate the suitability of the proposed framework for simulating and capturing the system behavior in the context of internal fault occurrences within the system's components. Furthermore, the proposed framework's efficacy in analyzing system behavior and identifying critical faults during real-time system validation in realistic operational conditions has been demonstrated.

6.2 Proposed Framework

In this section, the proposed framework is presented, including real-time HIL simulation, FI framework, data analysis, and management, as shown in Figure 6.1.

6.2.1 Real-Time HIL Simulation System

The HIL system, which simulates and executes the complex target system in real-time, constitutes the core of the framework. The system is comprised of two primary components, i.e., the HIL simulator and the target prototyping system (control unit). The HIL simulator is responsible for the real-time execution of the controlled system (plant). In this case, the entire vehicle model constitutes the controlled system. The principal subsystems of the selected system are the engine model, the vehicle dynamics model, the environment model, the traffic model, and the powertrain model. It should be noted that the model of the controlled system also includes the model of the SoftECU, which may be considered a virtual ECU. The ECU model represents the SUT and is deployed and executed on the target machine. In this study, MicroAutoBox II is the target machine and is considered as a Rapid Control Prototype (RCP) and acts as the real ECU. The connection between the HIL simulator and MicroAutoBox II is established via the CAN bus. In the simulation

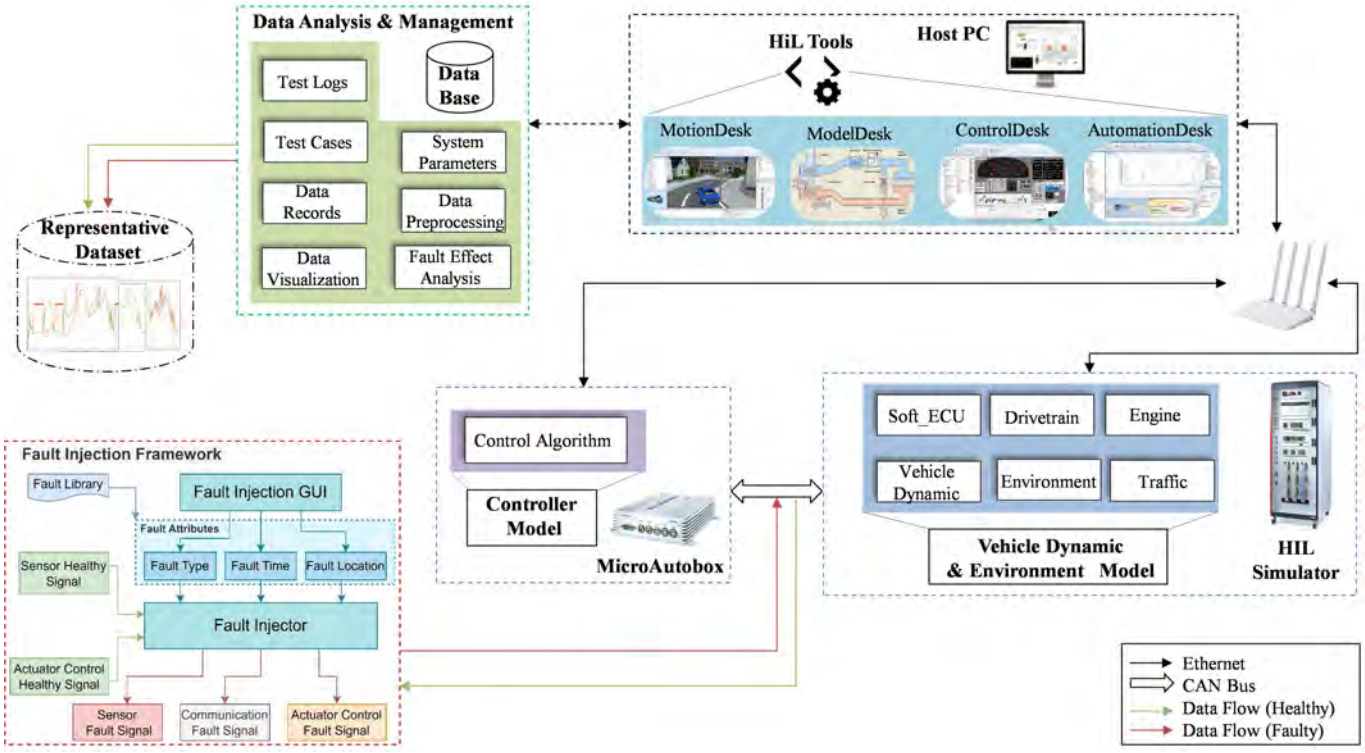


Figure 6.1: Proposed methodology for fault effect analysis and representative real-time dataset generation.

environment, i.e., in the ECU model and in the plant model, the signal interface on both sides is modeled to establish the connection. The generated code for these models is then injected into the ECU and HIL simulator via Ethernet from the host PC.

The configuration and experimentation processes are conducted on the host PC. To this end, MotionDesk, ConfigurationDesk, ModelDesk, AutomationDesk, and ControlDesk, five distinct software tools developed by dSPACE [166], are used. ModelDesk is employed to design the test drive scenarios and the TCs, as well as to parameterize the system model. Once the model has been configured, ConfigurationDesk automatically generates and deploys the code for both models, the SUT and the plant. 3D visualization of the driving environment and dynamic traffic is enabled by MotionDesk. ControlDesk is employed for instrumentation, measurement, data acquisition, and real-time experiment control. Furthermore, the tool enables the user to switch between offline (SoftECU) and online (real ECU) execution modes.

6.2.2 FI Framework with HIL

FI process is conducted at the signal interface between the ECU and the HIL simulator during the real-time execution of the system, as shown in Figure 6.2. This ensures the real-time execution of the ECU and plant system model as a black box without any modifications. Moreover, injection can be carried out in a programmatic manner, obviating the necessity to incorporate any supplementary code into the target system. The FI framework accepts data representing the system's behavior under normal conditions as input. Subsequently, the target signals are manipulated in accordance with the user-configured fault attributes. Prior to injection, three attributes should be configured in the fault injector, i.e., fault type, fault location, and FI time. The fault type encompasses a range of fault types, including gain, offset, hard-over, stuck-at, delay, noise, packet loss, drift, and spike faults. This comprehensive approach allows for the coverage of the majority of sensor and actuator faults.

The location of the faults to be injected is the target sensor and actuator signal. The duration of the FI and the injection time are specified in accordance with the drive cycle or standard system behavior. When the HIL is operational in real-time, the fault injector unit assumes responsibility for managing the healthy signals on the CAN bus, once the aforementioned FI attributes have been configured. The manipulated signals interact with the system variables in a closed-loop configuration between the simulator and the ECU. Consequently, the system response to the abnormal state of faulty components, such as sensors or actuators, is captured as a deviation in system behavior, which is recorded as multivariate time-series data. In order to achieve comprehensive coverage and a diverse range of conditions within the collected datasets, the FI process is repeated for each fault test case during system execution. In the case of simultaneous faults, two distinct types of faults are injected simultaneously at two different locations, and the aforementioned process is repeated.

In the HIL user environment, the tester is able to analyze the recorded data and to control the HIL system in real-time via the use of HIL tools. This is accomplished by creating a test case that includes a list of sensors, actuators, and driving situations. The HIL system is operated by means of HIL software tools from a computer machine. The aforementioned tools facilitate the configuration of the HIL system in a real-time environment. For the purposes of control and data transfer, the computer and the HIL system are typically connected by an Ethernet cable.

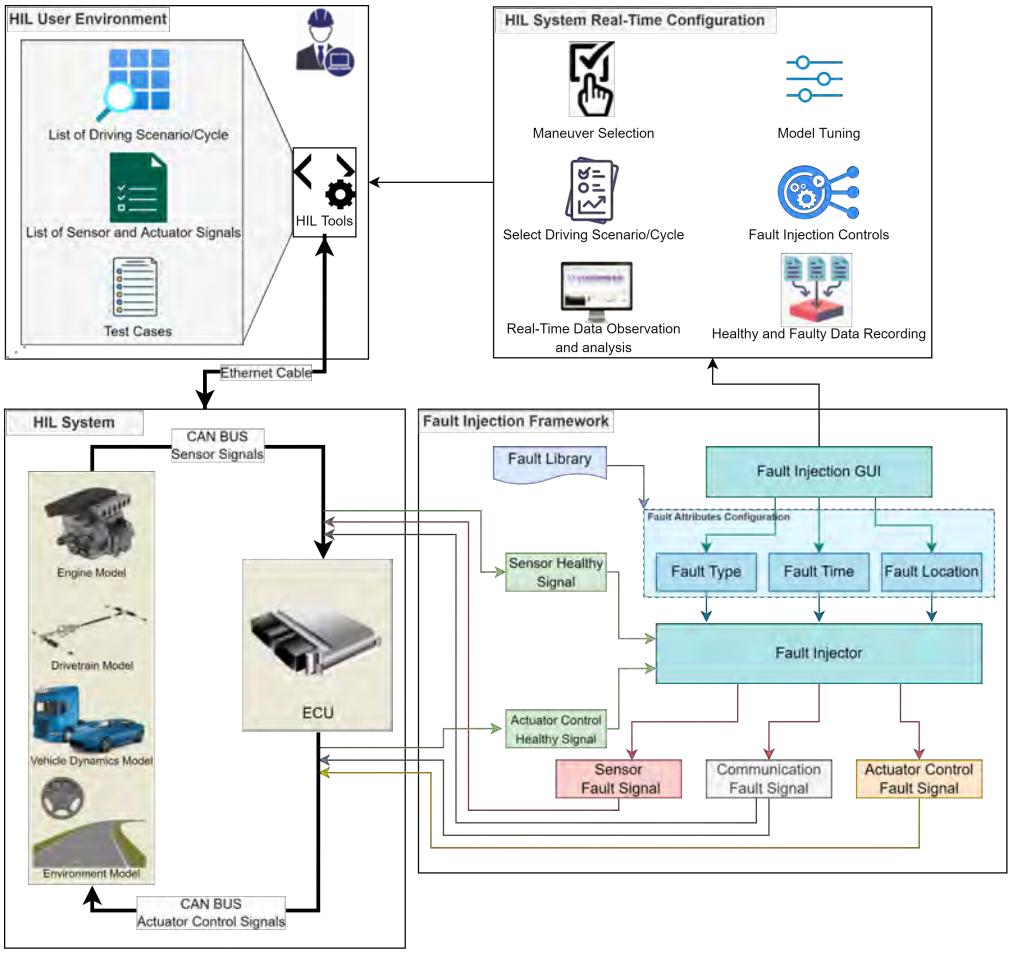


Figure 6.2: Fault injection framework with HIL system

The primary objective of the FI GUI is to provide support to the tester within the HIL user environment during the execution of the FI runtime. A tester is able to perform a number of actions via GUI as shown in Figure 10.5.

Besides the manual FI, this study facilitates also the automated execution of real-time FI within the proposed framework, thereby overcoming the limitations of the manual FI process. To this end, an automation software tool, AutomationDesk, was employed. AutomationDesk facilitates not only the automatic execution of systematic TCs but also the automatic evaluation and reporting of test results. The AutomationDesk software was selected to provide the necessary environment for the automated execution and evaluation of the FI process. The test scenarios, which include the environment, the roads, and the dynamic objects, are designed in ModelDesk and then triggered for execution in the test routines of AutomationDesk. As illustrated in Figure 6.4, the FI scenarios

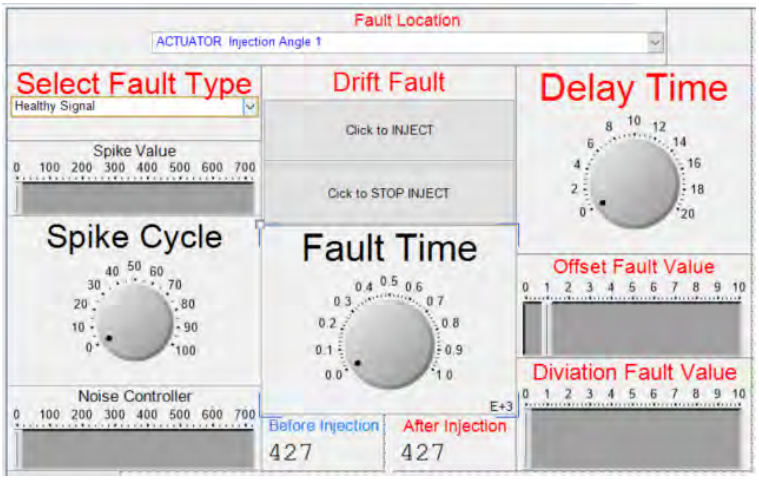


Figure 6.3: GUI of the proposed fault injection framework

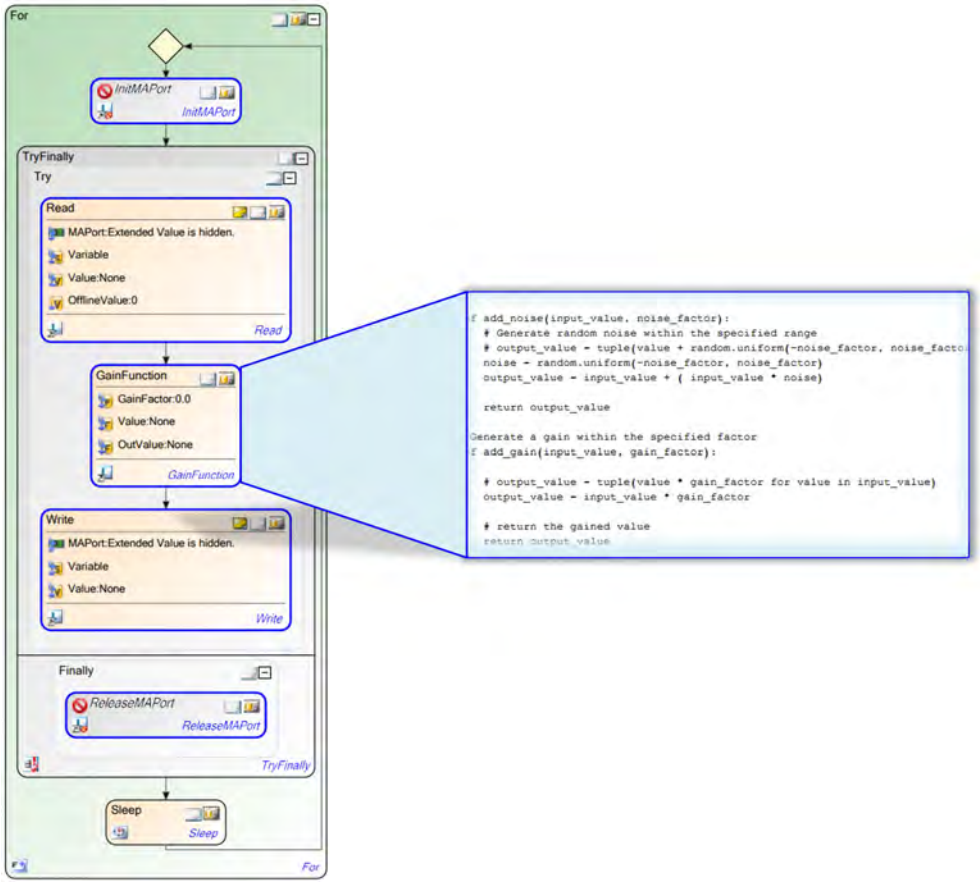


Figure 6.4: Automated real-time fault injection process.

are designed in AutomationDesk as a sequence of block hierarchies. The FI process is executed in three phases: the read phase, the signal manipulation phase, and the write phase. The "read" block

utilizes the signals of the system variables to identify the locations of the faults to be injected, i.e., the system components. The "write" block employs the signals of the system tags to identify the locations of the faults to be injected, which correspond to the system components. It should be noted that the injection operation is accomplished through the source code of the manipulator function representing the fault modes, without extending the original architecture of the system model. Subsequently, the fault conditions are initiated by the write block during runtime without compromising the timing behavior of the SUT.

6.2.3 Dataset Collection and Preparation

The execution of the TCs, including the FI, results in the recording of the system's behavior as a time series of data, which is then evaluated against the expected behavior. This process automatically identifies and documents in the generated test reports the critical faults that result in violating the functional safety requirements. The specifications of the recording process are determined by the logging system within the HIL platform. At the system level, the logging procedure considers the following system variables: engine speed, vehicle speed, engine temperature, engine torque, throttle position, intake manifold pressure, and rail pressure. In order to collect fault-free data samples, the system should be run under fault-free conditions. Conversely, the faults samples are collected as a consequence of operating the system under conditions of malfunction, i.e., random faults of sensors and actuators. Notable, transient faults are injected into the target sensor/actuator signals for a specified period of time in order to collect a dataset containing unbalanced data. Consequently, the ratio of faulty and healthy samples in the generated dataset is different. The dataset collected from the injection of critical faults, as well as the healthy data, is then considered a representative real-time dataset for the development of an intelligent model using ML/DL methods.

Data preprocessing, which involves the removal of outliers and redundant patterns, is performed along with the generation of data. The main steps in data preparation are variable selection, data cleaning, data labeling, scaling and normalization, balancing, and data partitioning. Furthermore, in order to be prepared for the ML/DL model development process, the data should be converted into an appropriate format, such as CSV, Mat, or XML. Correcting missing samples and reducing irrelevant data are also included in this phase. Moreover, this approach reduces the cost of computation and helps to avoid the problem of overfitting. In order to perform the labeling process, this

work employs a multi-class, multi-label approach, which allows for the consideration and labeling of all possible classes. The normalization process is conducted via the Z-score function, which scales the variable values within the range of [0, 1] to ensure that attributes with high values do not outweigh those with lower values. This process ensures a more balanced representation across the entire range of values. Finally, the preprocessed data is presented on the host PC in order to facilitate a more comprehensive grasp of the underlying patterns and features prior to training the DL/ML models. It is important to note that the data management is conducted on a host PC. The document outlines the system requirements, test specifications, scenario descriptions, and test methods.

6.3 Case Study and Experimental Implementation

In order to demonstrate the applicability of the proposed framework, this section presents the selected case study, outlining the system architecture and implementation steps.

6.3.1 Case Study

A gasoline engine system from dSPACE [166] was used as a case study in the automotive domain for the purpose of validating the proposed approach. The system's architectural layout is depicted in Figure 6.5.

The modeling of the various systems and subsystems was conducted in a manner that accurately reflected the actual physical properties. MATLAB/Simulink was selected as the graphical simulation and modeling environment for the dynamic system. The Simulink tool offers a number of important features that facilitate the design, analysis, and testing of dynamic systems using a model-based approach. Furthermore, Simulink facilitates the testing of SUTs within a simulated environment and the generation of code automatically prior to deployment, thereby supporting real-time simulation. Notably, a complex gasoline engine model was integrated into the vehicle dynamics system model. This approach enables the simulation of both lateral and longitudinal driving with both manual and automatic transmissions. The main subsystem models that structure the gasoline engine are the exhaust, fuel, air path, radiator, and piston engine systems. In order to simulate the comprehensive functions of the vehicle system, a number of additional models have

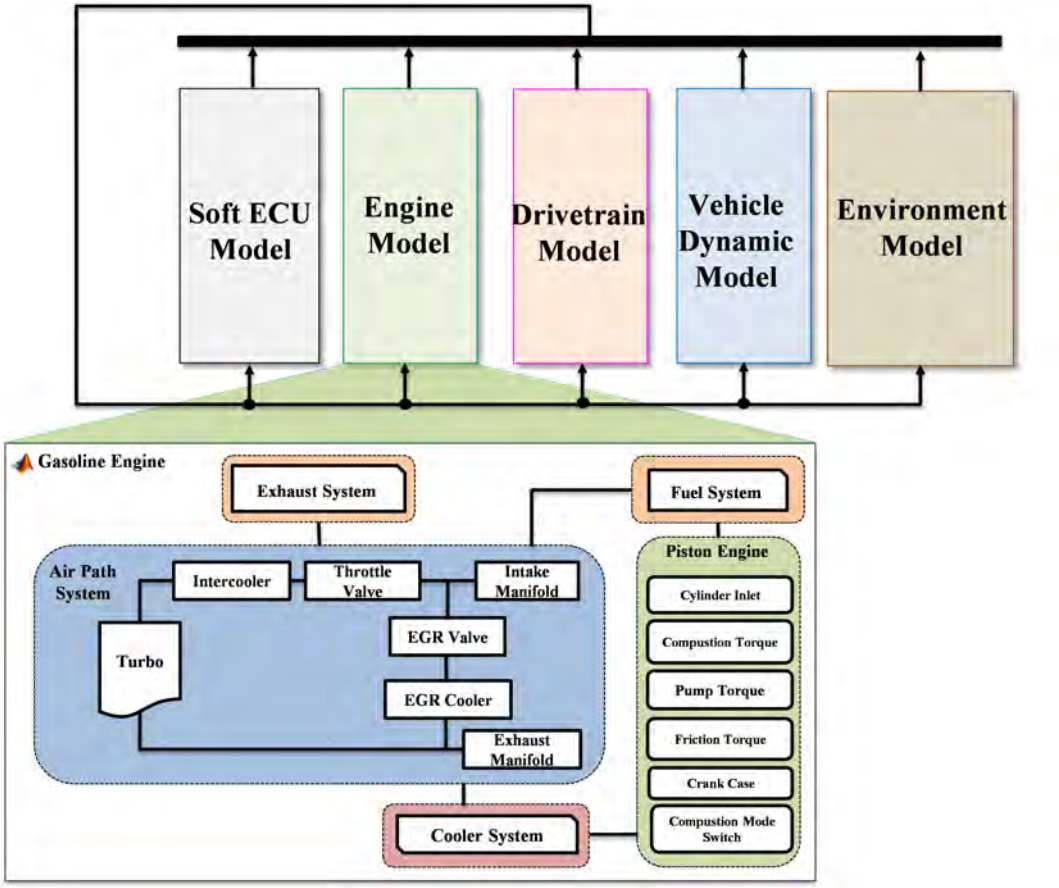
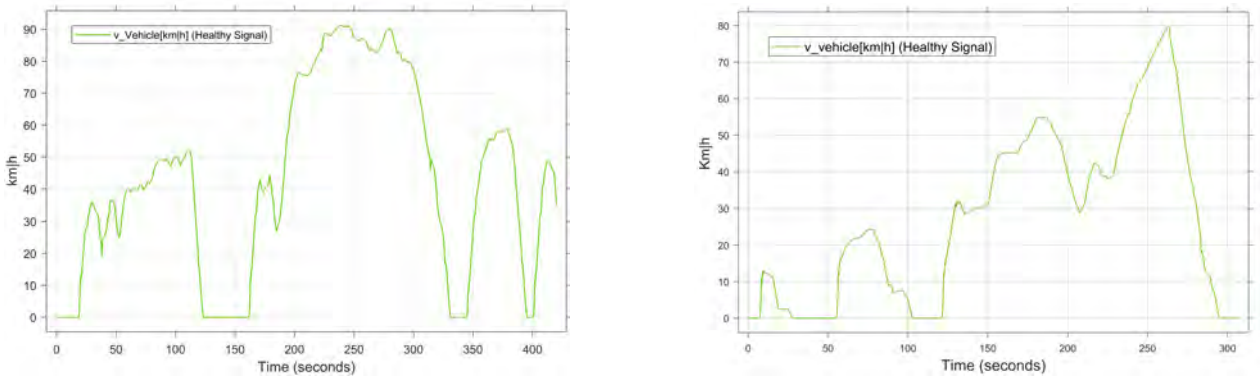


Figure 6.5: System architecture of the case study.

been included and interconnected in a block diagram environment. These include models representing the powertrain, the driver, and the environment. The control of the gasoline engine by the ECU is divided into two modes, i.e., offline and online. In the offline mode, the soft ECU is connected to the engine in an internal manner. In the online mode, the actual ECU is represented by a distinct ECU model in RCP. The interface between this ECU model and the gasoline engine model is realized by a Real Time Interface CAN multi-message blockset (RTICANMM), where the FI are configured and modeled. This RTICANMM feature allows testers to access and manipulate system signals in a programmatic manner without modifying the original system model.

6.3.2 Experimental Setup

In the context of our case study, three distinct levels of configuration can be identified, i.e., the configuration of the system model, the configuration of the experiment, and the configuration of the



(a) (b)

Figure 6.6: Driving scenario of the selected case study. (**a**) Highway driving scenario. (**b**) Ftp 75 driving scenario.

FI. At the initial stage, the parameters and variables of the models are defined through the graphical user interface (GUI) using the dSPACE software tools ModelDesk and ConfigurationDesk. In addition, the drive scenarios and TCs are designed based on the specifications. As illustrated in Figure 6.6, two driving scenarios were selected for analysis, i.e., "Highway" and "Ftp 75". This approach allows us to encompass freeway, non-urban/open road, and urban driving scenarios. A ConfigurationDesk is employed to automatically generate and deploy the code to the target machines once the system and test configurations have been specified. The study employs two distinct target machines: the MicroAutoBox II for the ECU system and the HIL SCALEXIO for the complete vehicle system. The transfer of data between the ECU and the HIL simulator is conducted via the CAN bus. The model code is automatically generated and deployed on the target machine as a consequence of the model-based development approach's capacity to generate code from the model. The generated code is compiled and transferred to the corresponding hardware, i.e., the MicroAutoBox II and the HIL simulator, using the ConfigurationDesk tool. To provide access to all system variables at runtime, the RTICANMM interface plays a pivotal role. Consequently, it is feasible to introduce faults into the target components in real-time without modifying the system architecture by integrating the FI framework with RTICANMM.

Finally, MotionDesk employs as a 3D visualization technique to represent driving maneuvers, i.e., defined scenarios. This implies that MotionDesk receives data from the HIL simulator and renders the moving objects in realistic 3D scenes. Consequently, it is possible to consider the nor-

mal and abnormal conditions of the driving environment, including the road, weather, and dynamic objects, thus providing a comprehensive understanding of the simulated system.

The fault injector configurations are established at the second level. In particular, the FI attributes are set, i.e., the location of the fault, the type of fault, and the time at which the fault is injected. As previously stated, all system signals, including sensors and actuators, are accessible via the CAN bus interface. In comparison to the traditional approach, the coverage of our framework is therefore extensive. Examples of sensor signals accessible via the CAN bus interface include crank angle, battery voltage, RPM, APP, EGR mass flow, inlet and exhaust manifold pressures, coolant temperature, and railbar sensor. The throttle valve, fuel metering unit, pressure control valve, and ignition timing adjuster are the actuators involved. The locations selected for the FI study were chosen due to the significant impact of a malfunctioning in (APP) and (RPM) sensor on vehicle behavior. Conversely, in order to illustrate the coverage of fault data generation, nine distinct types of single faults and 15 combinations were identified. In this manner, both the dataset comprising single faults and the dataset comprising concurrent faults can be collected. Finally, the timing and duration of the FI are determined based on the selected drive cycle. The injection of faults may be either permanent or transient. In the former case, the faulty component is not addressed until the fault is no longer present. In the latter case, the fault is present for a specified period of time, occurring frequently. Both balanced and unbalanced datasets can be acquired in order to develop a reliable FDD model for real-world applications by injecting permanent and transient faults.

The final level is the experimental setup, where the measurements, recordings, and instrumentation are performed. At the system level, six target system variables have been selected to represent the recorded system behavior under normal and fault conditions. As illustrated in Figure 6.7, the aforementioned system variables encompass engine speed, throttle position, intake manifold pressure, engine temperature, mean effective engine torque,rail pressure, and vehicle speed.

The system is operated under a variety of circumstances via ControlDesk, which is employed to configure and oversee the execution of experiments in real-time. The system's response to the perturbations is recorded as a time series and stored in a CSV file, which contains system variables and data samples after the real-time execution is complete. In order to generate the documented failure logs in textual form, the TCs designed in ModelDesk are executed on the target SUT. It is recommended that TCs include data inputs and expected outputs. In order to ensure the genera-

	A	B	C	D	E	F	G	H
1	Time[sec]	Pos_Throttle[%]	T_Out_Engine[degC]	Trq_meanEff_Engine[Nm]	n_Engine[rpm]	p_InMan[Pa]	p_Rail[bar]	v_Vehicle[km|h]
2	0	1.2864	753.3748	1.8067	736.7779	22732.8420	120.2911	$3.3791420517 \times 10^{-41}$
3	0.001	1.2835	753.3705	1.8066	736.1490	22716.2393	120.2894	$3.3453506312 \times 10^{-41}$
4	0.002	1.2813	753.3660	1.8148	735.6940	22699.6510	120.2871	$3.3118971249 \times 10^{-41}$
5	0.003	1.2797	753.3615	1.8233	735.4068	22683.0887	120.2845	$3.2787781536 \times 10^{-41}$
6	0.004	1.2790	753.3569	1.8321	735.2794	22666.5553	120.2816	$3.2459903721 \times 10^{-41}$
7	0.005	1.2792	753.3523	1.8413	735.3018	22650.0573	120.2787	$3.2135304683 \times 10^{-41}$
8	0.006	1.2804	753.3476	1.8506	735.4619	22633.6030	120.2758	$3.1813951637 \times 10^{-41}$
9	0.007	1.2824	753.3430	1.8599	735.7450	22617.2017	120.2730	$3.1495812120 \times 10^{-41}$
10	0.008	1.2852	753.3385	1.8689	736.1341	22600.8620	120.2703	$3.1180853999 \times 10^{-41}$
11	0.009	1.2888	753.3339	1.8775	736.6096	22584.5915	120.2678	$3.0869045459 \times 10^{-41}$
12	0.01	1.2928	753.3295	1.8854	737.1497	22568.3958	120.2657	$3.0560355005 \times 10^{-41}$
13	0.011	1.2971	753.3252	1.8926	737.7305	22552.2781	120.2638	$3.0254751455 \times 10^{-41}$
14	0.012	1.3016	753.3209	1.8988	738.3263	22536.2393	120.2623	$2.9952203940 \times 10^{-41}$

Figure 6.7: Generated time series dataset.

tion of accurate and comprehensive failure logs, the TCs must include both the data inputs and the expected outputs. The specified output expectations are then compared with the actual output measurements, contingent on the actual system behavior. The outcome of the test is either a pass or a fail, contingent on the manner in which the test is executed. The automated testing functionality of the ModelDesk ASM enables the execution and evaluation of TCs to be carried out automatically. In addition to the execution results (pass or fail), the generated report contains important information about the specifications of the TCs. Furthermore, the report includes an fault log, which describes the fault that occurred. The failure logs of TCs can be utilized to develop an intelligent model for root cause analysis (RCA). This model is based on natural language processing and ML methods.

6.3.3 Configuration of Fault Injection Experiment

The results of this study demonstrate the effects of both permanent and transient faults. A single or multiple fault is introduced according to a predefined configuration in the context of a permanent fault. Tables 6.1 and 6.2 present the configuration of the permanent single and multiple fault injector, including the primary attributes of the FI framework. These attributes include fault location, fault type, and fault time. Conversely, Table 6.3 provides an overview of the configuration of the transient faults.

Fault Type	Fault Location	Fault Time	Fault Value
Gain	Acceleration Pedal Position Sensor	5	10
Offset	Engine RPM Sensor	24	1800
Noise	Acceleration Pedal Position Sensor	5	0–100
Packet Loss	Rail Bar Sensor	120	0
Stuck-at	Engine RPM Sensor	5	0
Stuck-at	Ignition Angle Control Signal	131	0
Drift	Acceleration Pedal Position Sensor	30	1
Hard-Over	Acceleration Pedal Position Sensor	36	127
Spike	Mass Flow Through Throttle Sensor	0	1–510
Delay	Engine RPM Sensor	120	5

Table 6.1: Configuration of permanent single fault injector.

Fault Type	Fault Location	Fault Time	Fault Value
Noise	Acceleration Pedal Position Sensor	25	0–100
Stuck-at	Engine RPM Sensor	35	0

Table 6.2: Configuration of permanent simultaneous fault injector.

Fault Type	Fault Location	Inject Time	Eject Time	Fault Value
Stuck-at	Engine RPM Sensor	40	160	0

Table 6.3: Configuration of transient fault injector.

6.4 Results and Discussion

This section outlines the findings of the data collection conducted for a range of FI configurations. In particular, the generated data are described in detail, with particular attention paid to the effects of single and concurrent critical faults on system behavior. Furthermore, the log data of failed TCs generated in the test report is presented.

6.4.1 Fault Effect Analysis on System Behavior

This section presents the results of the application of the real-time FI framework proposed in this paper. The selected faults were injected based on their duration of occurrence, categorised as persistent (single and concurrent) and transient faults in different modes. This was done in order to analyze the effects of the faults on the system behavior in a real-time simulation.

Real-Time Permanent Single Fault Injection

A single fault implies that there is a single variable that is manipulated and forced to its extreme value. All the faults defined in Table 6.1 are injected into the corresponding location at the specified time in order to observe the reaction of the system in the presence of faults in the sensors and control signals of the actuators. The accelerator pedal position sensor was selected as the location for the fault. The sensor is subjected to a variety of fault types at different stages of the driving scenario. The gain fault with d_v a value of 10 in the Equation (2.1) is injected at 5 sec on the mentioned sensor signal. The results are recorded at the system level where engine RPM (Figure 6.8a) and vehicle speed (Figure 6.9a) are observed. When the driver attempts to regulate the speed of the vehicle by adjusting the position of the accelerator pedal, this adjustment results in fluctuations in the vehicle speed and engine speed. This noise can be observed in Figures 6.8a and 6.9a between 135s and 145s, as well as in numerous other analogous scenarios. The noise fault is also introduced at 5 seconds into the driving scenario, when the vehicle speed is 0 and the gear stick is in neutral. Following the injection of the fault, the system-level engine speed values exhibited a range of between 2200 and 2500 revolutions per minute (RPM), in contrast to the normal (fault-free) case, where the values ranged between 700 and 800 RPM. This results in a higher energy consumption.

Furthermore, it is observed that the engine speed exhibits some irregularities when the driver attempts to maintain the vehicle's speed. Figure 6.8c shows the effect of the noise fault on the engine RPM. In contrast, the impact of the fault on vehicle speed was minimal. At the outset of the driving scenario, the drift fault is injected. At 30 seconds, with the engine in neutral, the effects of the fault on the system-level engine RPM values are observed (Figure 6.8f). As shown in Figure 6.8f, the engine RPM values start to increase slowly. Furthermore, a slight increase of the vehicle speed is observed at 85 sec to 90 sec and between 130 sec and 150 sec, as shown on 6.9d. The HardOver fault (Figure 6.8g) is injected at 36 s, and the effects of this fault are immediately observed on the system-level engine speed values. The values exhibit a sudden increase from 700 to 800 rpm to 2200 to 2500 rpm. The system values for the hard-over fault are strikingly similar to those for the noise and drifting faults observed in the 90s to 120s, yet they exhibit slight differences. The engine speed values following the occurrence of a drift fault are observed to be between 2200 and 2400 rpm, while they reach up to 2500 rpm for the hard-over and noise faults. In addition, the engine speed is more pronounced in the case of a hard override than in the case of a noise override. For instance, it is challenging to ascertain the nature of the fault from the engine speed measurements after 290 seconds.

Offset and stuck-at faults are introduced at 24 seconds and 5 seconds, respectively, in the driving scenario to analyze the impact of a potential fault in the engine speed sensor on the system behavior. An offset fault with a o_v value of 1800 is injected in Equation (2.1) when the vehicle's gearshift knob is in the neutral position and the vehicle speed is 0. As can be seen in Figure 6.8b, within one second of the fault appearing, the engine stops immediately and the value of the engine speed becomes 0. In the same context, the stalling fault is also injected with d_v and o_v values of 0 in Equation (2.1). At the time of injection, the engine is running at a vehicle speed of approximately 10 km/h. The consequence of this malfunction is not apparent at this moment; however, as soon as the engine is placed in neutral, the engine speed increases to 2400 RPM at the system level. In addition, as shown in Figures 6.8e and 6.9c, a delay in the behavior of both the RPM and speed signals can be observed as the vehicle begins to accelerate at 50 s.

This study also considers delay and packet loss faults as communication fault categories. Consequently, a delay fault is introduced into the engine speed sensor at 120 seconds. The impact on both system-level values of engine speed and vehicle speed is minimal. As illustrated in Fig. 6.8i,

a slight noise is discernible in the engine speed between 210 s and 240 s, and its impact can also be observed in the vehicle speed (Fig. 6.9f). Figures 6.8d and 6.9b show significant fluctuations and jumps in the engine and vehicle speed signals. Given that the packet loss time for the entire driving scenario is less than two seconds, it can be concluded that the packets are lost for three seconds, resulting in a complete loss with instantaneous engine shutdown. At the point of mass flow through the throttle, the sensor spike fault is introduced at the outset of the driving scenario. As the vehicle begins to accelerate at 55–60s, a slight decrease in engine RPM is observed (Figure 6.8h), while no significant effect on vehicle speed is observed except for a sharp increase at 58s and 62s (Figure 6.9e). The engine speed remains constant between 90 and 120 seconds, although when the vehicle is accelerating, the engine abruptly ceases operation. At 121 seconds, both the RPM and speed system-level values are recorded as 0.

Real-Time Permanent Simultaneous Faults Injection

The FI framework permits the injection of multiple faults into sensor and actuator control signals. This is achieved by manipulating multiple variables and assigning extreme values, thereby enabling the effect of combining multiple independent faults to be analysed. In this study, two distinct types of faults at disparate locations and times are introduced into the sensor signals. At 35 seconds, an accelerator pedal fault is injected into an engine speed sensor where the values of the variables d_v and o_v are 0, as specified in Equation 2.1. Immediately after that, the effect of both faults is observed in the engine RPM, as shown in Figure 6.10a. It can be observed that the effects of the injected faults are identical to those of a stuck fault in the engine speed sensor.

On the other hand, it is worth noting that injecting multiple faults in the actuator control signals has no significant effect on the engine and vehicle speeds compared to the healthy signals. However, a slight difference is observed in the intake manifold between 155 sec and 190 sec when the stall fault with d_v and o_v values of 0 (Equation 2.1) is injected into the ignition timing control signal for all cylinders at 131 sec of the driving scenario, as shown in Figure 6.11, which illustrates the effect of the stuck fault on the pressure in the intake manifold signal.

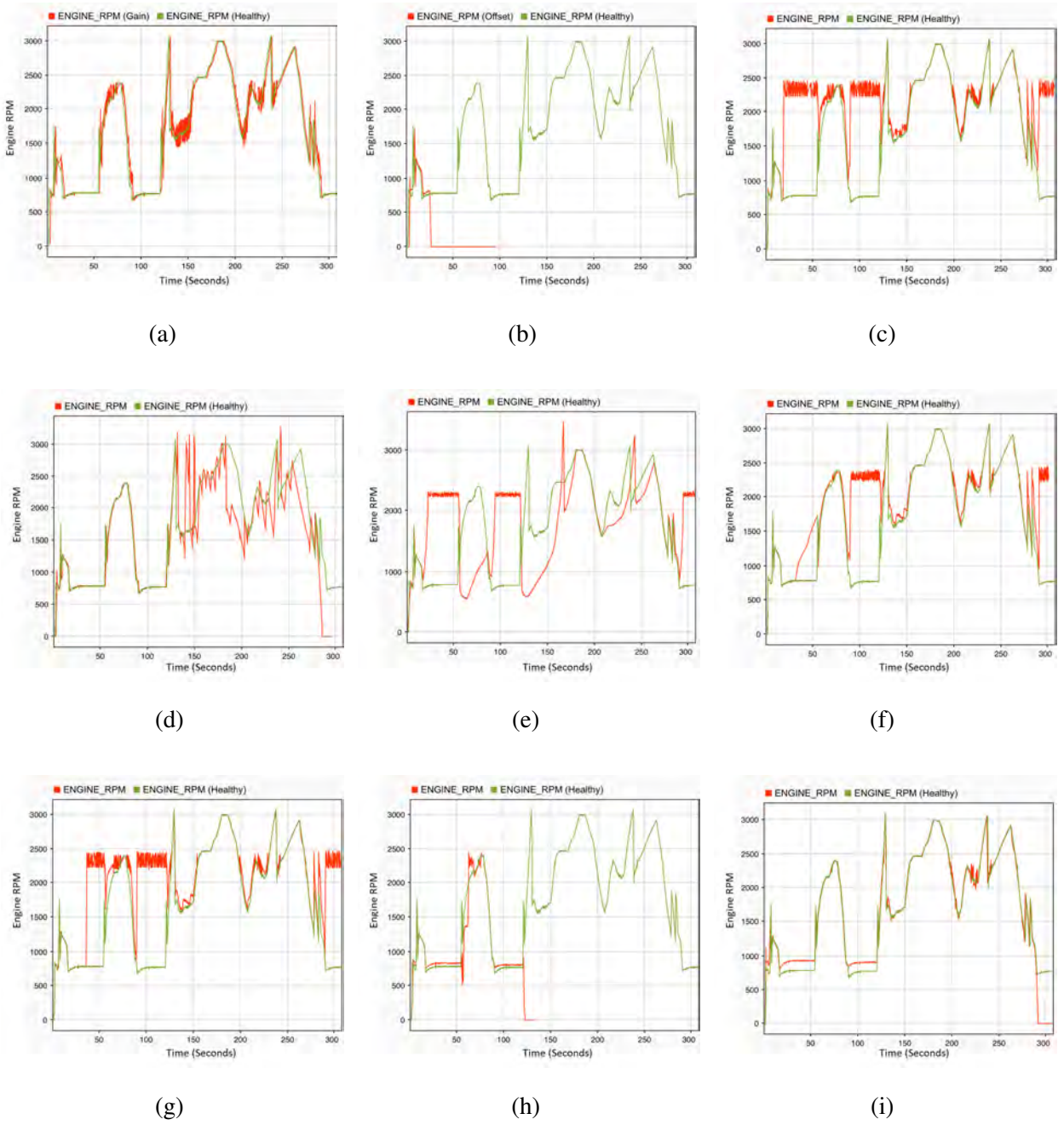


Figure 6.8: System output: engine RPM with single permanent fault injection. (**a**) Acceleration pedal with gain fault. (**b**) Engine RPM sensor with offset fault. (**c**) Acceleration pedal with noise fault. (**d**) Rail bar sensor with packet loss fault. (**e**) Engine RPM sensor with stuck-at fault. (**f**) Acceleration pedal with drift fault. (**g**) Acceleration pedal with hard-over fault. (**h**) Mass flow through throttle with spike fault. (**i**) Engine RPM sensor with delay fault.

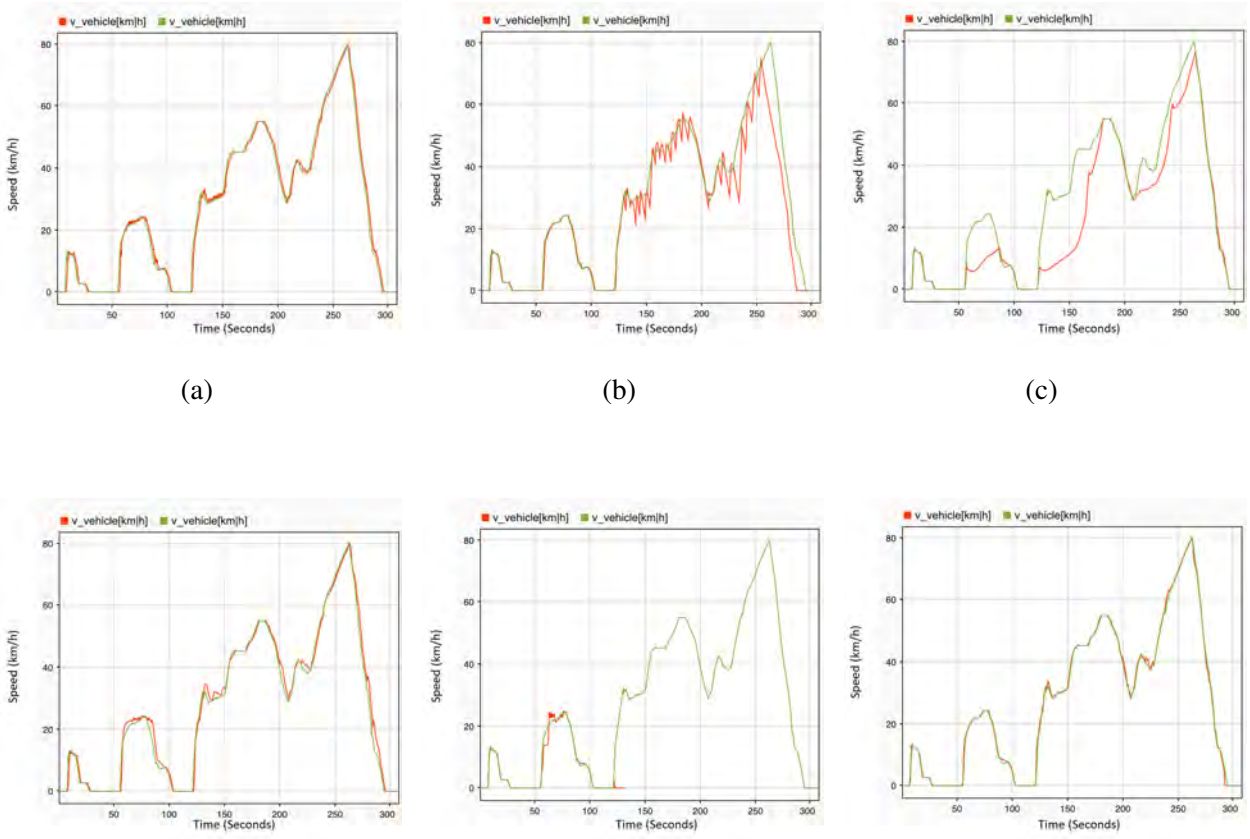
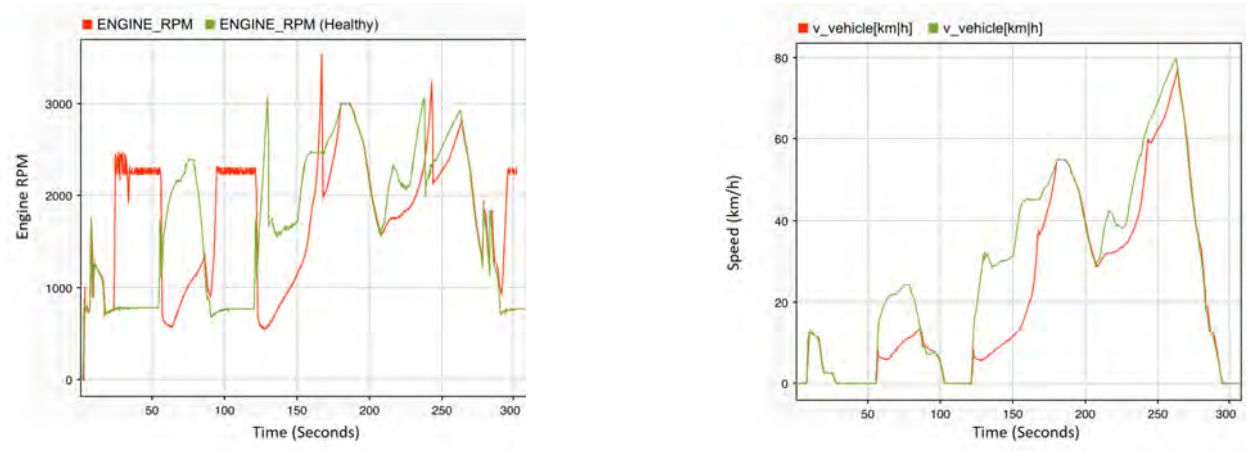


Figure 6.9: System output: vehicle speed under single permanent fault injection. (**a**) Acceleration pedal with gain fault. (**b**) Rail bar sensor packet with loss fault. (**c**) Engine RPM sensor with stuck-at fault. (**d**) Acceleration pedal with drift fault. (**e**) Mass flow through throttle with spike fault. (**f**) Engine RPM sensor with delay fault.



(a) (b)

Figure 6.10: System output: engine RPM and vehicle speed under multiple permanent faults injection. (**a**) Engine RPM with stuck-at fault and noise fault. (**b**) Vehicle speed with stuck-at fault and noise fault.

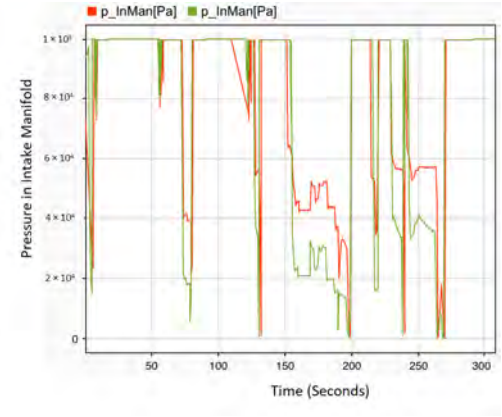
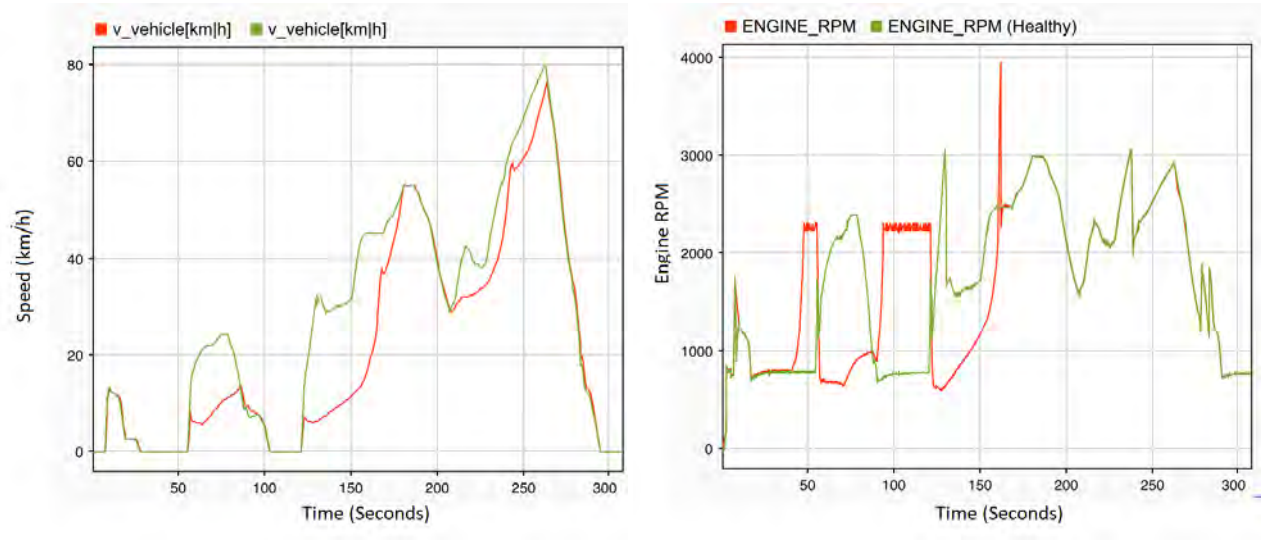


Figure 6.11: Intake manifold with ignition angle control signal stuck-at fault injection.

Real-Time Transient Fault Injection

Due to the difficulty in identifying transient faults, which are less predictable than permanent faults, the impact of these faults on the dynamic behavior of complex vehicles is investigated. For this purpose, the stuck-at fault with d_v and o_v values of 0 is injected as a transient fault into the engine speed sensor for a period of 160 s at 40 s intervals. Figure 6.12 shows the effect of the fault on the system response, where the stuck fault with d_v and o_v values of 0 (Equation 2.1) is injected as a transient fault into the engine speed sensor. The duration of the transient fault is 40-160

seconds. The results indicate that the effect is comparable to that of a permanent fault. However, when the fault is removed, a sharp increase in both engine speed (Figure 6.12b) and vehicle speed (Figure 6.12a) is observed at time 160s. Upon the removal of the fault, the system resumes its normal state and continue in a fault-free mode.



(a) (b)

Figure 6.12: System output: vehicle speed and engine RPM under transient faults injection. (**a**) Vehicle speed sensor with transient stuck-at fault. (**b**) Engine RPM sensor with transient stuck-at fault.

6.4.2 Representative Time Series Dataset

In accordance with the aforementioned system and FI configurations, both the SUT and the control system are executed in real-time under both normal and abnormal conditions. A total of 40 experiments were conducted in two driving scenarios, with single and simultaneous faults. In particular, two files were collected as representative data from the "highway" and "ftp 75" scenarios. Each file consisted of 18 data points for each type of fault, with a total of 30 files for simultaneous faults with 14 combinations. Consequently, 50 CSV files were generated as HIL test records. The total number of faulty data samples from the data collection phase, considering six types of single faults

and sampling time 0.001 sec, is 947,000 with 420,000 samples for each experiment, as shown in the table 6.4. In contrast, the total number of faulty data samples from the parallel FI process is 2,267,000,000, with 420,000 samples for each experiment, as shown in table 6.5.

Table 6.4: Collected dataset description of the single faults.

Fault ID	Fault Type	Fault Duration	Dataset Samples	Faulty Samples
H	Healthy	-	420,000	-
F1	Gain	165–320 s	420,000	155,000
F2	Stuck-at	172–330 s	420,000	158,000
F3	Noise	175–310 s	420,000	135,000
F4	Packet loss	170–332 s	420,000	162,000
F5	Delay	170–326 s	420,000	156,000
F6	Drift	164–345 s	420,000	181,000

Table 6.5: Collected dataset description of the simultaneous faults

Fault ID	Fault Type	Fault Duration	Dataset Samples	Faulty Samples
F1F2	Gain and Stuck-at	166–338 s	420,000	172,000
F1F3	Gain and Noise	164–340 s	420,000	176,000
F1F4	Gain and Packet loss	170–329 s	420,000	159,000
F1F5	Gain and Delay	177–334 s	420,000	157,000
F1F6	Gain and Drift	164–324 s	420,000	160,000
F2F3	Stuck-at and Noise	174–330 s	420,000	156,000
F2F4	Stuck-at and Packet loss	164–327 s	420,000	163,000
F2F5	Stuck-at and Delay	173–324 s	420,000	151,000
F2F6	Stuck-at and Drift	178–342 s	420,000	164,000
F3F4	Noise and Packet loss	166–320 s	420,000	154,000
F3F5	Noise and Delay	179–328 s	420,000	149,000
F3F6	Noise and Drift	169–341 s	420,000	172,000
F4F5	Packet loss and Delay	176–336 s	420,000	160,000
F4F6	Packet loss and Drift	166–340 s	420,000	174,000

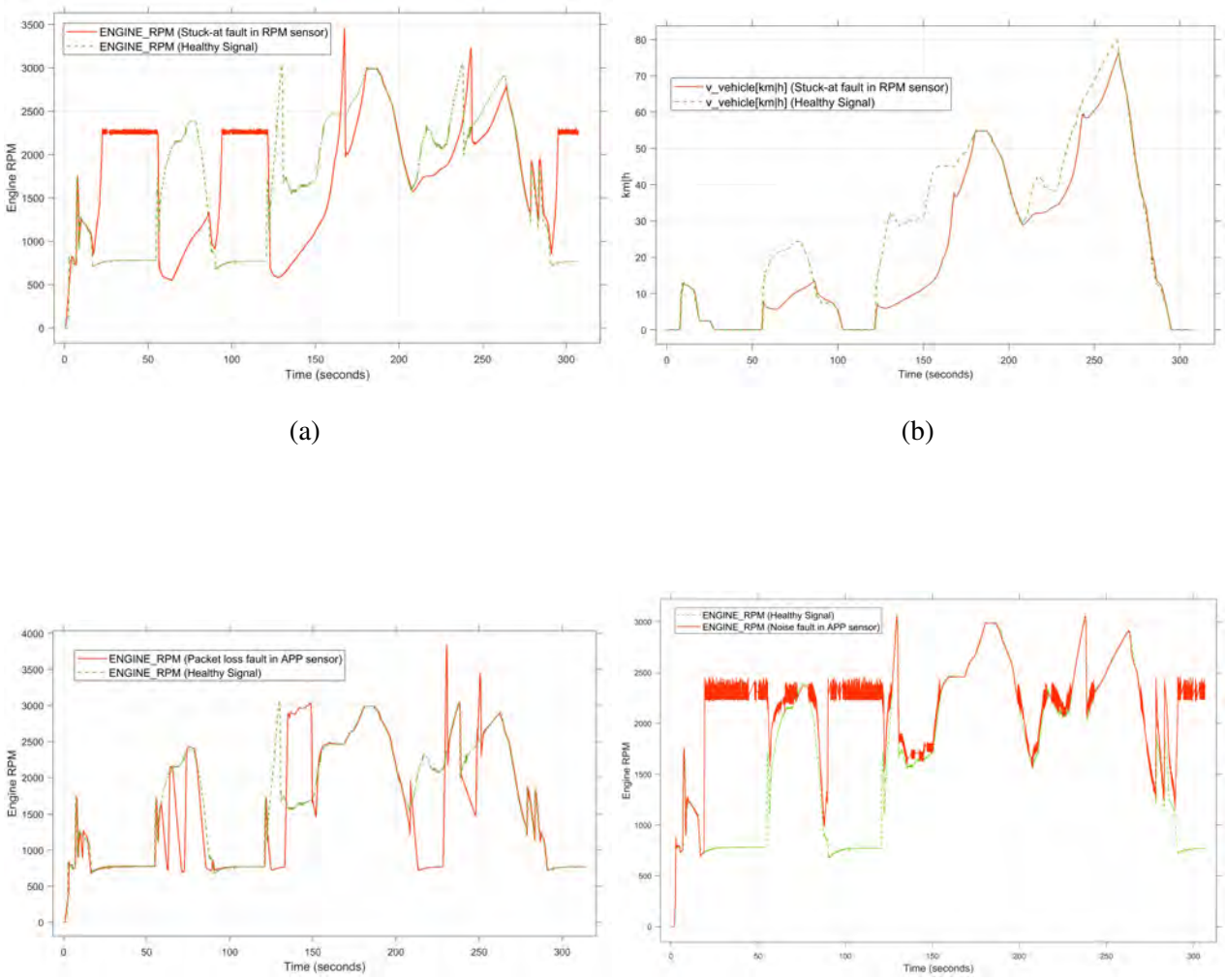
The system behavior under the injections is recorded at the system level on vehicle and engine speed signals for the purposes of illustrating the system's response to the faults of packet loss and stuck-at. In Figure 6.13a, the stuck-at fault has been injected into the speed sensor after 10 s of

the driving cycle. The engine's behavior undergoes a notable transformation from 750 rpm in the normal operating range to 2300 rpm when the fault occurs, resulting in a discernible increase in energy consumption. For a period of up to 180 seconds, the control unit is unable to cope with the fault that has occurred. Subsequently, the system attempts to rectify the fault and resume normal operation. Subsequently, deviations and fluctuations recur from 210 seconds and 290 to 300 seconds, respectively. The fluctuations and conditions associated with driving in acceleration, deceleration, and steady-state modes are the underlying cause of this change. This indicates that the control strategy is unable to effectively mitigate faults during rapid acceleration and deceleration, as illustrated in Figure 6.13b.

On the other hand, by applying the packet loss to the APP sensor during the driving cycle, the effect of the fault on vehicle speed and engine speed is not directly observable, as illustrated in Figure 6.13c. Nevertheless, notable fluctuations can be discerned from the 75th second onwards. Nevertheless, it is evident that notable fluctuations become apparent from the 75th second onwards. At the system level, the vehicle and engine exhibit behavior that differs from the standard in terms of strong jerk. At this juncture, the ECU endeavors to overcome the signal losses for a specified duration. This is due to the fact that while the duration of the signal loss is less than three seconds, the vehicle's behavior remains in a state of fluctuation.

The same effect may be observed by injecting noise into the APP sensor at 25 seconds in the form of a strong amplitude variation in the speed signals (see Figure 6.13c). The effect is particularly evident during the stationary period of the vehicle speed, i.e. between 20 - 55, 80 - 125 and 280 - 330 seconds. Conversely, during the driving mode, the SUT attempts to exhibit the desired driving behavior despite the occurrence of the fault. Consequently, within an acceptable range of the RPM signal, the system adheres to the predefined scenario between 125 and 280 seconds.

Following the "ftp 75" scenario, the impact of a single FI into the APP and RPM sensors, respectively, is illustrated in Figures 6.14a,b of the Validation Accuracy section. In particular, the APP sensor signal is manipulated by the stuck-at value, and the effect of the fault is directly observed at the system level as a constant value, i.e., 1160 rpm. Conversely, as illustrated in Figure 6.14b, the impact of a delay in the RPM sensor signal results in fluctuations in engine speed, ranging from 265 to 307 seconds. During this period, as the delay in the received signal increases, the ECU



(a) (b)

(c) (d)

Figure 6.13: Effect of the single faults on the system Behavior. (**a**)Engine system Behavior under stuck-at fault. (**b**) Vehicle system Behavior under stuck-at fault. (**c**) Engine system Behavior under packet loss fault. (**d**) Engine system Behavior under noise fault.

may be unable to perform the desired operation optimally, particularly when transitioning between driving modes.

Simultaneous faults are defined as the injection of two different types of faults into two different locations simultaneously. For example, while injecting the stuck-at fault into the APP sensor, injecting the delay fault into the RPM sensor at the same time. In other words, two or more factors have contributed to creating a new pattern in the signals as a result of the simultaneous FI. Figure 6.14c illustrates the impact of simultaneous faults on the vehicle system response, specifically the vehicle speed, between 170 and 330 s. The vehicle speed closely aligns with the desired response up to 170 s, the time at which the faults are initiated. At 190 s, the speed drops to 18 km/h, and the vehicle is unable to accelerate due to the large fluctuations shown in Figure 6.14d. This places the vehicle in danger of failing. However, the system returns to an acceptable state with small deviations.

The generated CSV files contain redundant or superfluous information, despite the fact that the collected data represent a comprehensive set of characteristics of the system behavior. Each file contains, for instance, information regarding the recording procedure and the data patterns. Consequently, the collected data should be subjected to pre-processing in order to eliminate outliers and redundant information. In the preprocessing phase, three distinct steps are applied to the data. The following steps are employed to preprocess the data: the removal of useless information and outliers, scaling and normalizing the data, and splitting the data. Subsequently, the training variables are selected and the useless data removed after the CSV files have been converted to Excel format. This process ensures the availability of useful samples for the development of the ML model. The range of values for each variable is distinct. Consequently, the subsequent step is to apply a normalization function to all variables. Consequently, all values of the variables are scaled within the range between [0, 1]. Finally, the data should be divided into three distinct categories: training data, validation data, and test data. It should be noted that the process of labeling the data should be conducted prior to the data being split when developing ML models based on supervised learning. This entails the assignment of the appropriate fault class to the data to be classified. In general, the data is divided into three categories: 80% for training, 10% for validation, and 10% for testing.

As illustrated in Table 6.6, in order to assess the efficacy of the proposed framework, the characteristics of the real-time dataset generated in this study were compared with those of the existing

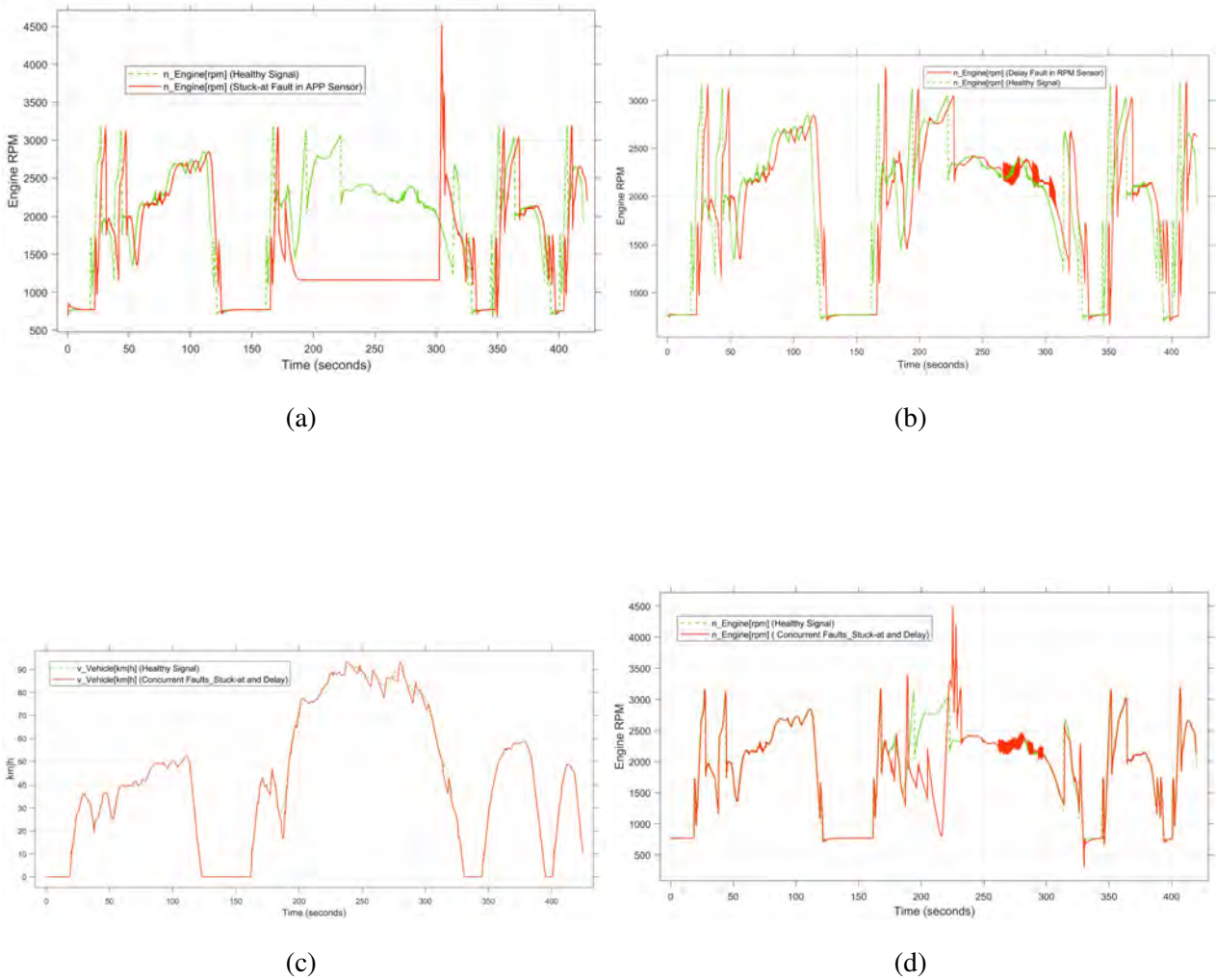


Figure 6.14: Effect of the simultaneous faults on the system behavior. (**a**) Engine system behavior under stuck-at fault. (**b**) Engine system behavior under delay fault. (**c**) Vehicle system behavior under stuck-at and delay faults simultaneously. (**d**) Engine system behavior under stuck-at and delay faults simultaneously.

datasets from the existing systems. It is observed that time series data collected from the system components are provided by the majority of existing simulation systems. However, the datasets provided, such as CMAPSS, CarMaker, and IIoT datasets, have not considered the occurrence of concurrent faults. In contrast, the datasets collected in our proposed work were obtained under both single and simultaneous fault conditions in real-time. In particular, the A2D2 dataset includes only single and concurrent fault examples. The datasets in question are limited to four types of faults in specific driving situations that are not representative of real-life driving. The proposed work has considered the effects of adverse weather and lighting conditions, as well as the simulation of various sensor and actuator fault scenarios.

Table 6.6: A comparison of available datasets from the existing systems with the proposed dataset.

Dataset	Year	Type	Size	Label	Faults	Scenarios
CMAPSS [174]	2008	Time series data	76 kB	Yes	Single	Limited
A2D2 [214]	2020	Time series data	657 kB	Yes	Single, concurrent	Limited
CarMaker [221]	2021	Time series data	3 MB	Yes	Single	Limited
IIoT [217]	2019	Time series, logs	-	Yes	Single	Diverse
CarFree [222]	2022	Images	5 kB	Yes	-	Diverse
TrainSim [223]	2023	Images, point clouds	8 kB	Yes	-	Diverse
Proposed work	2024	Time series and logs	8.8 MB	Yes	Single, concurrent	Diverse

6.4.3 Automated Test Evaluation

The proposed framework employs the use of AutomationDesk for the purpose of automatic FI and test case analysis. Prior to the execution of the test, it is necessary to determine the expected output of the SUT. Subsequently, the actual behavior is compared with the expected behavior by the evaluation function. In the event of a discrepancy between the two outputs, the executed TCs are deemed to have failed. As demonstrated in Figure 6.15, the measured output from the SUT exceeds the predefined tolerance range, indicating that the vehicle position could not be corrected, which resulted in a failure. In other words, a system-level failure is defined as the inability of the required functions to be performed due to the propagation of the fault from the component through the subsystems to other components. This tool facilitates the automated execution and assessment

of hundreds of TCs, obviating the necessity for human intervention. Moreover, the results are documented in a report, which plays a pivotal role in the identification of critical faults. It is important to note that not every fault that is introduced necessarily results in a failure at the system level; some faults may have a more limited impact on the system's ability to perform its required functions. Nevertheless, the fault tolerance mechanism mitigates the impact of these faults. A critical fault is defined as one that results in the termination of system functions. The system's behavior resulting from the introduction of critical faults is recorded as a representative set of fault data, which serves as input for the construction of the FDD model.

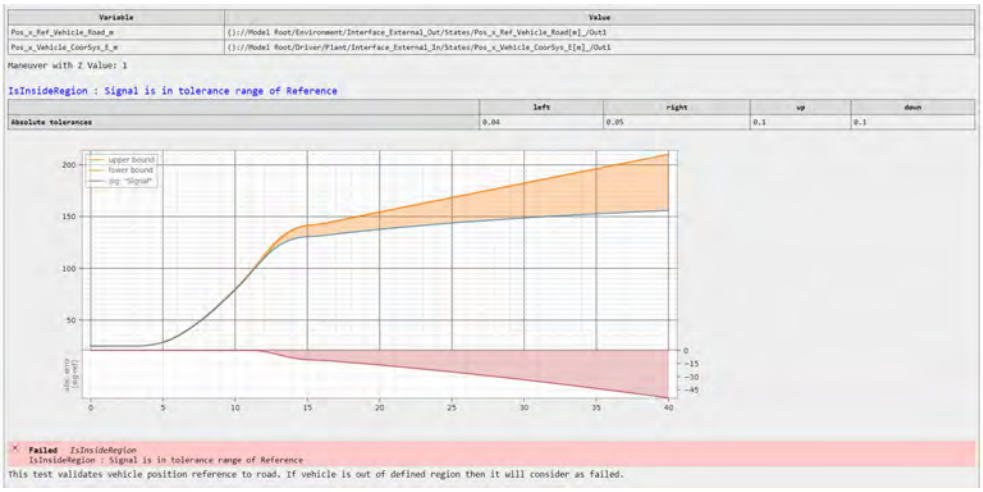
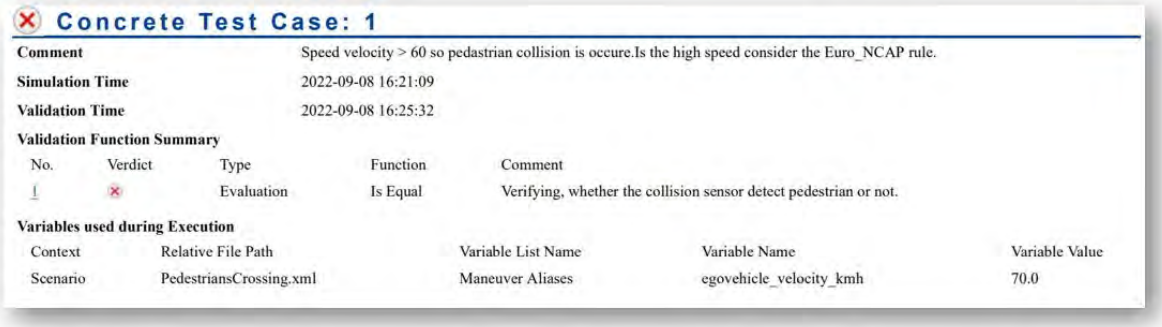


Figure 6.15: Evaluation of the test cases' execution.

6.4.4 Representative Textual Dataset

The test report is generated by running the TCs and comparing the desired system signals with the observed results. The natural language logs, which describe the faults that occurred, are of interest in the report. Figure 6.16 depicts an exemplar of a generated test report. As illustrated, a brief text is generated to convey the failure of the TCs and the underlying cause. It is, however, important to note that these log files cannot identify failure causes accurately without the expert knowledge of the tester. It can thus be concluded that a model can be trained on the basis of the collected logs in order to perform the RCA process in an efficient manner, with the assistance of no further human input required. Moreover, the generated logs can furnish supplementary data that can be employed to inform the decision-making process of the signal-based FDD model. Prior to

model training, a number of preprocessing steps, analogous to those applied to time series data, are implemented on the collected logs. The primary preprocessing steps for text data are tokenization, normalization, stop word removal, and stemming. The training phase is divided into three distinct parts: 80% training, 10% validation, and 10% testing. Subsequently, natural language processing (NLP) techniques are applied to the collected data.



Concrete Test Case: 1

Comment	Speed velocity > 60 so pedastrian collision is occure.Is the high speed consider the Euro_NCAP rule.
Simulation Time	2022-09-08 16:21:09
Validation Time	2022-09-08 16:25:32

Validation Function Summary

No.	Verdict	Type	Function	Comment
1	✗	Evaluation	Is Equal	Verifying, whether the collision sensor detect pedestrian or not.

Variables used during Execution

Context	Relative File Path	Variable List Name	Variable Name	Variable Value
Scenario	PedestriansCrossing.xml	Maneuver Aliases	egovehicle_velocity_kmh	70.0

Figure 6.16: Generated report of test case execution.

6.4.5 Setup Cost Discussion

It is crucial to acknowledge the potential complexities associated with the setup and implementation of the proposed framework, despite its demonstrated reliability in simulating realistic system behavior in the presence of faults. A virtually unlimited number of potential configurations or fault scenarios can be generated, as three distinct fault types must be defined during the setup process. For complex software systems, the challenge of achieving an optimal balance between comprehensive testing and the identification of critical faults remains. In particular, this research identified fault TCs based on the following criteria: the representative and realistic nature of injectable faults, the impact and speed of the injectable faults, and the variety of fault scenarios. Moreover, in addition to the time and resources that were required to set up the framework, it was necessary to implement a real-time system to execute the system model in real-time alongside the framework. Accordingly, a MicroAutoBox II embedded PC (DS1401 baseboard) with a 900 MHz processor, 6th Gen. Intel Core i7-6822EQ, and a boot time of 340 ms for a 3 MB application was employed. The Core i7-6822EQ processor, with 16 MB of memory and a boot time of 340 ms, is capable of executing a 3-MB application. The maximum execution time for a task is 0.738 ms. This value

was derived from the debugging execution times for the 10 s protocol, which indicated an intended sampling period of 1 ms. The mean execution time for each test case is 40,242.6164 milliseconds, inclusive of the evaluation period.

Finally, it is now possible to collect very large volumes of data, which presents an additional challenge to the computational cost of developing ML and DL models. In particular, if we consider a sampling period of 0.001 seconds, the generation of several million data samples results in the inclusion of anomalies, noise, and missing or redundant samples. Consequently, in addition to the significant investment of time and resources required for preprocessing and training the ML/DL model, the deployment of high-performance computing infrastructure, such as GPUs or TPUs, is essential to process the vast quantities of data generated.

6.5 Conclusions

The objective of the study in this chapter is twofold. First, it aims to propose a novel framework for identifying critical faults and collecting representative, real-time datasets for ML/DL applications during the system validation process of ASSs. Second, it aims to demonstrate the effectiveness of our framework through the use of HIL simulation under real-time constraints. This is accomplished through the utilization of HIL simulation under real-time constraints and an automated FI method. The objective of this proposed framework is to provide an essential dataset that can be employed in the training, testing and validation of DL-based intelligent fault analysis of HIL datasets. In a novel approach, the FI methodology for data acquisition does not necessitate the construction of a fault model within the system architectural framework or the physical hardware device. In contrast, the target faults are injected programmatically in real-time, thus eliminating the necessity for any modeling effort or modeling time. Moreover, the faults are injected and evaluated in an automated manner without modifying the SUT model, using an automation tool, i.e., AutomationDesk. The proposed framework has the capability to simulate and collect fault data across a range of formats, including time-series and textual formats. To validate and demonstrate the proposed approach, a complex automotive gasoline engine was selected for this study. The findings demonstrate that the proposed approach is capable of accurately representing the effects of both individual and concurrent faults on a system level. In this study, a total of nine different types of sensor and actuator faults

were simulated. Two faults were simulated that occurred simultaneously at different locations. It can thus be noted that during real-time simulations, it is possible to collect a high-quality dataset with high fault class coverage with minimal effort. The datasets are employed to develop novel DL models for a range of tasks and applications, including fault detection, diagnostics, prognostics, and RCA.

7 Hybrid Deep Learning Methods-based Intelligent FDD

In the previous chapter, the critical faults that result in a violation of system functionality were determined. The representative dataset capturing the system behavior in case of the mentioned critical faults was collected. In this chapter, the collected data is used as the basis for answering research question RQ2. In particular, a data-driven hybrid DL method is proposed in this chapter to address the challenge of FDD during the real-time validation process of ASSs. The chapter is organized in such a way that an overview is given in Section 7.1. Subsequently, the architecture and phases of the proposed method are presented in Section 7.2. The detailed description of the dataset used and the experimental setup is presented in Section 7.3. Section 7.4 demonstrates the superiority and performance of the proposed method compared to a standalone traditional method. Finally, Section 7.5 summarizes the results and findings of this chapter.

Note: It is important to highlight that the proposed methodology and the corresponding implementation outcomes presented in this chapter have been published in [279].

7.1 Chapter Overview

The ISO 26262 standard has identified the use of HIL as an essential testing methodology for the assessment of ASSs in terms of their safety and reliability characteristics. Nevertheless, the voluminous data HIL generates throughout the testing process, renders the conventional methods of fault detection and classification, based on the input of human experts, infeasible. It is thus clear that the creation of an effective methodology based on the historical dataset is required in order to analyze the records of the testing process in an efficient manner. Although data-driven fault diagnosis has been found to be more effective than other diagnostic approaches, selecting the optimal technique from the extensive array of DL techniques remains challenging. Moreover, the training

data containing automotive faults are scarce and considered highly confidential by the automotive industry. This chapter presents a novel intelligent FDD model that employs hybrid DL techniques, which represent a significant departure from conventional methods. The model is designed for use during the V-cycle development process, particularly during the system integration testing phase. In this regard, HIL based real-time FI framework (developed in Chapter 6) is employed to facilitate the generation of faulty data without any alteration to the original system model. Furthermore, a combination of the CNN and LSTM is employed to construct the model structure. In this study, eight types of sensor faults are considered to encompass the most prevalent potential faults in the signals of ASSs. To illustrate the efficacy and benefits of the proposed methodology, a gasoline engine system model with the developed framework (in Chapter 10) is employed as a case study. This provides a means of validating the model's performance while also showcasing the advantages and capabilities of the proposed approach. These results indicate that the proposed method demonstrates enhanced detection and classification performance in comparison to other standalone DL methods.

7.2 Proposed Methodology

The framework of the proposed FDD method comprises five distinct phases: data acquisition, data pre-processing, model training, model validation, and model evaluation. Collectively, these phases constitute the steps required to develop the target model, as illustrated in Figure 7.1.

Phase I: HIL-Based dataset Acquisition

Supervised learning necessitates labeled training data to develop FDD models that are capable of classifying system states into known fault types and categories. In order to simulate the system's behavior during faults, the system model and the controller have been augmented with additional modules. The proposed methodology employs the real-time FI method to generate a representative dataset, thereby overcoming this drawback. In this manner, the majority of potential faults in sensor and communication signals can be addressed without necessitating any modifications to the original system model. The real-time FI framework and the HIL system represent the principal constituents of the data acquisition phase. The real-time FI framework developed in Chapter 6 is employed to

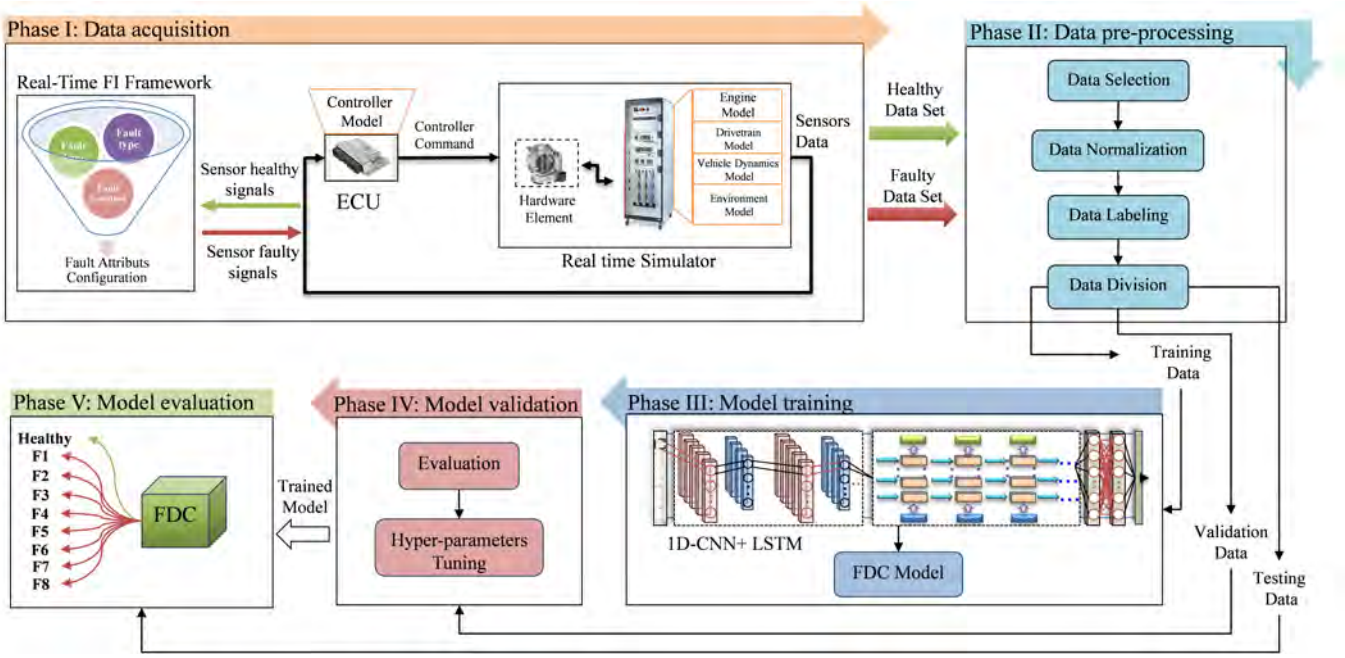


Figure 7.1: Hybrid DL-based FDD methodology.

generate a representative set of faults under varying conditions.

Phase II: Data Pre-Processing

It is of significant importance to recognise that the inclusion of irrelevant features in the training data may result in a detrimental effect upon the accuracy with which the data is classified [280]. This is why data preprocessing is an essential step in the development of DL models. Moreover, the inclusion of a significant number of superfluous features increases the cost of DL implementation due to the high computational overhead. Consequently, preprocessing the datasets in advance serves to circumvent the overfitting issue and reduce the computational cost. Another significant challenge in developing effective intelligent FDD is the imbalanced dataset, which is particularly prevalent in real-world applications. The ratio between the datasets comprising healthy and faulty information is typically not equal, which has a negative impact on the performance of the model. In light of the aforementioned issues, various solutions have been put forth with the aim of addressing these concerns. These include augmentation approaches, feature learning methodologies, and classifier design strategies [180]. To ensure an adequate comparison between faulty and healthy data classes, permanent faults were introduced. In the meantime, to address the issue of imbalanced

data distribution among faulty classes, a feature-learning-based CNN with a large dataset was employed to effectively learn the features of the imbalanced faulty classes. In this manner, the risk of overfitting when the distribution of faulty data differs can be mitigated by leveraging the feature extraction and learning capabilities of the CNN. In particular, when dealing with transient FI, it is necessary to modify the neural network in order to handle the small and unbalanced fault dataset, for instance by using regularized CNN.

The proposed method comprises four stages: selection, normalization, labeling, and partitioning. In order to ensure the inclusion of useful system variables in the training data on which the classification task is performed, it is necessary to select the relevant features at the outset of the process. In this study, five distinct variables at the system level were selected as candidates for the classification task. The variables selected for this study were those of engine speed, engine torque, vehicle speed, rail pressure, and intake manifold pressure. The subsequent stage in the process is the cleansing and normalization of the data. It is important to note that due to the inherent uncertainty of real-world data, it is necessary to perform data cleansing and filtering for outlier detection and missing values prior to feeding the data into learning algorithms. Nevertheless, in the context of the HIL platform, which is the subject of this study, the aforementioned procedure is not obligatory. Conversely, the selected data is transformed into a binary matrix in order to enhance the performance of the training network. To achieve this, the Z-score normalization function, as described in [281], is employed to rescale the amplitude of the variables, ensuring that the data values are uniformly distributed within the range [0–1]. In order to facilitate the supervised learning process, once the data have been normalized, the samples of the time series are then labelled with the corresponding classes. This is accomplished by assigning a label to each dataset belonging to a specific category, which may be related to the healthy class or a specific class of the fault. In order to encompass the majority of fault types, nine distinct classes have been labelled. Finally, the data is divided into three subsets for subsequent use in distinct phases: training, validation, and testing. In particular, 60% of the total dataset is allocated to the training process, while the remaining dataset is utilized for validation and testing. Twenty percent of the dataset is allocated to each of the aforementioned phases. The proposed method employs deep neural networks to automatically extract features from training samples, overcoming the limitations of traditional feature extraction techniques that rely on domain knowledge and human experience. In particular, DL enables the

effective capture of the deep features of faulty and healthy input data through the utilization of the feature extraction capability of CNNs [282].

Phase III: Model Training

A key aspect of the proposed methodology is its deep neural network architecture. In addition, the selection of appropriate hyperparameters is a crucial step in the construction of an accurately optimized model. The literature on FDD based on DL demonstrates a plethora of approaches corresponding to different domains. Among the proposed DL models, CNNs and LSTM networks have demonstrated significant success in the construction of intelligent diagnostic models. As the various types of malfunction exert unique effects on the system data, a multitude of vital characteristics are gleaned from the data at different points in time. Consequently, the CNN algorithm represents an optimal choice for developing FDD models due to its capacity to identify significant features of the data without the need for human supervision. Nevertheless, the investigation of alternative algorithms has been prompted by the inherent limitations of the CNN in processing data that consists of multivariate time series data. In contrast, LSTM and GRU were developed to address the shortcomings of RNN, i.e., the vanishing and exploding gradients. Moreover, in the context of sequential time series data, it has been demonstrated that LSTM and GRU techniques outperform other approaches in terms of their accuracy. As shown in [283], [284], LSTM achieves better accuracy on larger datasets, despite the advantages of GRU reflected in its lower resource requirements and faster inference time compared to LSTM [284]. Moreover, LSTMs have shown remarkable performance in classification tasks in [285, 286, 287]. Therefore, the structure of recurrent units can be effectively selected depending on the dataset and the corresponding application [288]. In this study, LSTM is employed to address the dynamic multi-mode behavior exhibited by large datasets, thereby facilitating the accurate classification of faults. However, for complex multi-class classification tasks, a significant increase in overhead is required in terms of cell structure and training time. In order to overcome the limitations of both techniques, i.e., CNNs and LSTMs, and to take advantage of both classifiers, two structures to develop the target model have been integrated, as illustrated in Figure 7.2.

The primary components of the proposed network architecture are as follows: input layer, 1D-CNN layers, LSTM layers, fully connected layer, and output layer. In a process referred to as

training, datasets, represented in time series format, are fed into the 1D-CNN network through the input layer. In particular, following the reception of the row data from the input layer, the convolution layer extracts local features and stores them in the form of a feature map. Feature maps comprise a set of nodes associated with a specific input region. Similarly, several non-linear activation functions are employed to extract features from the input layer. Among these, the rectified linear unit (ReLU), Tanh and sigmoid function are the most commonly used. The results of the research on the development of intelligent FDD of machines indicated that the ReLU activation function achieved the highest test accuracy compared to the Sigmoid and Tanh functions [289]. Moreover, during the training process, a notable enhancement in the performance of DL network is observed when the ReLU is employed in comparison to the Sigmoid function [290]. Consequently, following the assessment of the aforementioned activation functions, ReLU was selected for the proposed architecture in order to generate a non-linear expression and to enhance the discrimination of the learned features.

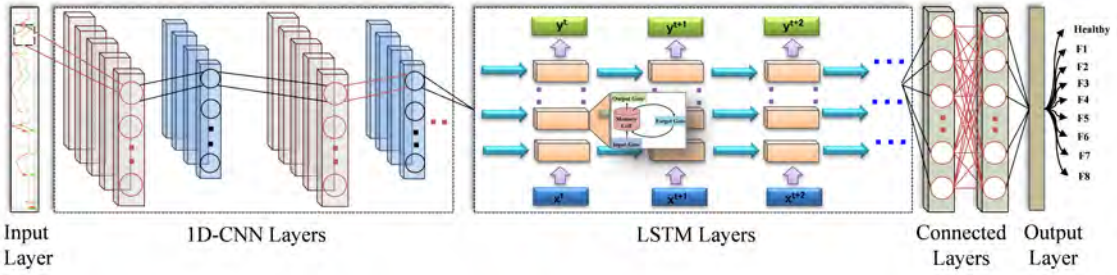


Figure 7.2: Deep neural network architecture of the proposed model.

It can be observed that the reduction of the feature map subsequent to the convolutional layer is indicative of the pivotal role played by the pooling layer. Max pooling, average pooling, and L2 norm pooling are the most common methods employed in this layer. The proposed study employs a multilayer LSTM network positioned subsequent to the CNN layers, rather than connecting the final pooling layer of the CNN to the subsequent fully connected neural network layer. In this manner, the features are subjected to additional processing, thereby enabling the calculation of the probability score for each class by the fully connected layer. Subsequently, the features of the final LSTM layer are processed by the fully connected layer to generate the output probability score for classification. Finally, at the output layer, the class with the maximum likelihood is considered the output, thereby indicating the nature of the detected fault. Table 7.1 presents the specification

parameters of the proposed network architecture.

Table 7.1: The architecture parameters of the proposed model.

Layer	**Size and Parameters**
CNN Layer 1	Filters: 8, Kernel Size: 2, Activation: ReLU, Input Shape: [30 × 5], Output Shape: [30 × 8]
CNN Layer 2	Filters: 8, Kernel Size: 2, Activation: ReLU, Output Shape: [30 × 8]
CNN Layer 3	Filters: 8, Kernel Size: 2, Activation: ReLU, Output Shape: [30 × 8]
CNN Layer 4	Filters: 8, Kernel Size: 2, Activation: ReLU, Output Shape: [30 × 8]
CNN Layer 5	Filters: 8, Kernel Size: 2, Activation: ReLU, Output Shape: [30 × 8]
Batch Normalization Layer 1	Output Shape: [30 × 8]
Max Pooling Layer	Pool Size: 2, Output Shape: [15 × 8]
LSTM Layer 1	Neurons: 64, Activation: ReLU, Output Shape: [15 × 64]
LSTM Layer 2	Neurons: 64, Activation: ReLU, Output Shape: [15 × 64]
LSTM Layer 3	Neurons: 64, Activation: ReLU, Output Shape: [15 × 64]
LSTM Layer 4	Neurons: 64, Activation: ReLU, Output Shape: 64
Batch Normalization Layer 2	Output Shape: 64
Flatten Layer	Output Shape: 64
Output Layer	Output Shape: 9

Phase IV: Model Validation

The accuracy of the trained model from Phase III is validated using the validation data pre-processed in Phase II. The validation results are used to adjust the hyperparameter specifications of the network structure in order to optimize the FDD model. In particular, the value of the loss function is reduced, while the accuracy is increased. The main hyperparameters considered during the validation phase were the learning rate, number of epochs, batch size, number of batch normalizations, CNN, LSTM, dense, and drop layers.

Phase V: Model Testing

In this phase, the final model with the optimal hyperparameters is subjected to a test after the developed model has undergone optimization, i.e., the hyperparameters have been tuned based on the validation results. The performance of the optimized FDD model is evaluated on the basis of the

unseen data from Phase II. A variety of metrics, including precision, recall, F1-score, and accuracy, can be employed for this purpose [291]. The objective of the model is to detect and identify faults in test drive recordings as a classification problem. Specifically, the objective is to correctly identify and classify, along with healthy data, eight distinct types of sensor and communication faults. This study presents the performance of a hybrid 1D-CNN and LSTM in comparison with a single CNN and a single LSTM, respectively.

7.3 Case Study and Experimental Implementation

The validation process of the approach proposed in this chapter, including the case study and the experimental setup, has been performed using the virtual test framework developed in Chapter 10.

7.3.1 Case Study

To exemplify the applicability of the proposed methodology and to ascertain the efficacy of the proposed model, the ASM gasoline engine system integrated with the dynamic vehicle model provided by dSPACE [166] is employed as a case study. Furthermore, the HIL system is employed to generate healthy and faulty training, validation, and test datasets in real-time during operation. The architecture of the experimental platform utilized in this study is depicted in Figure 7.3. As illustrated, in order to achieve high fidelity in the simulation and to capture comprehensive system behavior characteristics, case study environmental system models are included. In particular, the powertrain, vehicle dynamics, and environmental systems are modelled and simulated along with the engine system. Furthermore, the virtual ECU is employed to simulate the control logic in the absence of a real ECU. The aforementioned systems are constructed and analyzed in the MATLAB/Simulink platform. Following the generation of code from models, namely the plant and control models, the code is deployed and executed on the corresponding hardware, i.e., the MicroAutoBox II and dSPACE SCALEXIO, respectively.

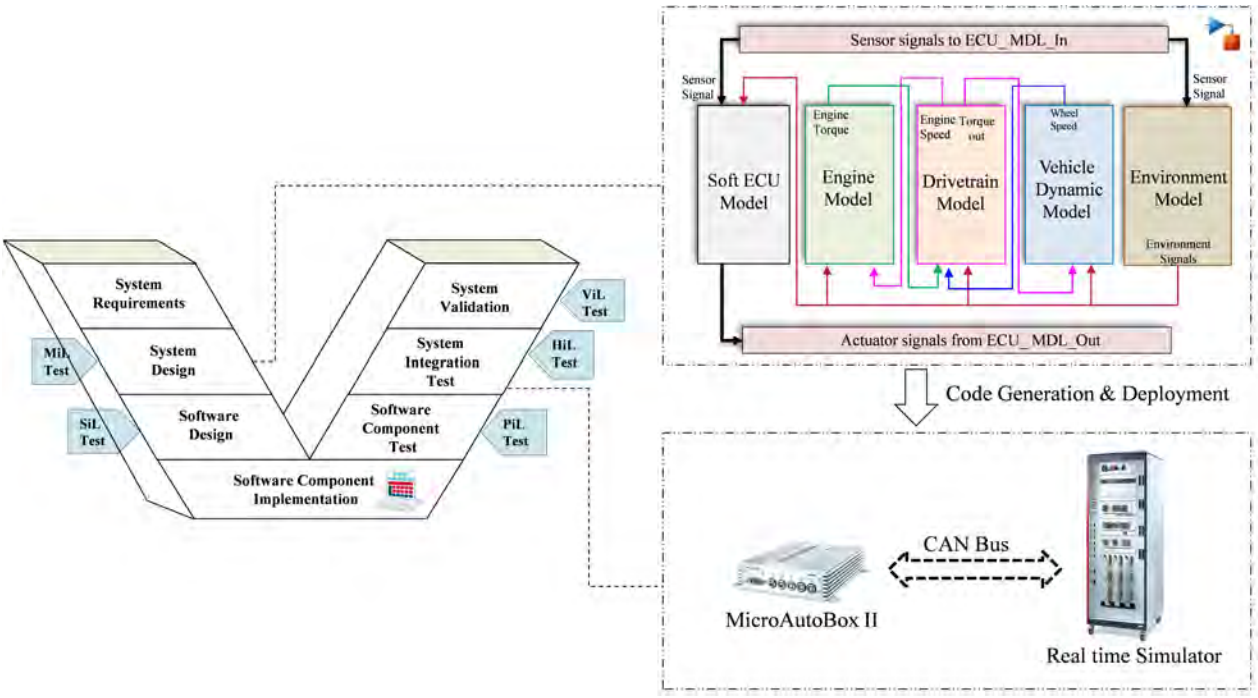


Figure 7.3: System architecture of the case study with different test phases of V-model.

7.3.2 Experimental Setup

The software tools provided by dSPACE facilitate the configuration and parameterization of the selected system and its components within the user environment, despite the complexity of the system in the case study. For instance, ControlDesk enables the user to select, modify, and acquire target sensors and communication signals in real-time. This enables the user to configure the experiment, calibrate the controller, read out the measurements, and acquire the data at the control station.

The HIL experiment configurations in this study are organized into three distinct levels. These comprise the system configuration layer, the drive cycle configuration layer and the fault injector configuration layer. At the system configuration level, the system parameters, including the internal and external system specifications, are defined. The principal specifications of this case study are presented in Chapter 6. A representative fault dataset reflecting the system behavior in the presence of faults is collected by defining the fault injector attributes, i.e., the fault type, fault time, and fault location. Table 7.2 provides a summary of the primary characteristics of the fault injector utilized in the generation of the dataset, along with the number of samples collected in each class.

The fault injector configurations specify the injection of different fault types into two locations:

the accelerator pedal position (APP) sensor and the engine speed sensor. The sensors into which the faults were injected were selected because of their significant impact on vehicle behavior. A NASA study found that the majority of accidents involving Toyota vehicles are caused by unintended acceleration [292]. Conversely, the engine speed sensor exerts a significant influence on all actuator control signals. For this reason, it is crucial to investigate the behavior of the system in the presence of a fault in the selected sensors.

Table 7.2: Configuration of fault injection with collected data samples.

Fault Type	**APP Value**	**RPM Value**	**Fault Duration**	**Data Samples**
Gain	10	5	0–300	59,945
Offset	5	300	0–300	63,910
Noise	1–100	0–8191	123–288	46,115
Stuck-at	0	0	0–300	59,830
Drift	0.1	10	17–86, 44–100, 122–300	60,195
Hard-Over	127	10,000	122–135, 184–225	34,800
Spike	1–100	700–8191	0–96, 103–293	57,805
Delay	10	3	0–96, 121–286	55,905

In order to configure the driving scenario, the user should select and configure the driving cycle from a list provided by dSPACE in ControlDesk. This list includes a variety of scenarios, such as city and highway scenarios. The selected driving scenario may be executed in either automatic or manual mode, contingent upon the user's requirements.

7.3.3 Dataset Description

In conducting these experiments, we record system-level behavior under normal and fault conditions in the form of temporal data. The data is employed to train and validate the FDD model. The system-level variables that are considered for the purposes of performing the detection and classification tasks are engine torque,rail pressure, vehicle speed, engine speed, and intake manifold pressure. The data is stored in an "IDV" file format. Subsequently, the ControlDesk tool is employed to transform the data into a comma-separated values (CSV) format.

Once the datasets have been captured, they are subjected to pre-processing based on the proposed methodology for data cleaning, labeling, normalization, and splitting. However, the resulting

CSV files are populated with irrelevant data, which may negatively impact the precision of the target model. Consequently, it is imperative to eliminate this type of supplementary data. During the labeling process, the fault types and corresponding locations are assigned to the data. The labels are incorporated into the system variables previously mentioned in the form of numerical values. To enhance performance, the HIL data is normalized with the Keras normalize function [293] subsequent to the labeling process. This process transforms the dataset into a binary matrix. Finally, the normalized data is partitioned into three discrete subsets, with one designated for training, another for early validation and model tuning, and the third for testing and evaluating the trained model. The data is then randomly divided into three portions: 60% is designated for training, 20% for validation, and 20% for testing. Following the cleansing procedures, the total number of data points collected was 503,850. In particular, the training phase employed 302,310 samples with 33,590 instances per class. The validation phase made use of 101,670 samples, while the testing phase utilized the same number of samples as a set of unseen data. It is crucial to acknowledge that the absence of real automotive datasets for the purpose of evaluating the efficacy of the developed model poses a challenge to the external validity of the model. To mitigate this potential limitation, the model's performance was evaluated using unseen samples, which were employed as test data in the experiments.

7.3.4 Hyperparameters Optimization

The subsequent stage is to develop the FDD model through training and validation following the data preprocessing. To reduce the learning time of the training process, the model is implemented in Google Colab using Python with the Tensorflow [294] and Keras [295] libraries. Figure 7.4 illustrates the flowchart of the model implementation and optimization process, which commences with the collection of data and culminates in the testing of the optimally trained model.

A trained model is validated using a separate validation dataset in order to identify the model exhibiting the greatest accuracy relative to the training data. This process serves the purpose of circumventing the problem of overfitting, which arises when the model becomes overly reliant on the training data. In the process of optimization, the values of hyperparameters are adjusted in order to identify the optimal structure of the FDD model, which exhibits the highest possible performance. A set of parameters, including epoch, batch size, batchnormalizing layer, learning

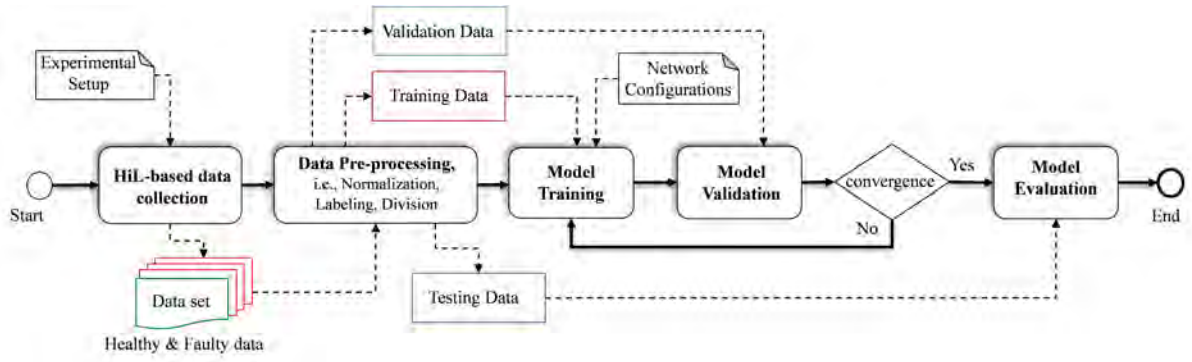


Figure 7.4: Flowchart of model implementation and optimization.

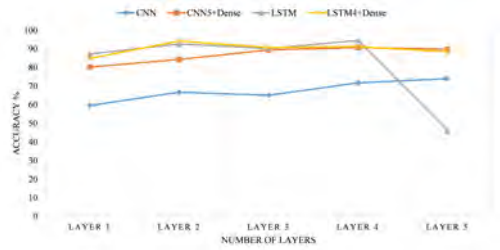
rate, CNN layer, LSTM layer, dense layer, and drop layer, were selected and adjusted based on the accuracy of each training. Table 7.3 provides a summary of the selected hyperparameters, which are to be optimized. In order to achieve the aforementioned objective, two evaluation metrics, i.e., accuracy and loss, are required.

Table 7.3: Range of hyperparameters used in optimization process.

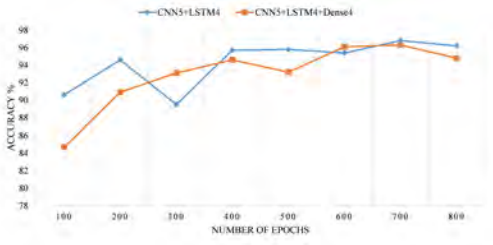
Hyperparameter	**Range**
CNN Layers	0–5
LSTM Layers	0–5
Dense Layers	0–5
Epochs	50–900
Max Pooling Layer	0–1
Drop Layer	0–2
Batch Normalization Layer	0–2
Batch Size	64–150
Learning Rate	0.001–0.0001

These performance metrics have been implemented in the Python programming language using the open-source scikit-learn library [296]. The outcome of the optimization procedure in terms of accuracy are presented in Figure 7.5.

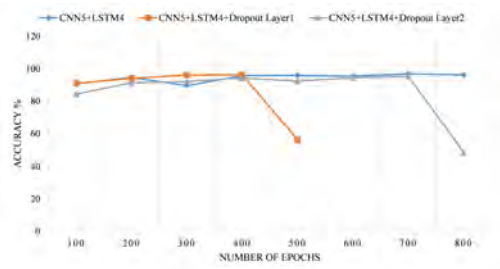
The results presented in Figure 7.5a indicate that a five-layer CNN with a dense layer achieved the highest validation accuracy of 90%. Conversely, the addition of further dense layers to the LSTM did not result in an improvement in validation accuracy. Accordingly, the proposed model structure comprises a five-layer CNN, a four-layer LSTM network, and four dense layers. Similarly,



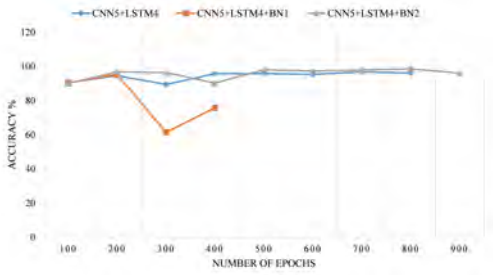
(a)



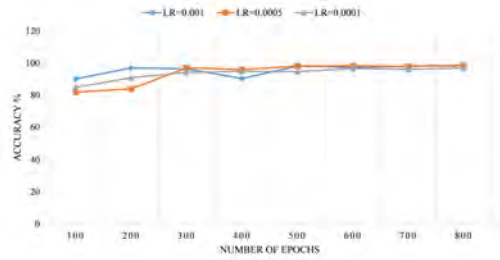
(b)



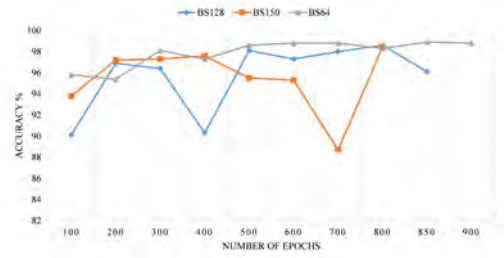
(c)



(d)



(e)



(f)

Figure 7.5: Hyperparameters optimization results. (**a**) Validation accuracy with different number of layers. (**b**) Validation accuracy with different number of epochs. (**c**) Validation accuracy with different number of dropout layers. (**d**) Validation accuracy with different number of batch normalization layers. (**e**) Validation accuracy with different number of learning rate. (**f**) Validation accuracy with different number of batch size .

as illustrated in Figure 7.5b, the model training with different epochs demonstrates that the optimal results, or 96.8%, can be achieved with 700 epochs. It is also noteworthy that the performance of the model without dropout layers is superior to that of the model with additional layers, as illustrated in Figure 7.5c. In subsequent experiments, the dropout layers is omitted. Two batch normalizing layers were added to the proposed structure due to the favorable validation results of the model, which achieved a validation accuracy of 98.6% (Figure 7.5d). With regard to the learning rate, Figure 7.5e illustrates the impact of reducing the learning rate on the validation accuracy. The optimal learning rate of 0.0005 was achieved at a validation accuracy of 98.3%. Finally, the model is trained with a varying number of batch sizes after the core hyperparameters have been determined. Figure 7.5f illustrates that the results obtained with a batch size of 64 are superior to those obtained with the other batch sizes with respect to the validation criteria. Once the validation phase has been completed, the optimal hyperparameters for the proposed model are presented in Table 7.4.

Table 7.4: Optimal Hyperparameters for CNN+LSTM Model.

Hyperparameter	**Value**
CNN Layers	5
LSTM Layers	4
Dense Layers	0
Epochs	850
Max Pooling Layer	1
Drop Layer	0
Batch Normalization Layer	2
Batch Size	64
Learning Rate	0.0005

7.4 Results and Discussion

In this chapter, the results of the optimization of the FDD model, based on test data, are presented, along with a discussion of the capabilities of the proposed methodology for the detection and classification of different fault types in sensors and communication systems. As indicated in the method-

ology section, the model is evaluated based on a number of evaluation metrics, including accuracy, precision, recall, and F1-score. These metrics are drawn from the evaluation framework outlined in [291]. In addition, the results of the proposed model are contrasted with those of standalone ML and DL techniques. In particular, the efficacy of the proposed model has been contrasted with that of two distinct single classifiers, i.e., LSTM and CNN.

7.4.1 Evaluation Metrics

This study focuses on the accuracy, precision, recall, and F1-score, among the various available evaluation metrics. Classification accuracy represents the ratio of the number of correctly identified instances to the total number of input samples. This ratio can be expressed as shown by the Equation (7.1).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.1}$$

The symbols TP and FP are used to indicate the number of cases correctly identified as positive, i.e., predicted faults correctly classified as such, and incorrectly identified as positive, respectively. The symbols TN and FN are used to indicate the number of cases correctly classified as negative, i.e., truly classified negative cases, and incorrectly classified as negative, respectively.

The precision, which is referred to as the positive predictive value (PPV), indicates the proportion of the total classification results that are predicted to be correct. Mathematically speaking, precision can be quantified by dividing the number of accurate predictions by the sum of correct and inaccurate predictions. This can be demonstrated by the Equation (7.2).

$$Precision = \frac{TP}{TP + FP} \tag{7.2}$$

Another metric utilized in evaluating the recall, or true positive rate (TPR), is to ascertain its sensitivity. This is represented by the percentage of correctly identified items compared to all items requiring identification. As previously indicated, the recall can be calculated by dividing the number of true positives by the sum of true positives and false negatives (Equation (7.3)).

$$Recall = \frac{TP}{TP + FN} \tag{7.3}$$

The final measure of performance in the field of ML is the F1-score. This score represents the harmonic mean between precision and recall. Consequently, it permits the assessment of the

Table 7.5: Fault detection accuracies of the models.

Model	Precision	Recall	F1-Score
CNN	92.69%	91.99%	92.32%
LSTM	95.98%	94.87%	95.36%
CNN-LSTM	98.86%	98.90%	98.88%

model in terms of precision and robustness. The mathematical calculation of the F1-score can be performed using the following Equation:

$$\text{F1-Score} = 2 * \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{7.4}$$

7.4.2 Testing Results

In the previous phase, the hyperparameters were tuned to yield the optimal FDD model. This was achieved based on the results obtained from the aforementioned optimization. Table 7.5 presents the accuracy metrics for the proposed model, as evaluated against both individual LSTM and CNN models, for comparative analysis. The outcomes indicate that the proposed model outperforms the individual models with respect to accuracy, recall and F1-score.

The performance of the proposed model in classifying the detected faults is evaluated in terms of accuracy, recall, and F1-score for each of the eight fault classes and healthy data. The results are presented in Figures 7.6–7.8. In general, all networks exhibited satisfactory classification accuracies on unseen test data, with an average accuracy of 91.7% for the CNN, 95.6% for the LSTM, and 98.8% for the CNN-LSTM. It is notable that the proposed model, which is a hybrid architecture, outperforms the standalone models, despite the optimization process being applied to the three DL models.

In particular, the CNN model demonstrated a lower accuracy in addressing drift, delay, and offset faults relative to the LSTMs and the hybrid CNN-LSTMs. The accuracy was 86.64%, 87.94%, and 88.98%, respectively. Additionally, the false alarm rate increases as the number of false positives and false negatives rises in proportion to the number of healthy classes in the CNN model. Nevertheless, the outcomes of the LSTM indicate a performance enhancement, with the LSTM model exhibiting superior performance in comparison to the CNN model with respect to accuracy.

The model exhibited the highest accuracy in the stuck-at, noise, gain, and delay fault categories, with accuracies of 99.85%, 98.21%, 97.7%, and 97.85%, respectively. Nevertheless, our proposed model, with an average accuracy of 98.8%, overcame the limitations of the remaining fault classes. Furthermore, it is notable that the accuracy for three classes, i.e., gain, noise, and stuck-at, is 100%, while for the remaining classes, it is above 98%, with the exception of the delay class, which is 96.36%. This limitation arises due to the fact that certain features of delay patterns exhibit similarities to other classes, which may result in their misinterpretation as delay patterns.

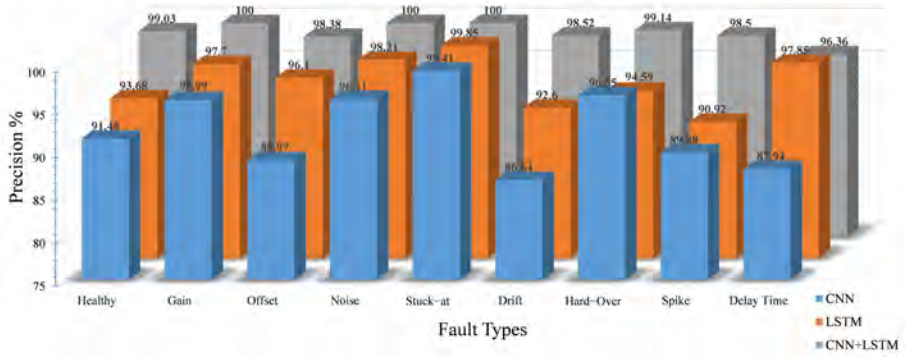


Figure 7.6: Classification performance in terms of precision.

A comprehensive examination of the recall data for the models under consideration indicates that the performance of the individual CNNs remains below expectations, with an average accuracy rate of 95.6%. This is due to the low recall of the delay, spike, and drift faults, which were 86.24%, 87.37%, and 88.51%, respectively. In contrast, in the LSTM model, recall of the seven classes is also above 90%. In comparison to the CNN, the number of false positives for the healthy classes is relatively lower. Furthermore, the false negative and positive rates were 5.92% and 6.32%, respectively. The probability of false positives for healthy data remains relatively high. Consequently, the higher number of false positives and false negatives in the healthy class leads to satisfactory precision and recall for fault classification, yet suboptimal precision and recall for fault detection. The results of the recall for our proposed model are encouraging, demonstrating good performance across all target classes. In terms of recall, the model demonstrated a success rate exceeding 98.38% in eight out of nine classes, with the exception of spike faults, where the success rate was 97.38%.

In the context of ML, the harmonic mean of recall and precision is represented by the F1-score. As illustrated in Figure 7.8, our model outperforms the other models in terms of F1-score for each

classification. The proposed model demonstrated an exceptionally high level of accuracy, with a minimum F1 value of 97.21%. This performance surpassed that of the other models, i.e., CNN and LSTM, which achieved minimum F1-score of 87.08% and 92.1%, respectively.

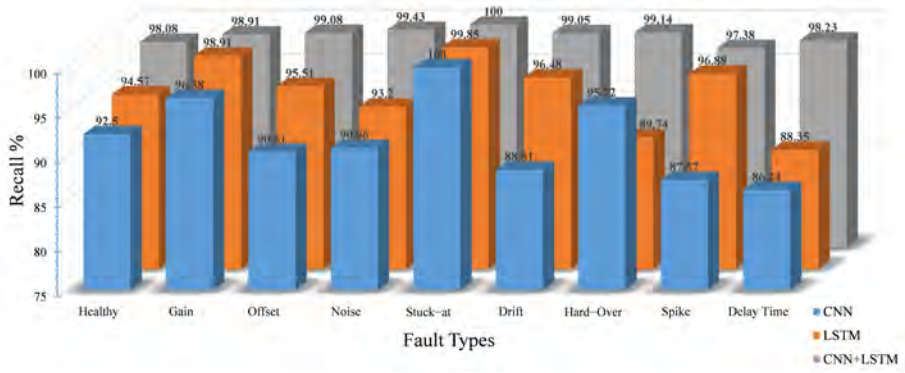


Figure 7.7: Classification performance in terms of recall.

The classification results, presented in the form of Receiver Operating Characteristic (ROC) curves, are shown in Figure 7.9 to illustrate the relationship between the True Positive Rate (TPR) and the False Positive Rate (FPR) of the proposed model. The ROC curve reaches the left corner of the graph in proportion to the classification performance. Consequently, the results indicate that the proposed model exhibits a low false positive rate.

In conclusion, the hybrid CNN and LSTM based FDD model with the proposed methodology demonstrates superior performance compared to other standalone DL techniques. The accuracy and F-score of our proposed model, which are 98.5% and 98.84%, respectively, are higher than those of other classification models with different fault classes. Thus, the hybrid technique significantly outperforms the majority of state-of-the-art models. Consequently, this could facilitate the development of more accurate FDD models in the future, contingent upon the integration of appropriate methodologies.

7.4.3 Computational Complexity Analysis

In addition to the importance of prediction accuracy, the time cost of training and inference should be considered in any application of an intelligent FDD model. As shown in Table 7.6, both training and inference times (in seconds) were investigated in this study. As mentioned in the training

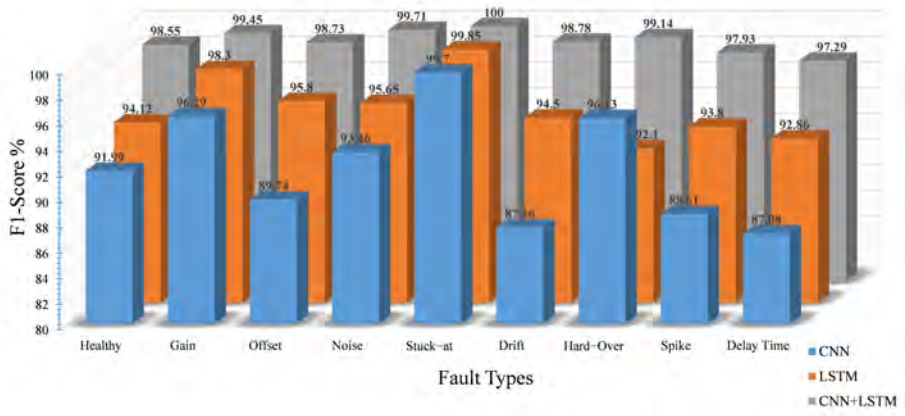


Figure 7.8: Classification performance in terms of F1-score.

section, the development environment for the model was a high-capacity cloud tool, i.e., Google Colab.

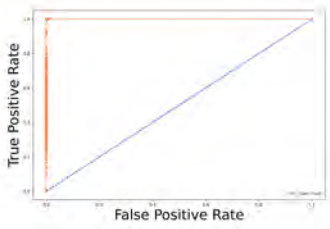
Table 7.6: Training and inference times of the models.

Model	**Training Time (s)**	**Inference Time (s)**
CNN	368.4	1.14
LSTM	4874.7	5.22
CNN-LSTM	9541.2	3.07

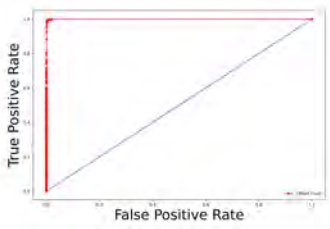
Despite its high classification precision, the hybrid approach requires a prolonged training period. This can be attributed to the more complex model architecture of the hybrid approach. However, the execution time for classification with test data is considerably shorter than that of the LSTM.

The proposed model's time requirement is comparable to that of traditional ML methods, which necessitate additional time for feature extraction. Moreover, the target model can be trained offline, but in order to run the model on a real-time system, it is necessary to ensure that real-time conditions are met. One potential avenue for reducing inference time is the use of dedicated software libraries or the selection of a specific network architecture, such as PCA-GRU. The comparison facilitates the pursuit of further enhancements to reduce the computational time in real-time applications. The aforementioned improvements are contingent upon a balance being struck between the objective of ensuring accuracy and the necessity of reducing the complexity of the trained model.

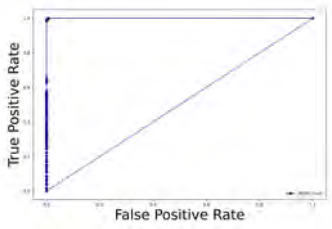
(a) Healthy



(b) Gain fault



(c) Offset fault

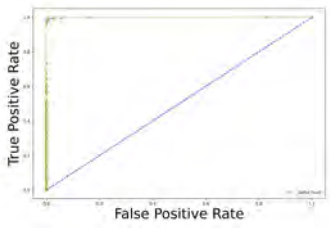


(d) Noise fault

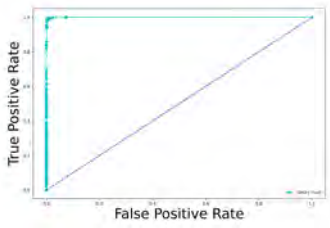
(e) Stuck-at fault

(f) Drift fault

(g) Hard-over fault



(h) Spike fault



(i) Delay fault

Figure 7.9: Classification performance in terms of AUC–ROC curve.

7.5 Conclusions

This chapter presents an original and efficient methodology for the early identification of faults in an ASSs during the developmental process, particularly during the system integration phase. The objective of this methodology is to detect and identify system-level fault types, as a classification problem, in sensors and communication signals during the real-time validation of ASSs. A hybrid CNN and LSTM architecture was chosen as the foundation upon which to construct a classification model with high resilience and accuracy. In this way, the deficiencies of each approach can be overcome, and the advantages can be leveraged. To address the challenge of limited representative training data, a novel real-time FI framework was developed along with the HIL simulation framework. In addition to complex test scenarios in real-time, the framework introduced eight distinct types of faults into the target devices and sensor signals. Consequently, the impact of the faults upon the system behavior was quantified in the form of time-series data. The data were employed for the training, validation, and testing of the target classifier. A novel approach was taken in which the original system model remained unaltered while eight distinct types of sensor and communication faults were introduced to the system in real-time. As a consequence, it is possible to represent the behavioral characteristics of the system with a high degree of accuracy and comprehensive understanding.

In order to illustrate the viability of the proposed methodology and to assess the efficacy of the proposed FDD model, a complex gasoline engine system was employed. In this study, a proportion of 60% of the collected dataset was utilized for the training phase, while 20% was employed for the validation phase and 20% for the test phase. The optimization process is conducted in a manner that permits the effective tuning of the hyperparameters in order to enhance the convergence rate of the proposed structure and to prevent the issue of overfitting. A variety of evaluation metrics, including recall, precision, F1-score have been employed to assess the efficacy of the proposed FDD model.

In contrast to standalone DL approaches, experimental results demonstrate that the proposed model outperforms its rivals in terms of classification accuracy for previously unseen test data, achieving an average accuracy of 98.85%. Specifically, the proposed model demonstrated superior precision and recall for eight different fault classes, as well as the healthy class. The average

precision and recall values for each class were 98.88% and 98.81%, respectively. Moreover, the proposed model demonstrated superior performance to individual DL methods, achieving a maximum F1-score of 98.84%. The results of our study indicate that the proposed model, which is a hybrid architecture, exhibits superior performance compared to individual models. Finally, the times required for training and inference for each model were analyzed with a view to identifying potential areas for improvement.

8 Detection and Clustering of Unknown Concurrent Faults

As previously stated in Section 3.5, RQ3 concerns the issue of identifying and categorizing unknown faults during the real-time validation process. This chapter contributes to answering RQ3 by proposing a novel method for FDD of unknown faults considering noisy conditions based on the representative data generated in Chapter 6. After an overview of the chapter in Section 8.1, the phases of the proposed method are described in Section 8.2. Subsequently, the structure of the target system and the model training are presented in Section 8.3. Based on the real-time dataset of two case studies with the developed framework (in Chapter 10), the evaluation results of the proposed model are presented in Section 8.4. Finally, the conclusion of the chapter is presented in Section 8.5.

Note: It is important to highlight that the proposed methodology and the corresponding implementation outcomes presented in this chapter have been published in [297].

8.1 Chapter Overview

Notable achievements have recently been made in the field of ASSs quality assurance through the use of real-time HIL simulations. The HIL platform enables the development of safe, flexible, and reliable realistic simulations during the system development process. Nevertheless, the generation of a substantial quantity of data during the execution of HIL tests represents a challenge, particularly in the context of test automation. The implementation of expert knowledge-based approaches to analyze generated recordings, with the objective of detecting and identifying faults, is costly in terms of time, effort, and difficulty. Consequently, this chapter presents a novel ML-based approach for the efficient and automated FDD of automotive sensor signal faults, negating the need for human intervention. Specifically, a hybrid GRU-based DAE model with the K-means algorithm is

developed for the fault-detection and clustering problem in sequential data. Consequently, the real-time historical data enables the accurate detection and clustering of unknown individual faults and simultaneous faults under noisy conditions. The suitability and benefits of the proposed method for the HIL testing process are illustrated through two automotive case studies. It is important to note that in order to test the effectiveness of the proposed methodology, a high-fidelity gasoline engine and vehicle dynamic system, along with an entire vehicle model, have been employed. The superiority of the proposed architecture is illustrated through the presentation of reconstruction error under various noise levels, demonstrating its effectiveness in comparison to other autoencoder variants. The results of the validation process demonstrate that the proposed model exhibits high detection and clustering accuracy in the clustering of unknown faults. In comparison to standalone techniques, it is able to outperform in this regard.

8.2 Proposed Methodology

This section outlines the principal stages in the development of the proposed FDD, based on unsupervised DL techniques. The proposed methodology offers a solution to the detection and clustering problem in the presence of noisy and unbalanced time series data, which represents a significant advantage. This enables the simulation of real industrial scenarios where the faults are not known. The objective of the proposed model is to detect and cluster the sensor signal faults of the ASSs during the system integration testing phase. Consequently, the analysis of the resulting failures during the test drive can be optimized in an efficient manner, thereby reducing the time and effort required for this process. As illustrated in Figure 8.1, the development of the proposed model is conducted through four distinct phases, i.e., data acquisition, data preprocessing, feature extraction, and detection and clustering.

8.2.1 Data Collection

The utilization of an HIL simulation system allows for the precise capturing of the actual behavior of the target system in various types of faults. This platform is capable of accurately simulating the complete vehicle system. Moreover, the real-time interaction between the SUT, i.e., the ECU, and the controlled system can be accurately ensured. Multivariate time series data can be generated by

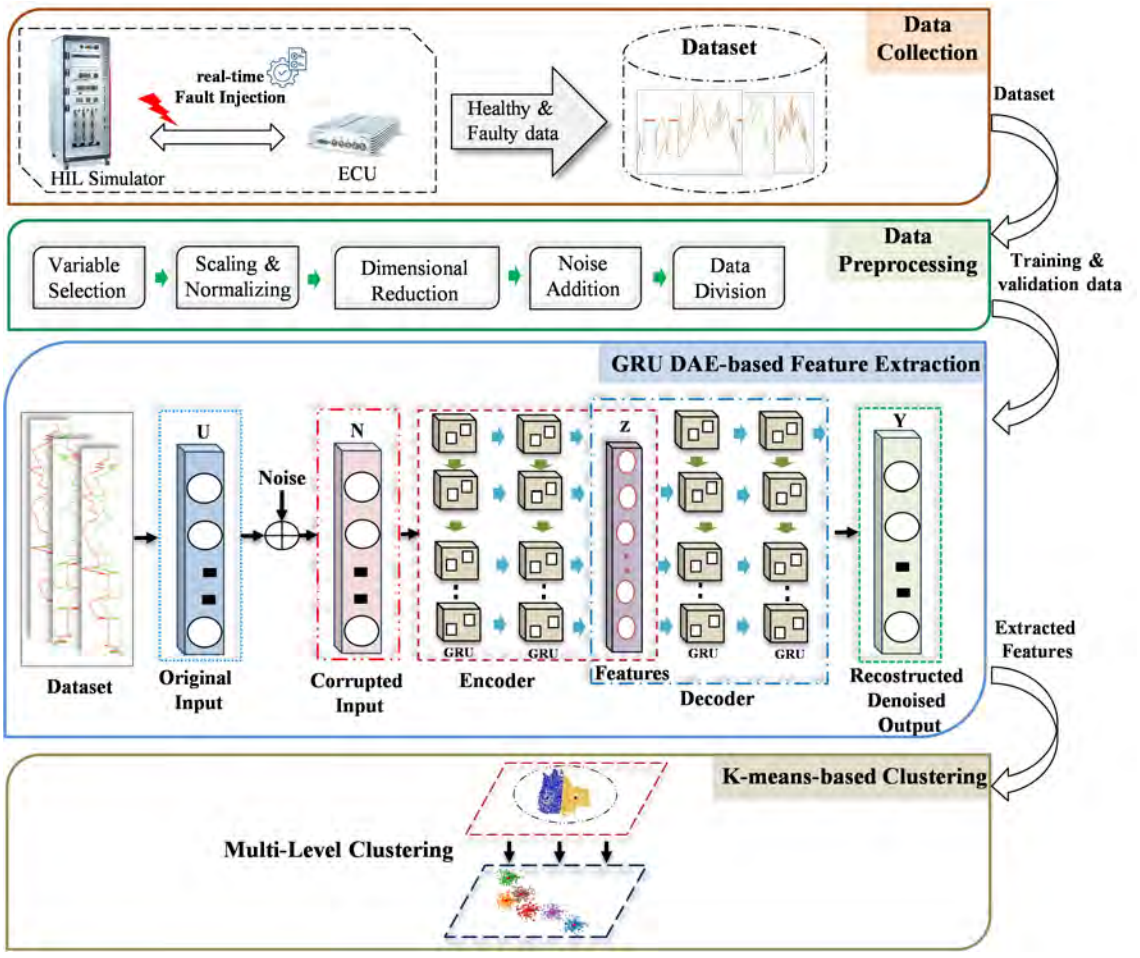


Figure 8.1: Proposed method for detection and clustering single and simultaneous faults.

recording the system performance during conducting the driving scenarios. To generate representative faults data, the real-time FI method developed in Chapter 6 is employed. Without modifying the original system models, a variety of single and simultaneous faults are injected into the CAN bus in real-time through a programmed process. Some examples of injected sensor faults include noise, gain, drift, delay, stuck-at, and packet loss. In order to collect concurrent fault data, two combinations of the aforementioned fault types are injected simultaneously at two different locations in real-time. Both the single fault data and the concurrent fault data are saved as comma-separated value (CSV) files and transmitted to the pre-processing stage.

8.2.2 Data Preprocessing

Although the data were generated in the context of a real-time simulation, additional pre-processing steps are necessary before the model is trained. This is due to the necessity of reducing irrelevant and superfluous features to prevent adverse effects on training, as previously documented in [280]. This approach avoids the problem of overfitting, while also reducing the computational cost of training. The main steps in this phase are variables selection, scaling and normalization, dimension reduction, noise addition, and data splitting. The high complexity of ASSs results in a considerable number of sensor signals exhibiting high dependencies. In this study, specific sensor signals of the target system were selected for inclusion in the model training, with the objective of capturing the most relevant features. At the system level, the input variables for detection and cluster formation include engine speed, throttle position, engine torque, engine temperature, intake manifold pressure, vehicle velocity, and rail pressure. Once the variables have been selected, the data values are normalized and scaled using the Z-score normalization function. This process is designed to enhance the efficacy of the training process. Consequently, the amplitude range of the input feature is uniformly rescaled to the range [0, 1]. A mathematical representation of the representative equation of the scaling step is provided in [281]. Principal Component Analysis (PCA) [298] is employed to reduce the dimensionality of the dataset. This process allows for the reduction of redundancy in the extracted features and the selection of representative features. Specifically, the principal variation factor is extracted from multiple variable vectors, and the correlated variables are transformed into independent variables. The Gaussian distribution [299], which incorporates different noise levels, is applied following the normalization of features and reduction in dimensions. In particular, random samples from a Gaussian distribution with a mean of zero and a standard deviation equal to a given value are added to the original data. Finally, the preprocessed data is divided into three distinct subsets. The initial portion of the data, comprising 80% of the total, is employed for training the target model. The subsequent 10% of the data, comprising the second and third portions, is utilized for validation and testing purposes.

8.2.3 GRU-DAE-based Feature Extraction

The majority of authentic automotive datasets are characterized by an inherent degree of uncertainty, as a consequence of environmental factors and the complexity of operational conditions. The current DL-based FDD models do not address this issue in experimental settings. The aforementioned issue has a negative impact on the accuracy of the developed model when utilized in real-world applications within the industry. Recently, the DAE technique, as described in [144], has demonstrated significant potential in enhancing the resilience of FDD models to noise, offering a promising solution to the challenge of noisy measurements. The model comprises four distinct layers, i.e., input, corrupted, hidden, and output. Figure 8.1 illustrates this structure. The hidden layers are comprised of an encoder and a decoder, as described in detail by Cho et al. in [300]. The encoder is responsible for transforming the high-dimensional, noisy input data (N) into a low-dimensional representation, commonly referred to as the latent code (Z), following the corruption of the original input data (U) with noise. Mathematically, the Equation 8.1 represents the encoder module of the GRU-based DAE.

$$Z = f_{enc}(WN + b) \tag{8.1}$$

where f_{enc} represents an activation function of the encoder, W is an encoder's weight, and b is an offset vector.

Conversely, the decoder's function is to map the latent code (Y) back into the original data space. This process enables the conversion of extracted features into the original input. Consequently, a compact representation of the input data can be generated, which can subsequently be utilized for the purposes of dimensionality reduction, data denoising, or generative modelling. The Equation 8.2 represents the decoder module of the proposed architecture.

$$Y = f_{dec}(W'Z + b') \tag{8.2}$$

The notation f_{dec} represents an activation function of the decoder, while W' and b' denote the decoder's weight and offset vector, respectively. The DAE structure enables the removal of noise and distortion from the input data through the encoding and decoding processes, resulting in denoised reconstructed output data. However, in order for the autoencoder to learn how to denoise the data, the input data should be provided with a certain level of noise, such as Gaussian noise or

corruption. In other words, two distinct types of input data, i.e., noisy and clean, should be provided to the structure. The fundamental principle underlying this technique is to train the underlying architecture by minimizing the reconstruction loss between the original input and the denoised reconstructed output. Mathematically, the loss function $L(U, Y)$ is computed according to Equation (8.3):

$$L(U,Y) = ||Y - U||^2 \tag{8.3}$$

In specific contexts, the design of DAE may vary based on the type of data being processed and the application domain in question [301]. Among the aforementioned architectures, GRU-based DAE has been demonstrated to exhibit superior performance in the detection of anomalies with multiple sensors relative to other DAE architectures, while simultaneously exhibiting reduced complexity and inference time [302]. In our proposed architecture, the GRU-based DAE is designed to enable the comprehensive extraction of representative healthy and faulty features in the presence of noisy data. Consequently, a series of GRU layers constitute both the encoder and decoder components.

The limitation of the RNN, i.e., vanishing and exploding gradients, can be overcome by leveraging the structure of the LSTM network. However, the intricate design of the LSTM necessitates a substantial computational burden. Consequently, GRUs have been developed to fulfill the necessities of classification tasks in real-time applications, particularly in terms of minimal resource consumption and rapid inference time [284]. The primary innovation of the GRU is its use of gating mechanisms to regulate information flow, which renders it computationally efficient and facilitates learning with fewer parameters compared to other types of RNN. In our study, the GRU was selected to construct the DAE model due to its aforementioned ability to capture long dependencies in the data with a less complex architecture, as cited in [303]. The structure comprises cells that store information and two gates for each block, i.e., the reset and update gates. The reset gate determines the extent to which the previous hidden state is to be discarded, whereas the update gate determines the extent to which new information is to be incorporated into the hidden state. The hidden state of the GRU is a composite of all previous timesteps and serves as input for the subsequent timestep. Figure 8.2 illustrates the GRU unit.

Mathematically, the output of the GRU cell can be described as follows:

$$z_t = \sigma(W_z x_t + V_z h_{t-1} + b_z) \tag{8.4}$$

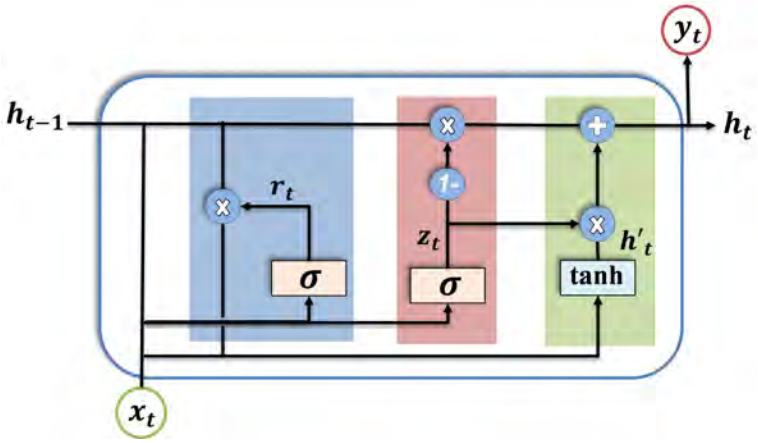


Figure 8.2: Internal structure of GRU cell.

$$r_t = \sigma(W_r x_t + V_r h_{t-1} + b_r) \tag{8.5}$$

$$h'_t = tanh(W_h x_t + V_h(r_t \times h_{t-1}) + b_h) \tag{8.6}$$

$$h_t = (1 - z_t) \times h_{t-1} + z_t \times h'_t \tag{8.7}$$

The notation z_t represents the GRU update gate, x_t represents the input vector, which may be considered a noisy input, h_t represents the output vector, which is the hidden state at time step t, W and V denote the weight matrices, b represents the bias matrices, and (σ, $tanh$) are the gate and output activation functions, respectively.

The comprehensive overview presented in Table 8.1 outlines the specific characteristics of the GRU-based DAE architecture employed. The proposed architecture employs a two-layer GRU encoder. The first layer comprises 128 units, while the second comprises 64 units. Furthermore, a batch normalization layer is integrated between the GRU layers. The encoder module employs a dense layer, which acts as a bottleneck, to perform the necessary transformations. In a comparable manner, the aforementioned layers are organized in a symmetrical configuration. The initial GRU layer comprises 64 units, while the subsequent layer contains 128 units.

As a result of training the GRU-based DAE with the backpropagation algorithm, it is possible to extract representative features of healthy and faulty behavior. The extracted features, i.e., the non-corrupted low-dimensional features, are subsequently employed in the clustering phase for the purposes of detection and clustering.

Table 8.1: The architecture parameters of the proposed model.

Layer	**Output Shape**	**Parameters**
$input_2$ (InputLayer)	[(None, 4)]	0
$reshape_3$ (Reshape)	(None, 1, 4)	0
gru_4 (GRU)	(None, 1, 128)	51,456
batch $normalization_2$ (BatchNormalization)	(None, 1, 128)	512
gru_5 (GRU)	(None, 64)	37,248
$dense_2$ (Dense)	(None, 4)	260
$reshape_4$ (Reshape)	(None, 1, 4)	0
gru_6 (GRU)	(None, 1, 64)	13,440
batch $normalization_3$ (Batch Normalization)	(None, 1, 64)	256
gru_7 (GRU)	(None, 1, 128)	74,496
$dense_3$ (Dense)	(None, 1, 4)	516
$reshape_5$ (Reshape)	(None, 4)	0
Total params: 178,184		
Trainable params: 177,800		
Non-trainable params: 384		

8.2.4 K-Means-Based Multi-Level Clustering

The denoised features are subsequently grouped into clusters on several levels using the K-means method [135], following the extraction of representative features by GRU-DAE in the preceding phase. The significance of this algorithm lies in its capacity to perform unsupervised learning, enabling the grouping of data samples. The algorithm does not require the availability of labeled data. Consequently, the hidden patterns can be extracted from the data without the necessity of knowing the true labels. K-means clustering has achieved considerable success in solving various technical challenges, including social tagging, shape recognition, and wireless sensor networks [304]. The fundamental tenet of the algorithm is the categorization of features into distinct groups on the basis of their similarity. Consequently, data points exhibiting analogous characteristics are allocated to the same cluster, with the cluster's mean value serving as the cluster's centroid. In other words, the distance between the data points and each cluster center is calculated. The objective of the training phase is to assign samples to the clusters that are most closely aligned with each center, with the aim of minimizing the sum of the squared distances within the clusters. The sum of squared

distances can be expressed mathematically as presented in [136].

The objective of the study in this chapter is to address the issue of detecting and clustering unknown faults. Consequently, the labels of the fault classes, which represent fundamental truths, are not accessible. In order to achieve optimal performance in the detection and clustering of faults, a multilevel feature clustering approach is employed. This involves three levels for individual faults and two levels for simultaneous faults. At the initial stage, the fault detection process entails the differentiation between healthy and faulty features. In the case of an individual fault, the faulty components can be determined by separating the corresponding features of the faulty component. Finally, the final step is carried out to group the features of a specific fault type, such as gain or delay, in the case of single and simultaneous faults. In order to identify the optimal cluster centroids, two distinct phases are undertaken. In the initial stage of the process, the euclidean distance between the data points and each centroid is calculated. The second phase entails calculating the mean of all data points assigned to a cluster and then updating each centroid. It is crucial to highlight that the extracted features lack any form of labeling. Consequently, the method should possess the capability to cluster samples of both individual faults and those of simultaneous faults, with different combinations. In the present study, five clusters were identified as representing single faults, while ten clusters were selected as representative of their respective combinations. In order to circumvent the limitations of the K-means algorithm when processing large amounts of data, the PCA technique is employed prior to clustering. The identification of the underlying structure of the data allows for the transformation of the data into a low-dimensional format, reducing the computational effort and complexity inherent in the analysis of high-dimensional data.

8.3 Case Study and Experimental Implementation

The following section outlines the experimental setup and implementation steps of the proposed method. Furthermore, the target system's structure, which serves as a case study to demonstrate the applicability of the proposed method, is described. In order to assess the efficacy and resilience of the proposed model, two distinct automotive case studies are employed. The first of these is a gasoline engine system, and the second is a dynamic vehicle system.

Is should be stated that the validation process of the approach proposed in this chapter, including

the case study and the experimental setup, has been performed using the virtual test framework developed in chapter 10.

8.3.1 Case Study 1: Gasoline Engine

In view of the pivotal function it performs in the vehicle system, the engine management system is categorized as an ASIL Class C to D system in accordance with ISO 26262. Consequently, any faults in such a system should be subjected to rigorous analysis and mitigation during the development process. This study employs a high-precision system model of the ASM petrol engine, which was developed by dSPACE [166]. The behavior model was constructed in the MATLAB/Simulink environment, encompassing all subsystems and components, in order to permit comprehensive modeling of the intricate characteristics of the system. As illustrated in Figure 8.3, the powertrain model, the vehicle dynamics model, and the environment model are also considered to represent the interaction characteristics of the target system with the other vehicle subsystems. In order to simulate the real ECU, the control algorithm of the target system is also modelled in the form of interconnected Simulink blocks. The ECU model of the engine system represents the SUT in this case study. The Real-Time Interface CAN Multimessage Blockset (RTICANMM) is employed to facilitate the connection between the SUT and the vehicle subsystems via the CAN bus protocol. The engine system is comprised of several partial systems, including the air path system, the fuel system, the piston engine system, and the exhaust system.

The model system offers an accessible method for identifying internal variables, which has led the selection of specific sensing and actuator components for potential fault locations in the context of this research. Such variables include crank angle sensing, battery voltage sensing, pedal sensor input, ignition and starting demand, mass flow of exhaust gas recirculation, engine revolutions per minute, pressures within the intake and exhaust manifold, fuel pressures, throttling control, temperature sensing of the cooling system, and rail bar sensing. A range of sensor faults is taken into account, including gain, stuck at, noise, delay, and packet loss. It is of paramount importance to emphasise that the aforementioned fault types are injected in real-time via RTICANMM without making any alterations to the system model.

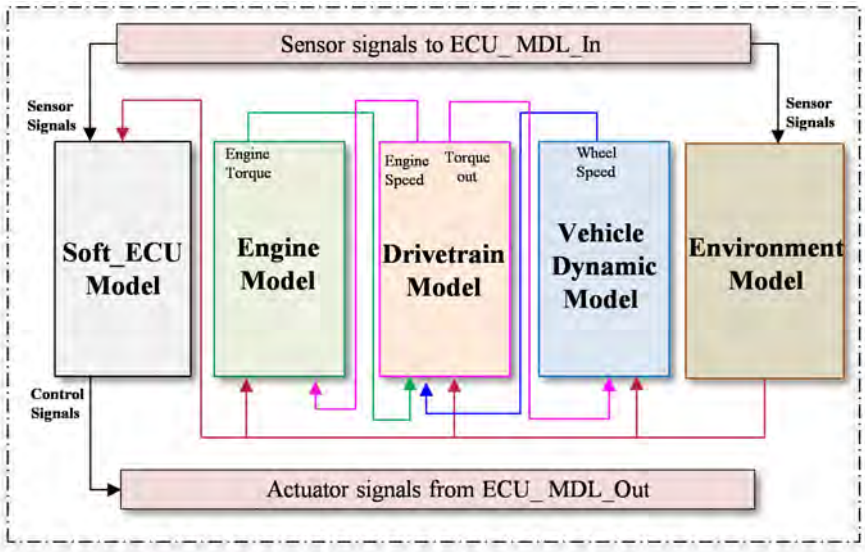


Figure 8.3: System architecture of the selected case study.

8.3.2 Case Study 2: Vehicle Dynamics with Traffic

The second case study is intended to verify the efficacy of the developed model in a real driving context. To avoid the limitations that a real test run entails in terms of the time required, the cost per test mile and the risk to the test subject, this study utilises a real-time simulation of a test run, i.e., a real-time virtual test drive framework using the ASM vehicle dynamics model from dSPACE. The framework comprises the HIL simulation, MicroAutoBox II as the actual ECU, CAN bus communication, driving elements, and a 3D model of the driving environment. The aforementioned framework is capable of supporting both manual and automated driving modes. The software development tools ModelDesk and MotionDesk facilitate the design and modelling of the driving environment and roads in precise and flexible manners. Consequently, the validation data for the proposed DL-based FDD model can be generated in real-time based on the user's behavior. The proposed system comprises the following key stages: (a) setting of internal and external system parameters; (b) design and selection of the driving environment (road and driving conditions), which is informed by the user's behavior; (c) configuration of the recording data system; and (d) execution in real-time. The test drive scenario is depicted in Figure 8.4 as a plot of vehicle speed versus time.

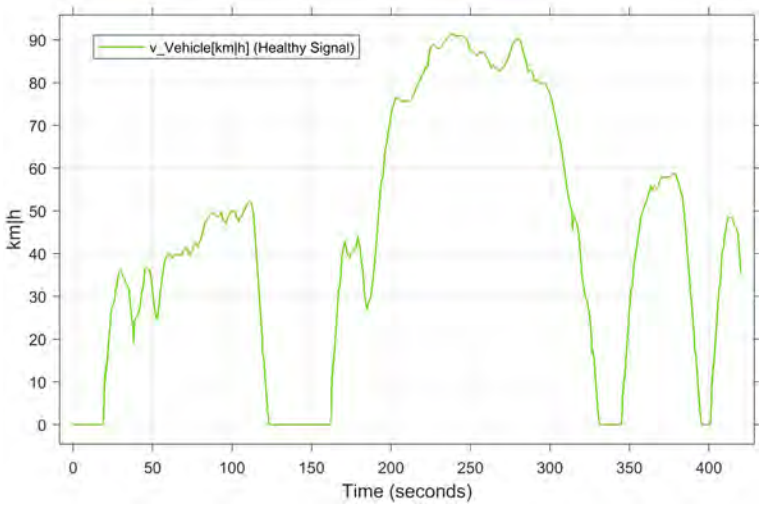


Figure 8.4: Driving scenario of the selected case study.

8.3.3 Data Description

Once the test configurations have been defined, the driving scenarios can be performed manually or automatically. In the absence of faults, the standard system behavior, including signals from sensors and actuators, can be recorded in real-time, thereby generating a healthy dataset. In Case Study 1, the highway scenario is selected from the list of driving scenarios to be performed automatically. In contrast, in case study 2, a real-time test environment that has been developed to perform a user-based test drive manually. The sampling rate of the recording system is 0.001 seconds. In both case studies, the input variables of the proposed model were selected as engine temperature [degC], throttle position [%], engine speed [rpm], average effective engine torque [Nm], rail pressure [bar], intake manifold pressure [Pa], and vehicle speed [Km/h]. The generation of the fault dataset involved the injection of the aforementioned five fault types into the target components, i.e., the APP sensor and the engine speed sensor. The real-time FI framework developed in Chapter 6 is utilized for this purpose. For further details on the potential fault types present in the time series data can be found in [164, 305].

The faults are classified as either permanent or transient in order to generate a representative dataset that encompasses a high level of coverage. For transients, an injection is conducted during the real-time execution of the program between 170 and 330 seconds. In contrast, for permanent faults, the injection is performed throughout the entire drive cycle. In conclusion, the generation of

faulty data results in the production of a total of 856,000 samples, including five single faults. In addition, concurrent fault samples are collected from ten different combinations of two fault types. The simultaneous injection of two distinct fault types into the APP sensor and speed sensor serves to accomplish this. The single and concurrent faults generated by the FI method are illustrated in Figure 8.5. Table 8.2 presents the data generated for each experiment corresponding to each of the fault types and their combinations. The dataset is generated in the form of CSV files, which contain all selected variables. The size of each CSV file is approximately 63,422 kB, which, when all files are combined, results in a 2.6 GB dataset for the DL approach. Subsequently, the collected data is transferred to the preprocessor, which performs data cleansing, normalization, and triplication. In Case Study 1, 80% of the generated time series data, comprising both healthy and faults data, is randomly selected for training the target model. The remaining 10% is used for validation and optimization of the decision threshold. Furthermore, 10% is utilized for the testing process. Consequently, no labeling process is applied to the dataset. The validation of the model is based on all data generated in Case 2, which comprises 180,000 patterns. In order to address the issue of robustness to noise, different levels of noise are added to the dataset according to the Gaussian mechanism.

Figure 8.5 illustrates some examples of the generated faulty data in comparison to the healthy data. Specifically, Figure 8.5a,b illustrate the system behavior in the presence of single fault, whereas the effect of the concurrent faults is presented in Figure 8.5c and d.

8.3.4 Training and Optimization of DAE

The initial step following the acquisition of real-time simulation data is the preparation of the data, which involves the cleansing and reprocessing of the data. Depending on the desired noise level, Gaussian noise is artificially added to the data after the data has been preprocessed. Subsequently, the data is divided into three distinct subsets, i.e., training, validation, and testing, as illustrated in Figure 8.6. Prior to the commencement of the training process, it is essential to establish the configuration of the model architecture. This entails the initialisation of the parameters and hyperparameters. In order to construct an efficacious model, a number of hyperparameters are selected in this study. These parameters include, but are not limited to, the number of layers, the number of GRU units, the learning rate, the batch size, the noise level, the activation function, and the

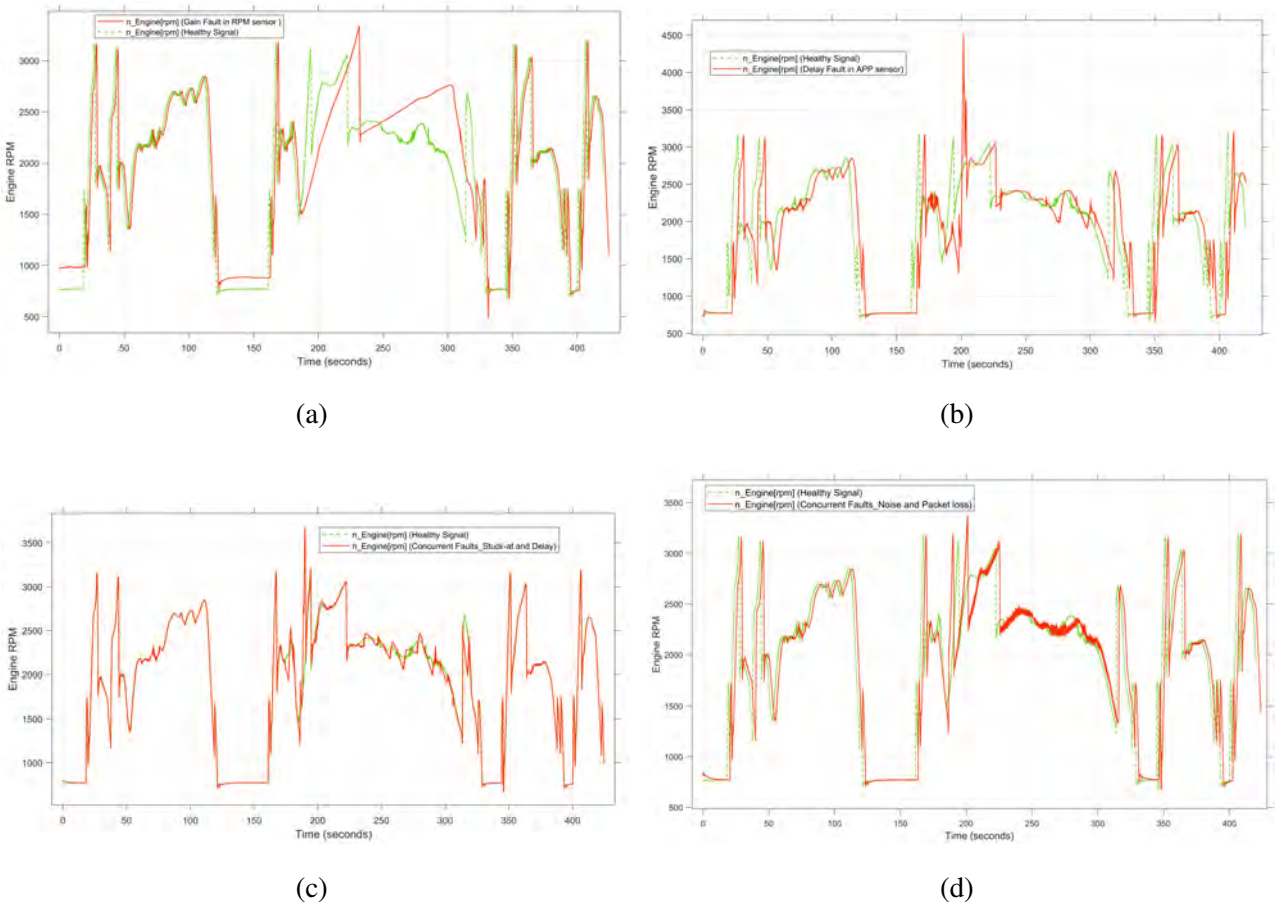


Figure 8.5: System behavior under single and concurrent transient faults. (**a**) Effect of gain fault as a single fault. (**b**) Effect of delay fault as a single fault. (**c**) Effect of stuck-at and delay fault as concurrent faults. (**d**) Effect of noise and packet loss fault as concurrent faults.

optimizer. In order to optimize the target model, the validation data samples are employed. This enables the values of the hyperparameters to be calibrated within the defined range. In order to select the optimal set of hyperparameters, the criterion is to achieve convergence with a high degree of accuracy. In light of this approach, a series of variants of the DAE model were trained, including ANN-based DAE, CNN-LSTM-based DAE and GRU-based DAE being the primary variants. Table 8.3 presents a summary of the optimized hyperparameters for the proposed GRU-based DAE architecture. To facilitate a more comprehensive comparison, the convergence curves of the proposed trained model resulting from the training process under varying levels of noise (3%, 6%, 8%, and 10%) are depicted in Figures 8.7a–d, respectively. One of the most significant challenges in the development of DL models is the tuning of hyperparameters during the optimization phase. Con-

Table 8.2: Faults and collected dataset description.

Fault ID	Fault Type	Fault Duration	Training Samples	Testing Samples
H	Healthy	-	2,352,000	294,000
F1	Gain	165–320 s	2,352,000	294,000
F2	Stuck-at	172–330 s	2,352,000	294,000
F3	Noise	175–310 s	2,352,000	294,000
F4	Packet loss	170–312 s	2,352,000	294,000
F5	Delay	170–325 s	2,352,000	294,000
F1F2	Gain and Stuck-at	166–338 s	2,352,000	294,000
F1F3	Gain and Noise	168–340 s	2,352,000	294,000
F1F4	Gain and Packet loss	179–329 s	2,352,000	294,000
F1F5	Gain and Delay	177–334 s	2,352,000	294,000
F2F3	Stuck-at and Noise	174–330 s	2,352,000	294,000
F2F4	Stuck-at and Packet loss	180–327 s	2,352,000	294,000
F2F5	Stuck-at and Delay	173–324 s	2,352,000	294,000
F3F4	Noise and Packet loss	166–320 s	2,352,000	294,000
F3F5	Noise and Delay	179–328 s	2,352,000	294,000
F4F5	Packet loss and Delay	176–336 s	2,352,000	294,000

sequently, to enable the dynamic adjustment of hyperparameters during the optimization process, the autotuning technique is employed in this study. Subsequently, following the initialisation of the model parameters, the training process is overseen by monitoring the loss and performance measures. Subsequently, the target parameter is updated in a corresponding manner until convergence is reached.

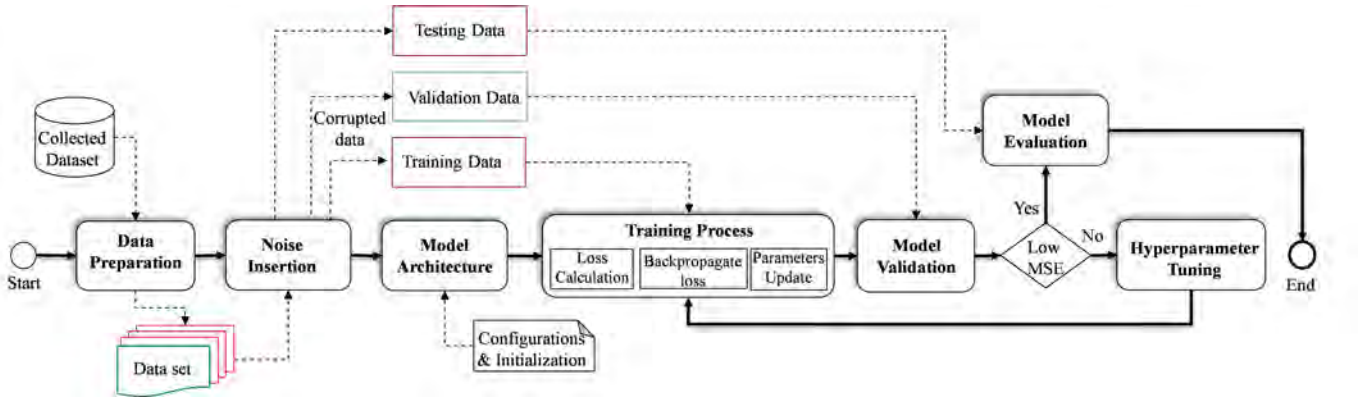
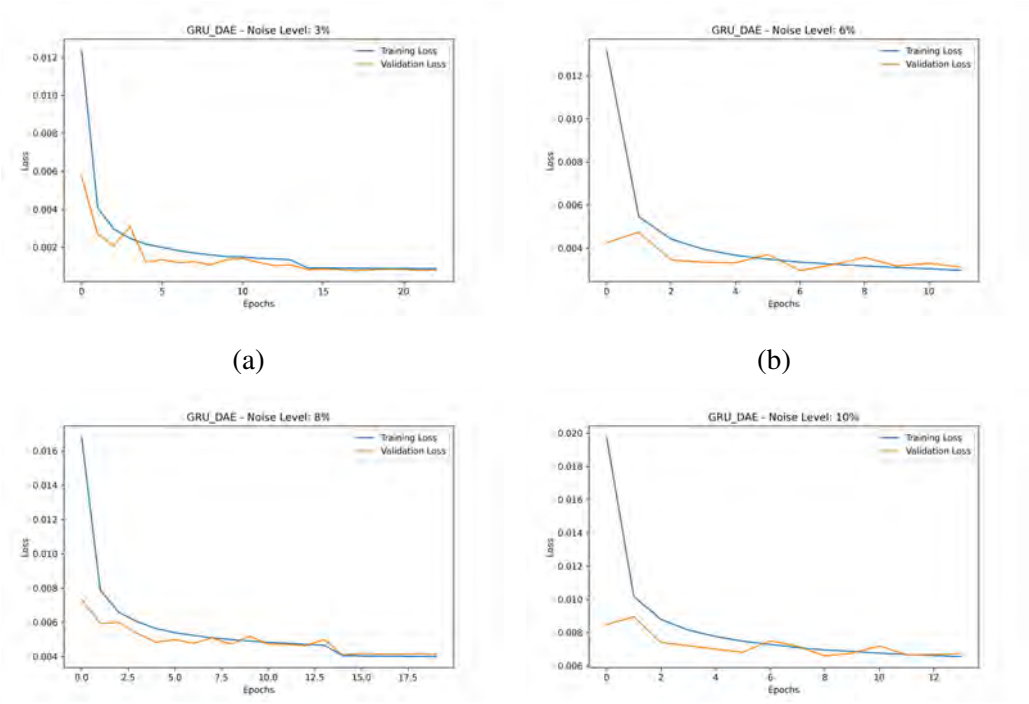


Figure 8.6: Flowchart of model training and optimization.



(a) (b)

(c) (d)

Figure 8.7: Hyperparameter optimization results with training and validation accuracy. (**a**) Under level noise 3%. (**b**) Under level noise 6%. (**c**) Under level noise 8%. (**d**) Under level noise 10%.

Table 8.3: Optimal Hyperparameters for GRU-DAE architecture.

Hyperparameter	**Optimal Values**
GRU Layers	4
Dense Layers	2
GRU Unit	384
Batch Normalization Layer	2
Epochs	850
Batch Size	128
Learning Rate	0.0001
Activation Function	adam
Optimizer	tanh
Noise Level	10%

8.4 Results and Discussion

This section presents the results of tests and evaluations conducted on the proposed architectural model, demonstrating its superiority over alternative architectural designs. Furthermore, the proposed model has demonstrated the capacity to address the issue of concurrent faults in the context of diverse noise environments.

8.4.1 Evaluation Metrics

The proposed methodology outlines a two-phase approach to the analysis of results, with the first phase focusing on feature extraction and the second on clustering. In order to evaluate the performance of GRU-DAE, the reconstruction error, which is defined as the mean square error (MSE), is employed. Conversely, to assess the performance of the clustering algorithm, the Davies–Bouldin (DB) score is employed.

The reconstruction error represents a quantitative assessment utilized to assess the extent to which a developed model is capable of accurately reproducing the underlying data pattern subsequent to the coding and decoding process. The mean squared error (MSE) [306] represents the loss function of the denoising auto-encoder (DAE) design and thus enables the reconstruction error between the original input and the reconstructed denoised data to be quantified. A lower reconstruction error indicates a greater accuracy in the data representation by the trained model. The objective of the training and optimization of the designed model is to minimize the reconstruction error. This ensures that essential features are accurately captured and reconstructed. The mathematical Equation for the MSE is presented in Equation (8.8), where X_i is the samples of the original input data, X_i' is the samples of the reconstructed output and L is the total number of data points:

$$\text{MSE} = \frac{1}{L}\Sigma(X_i - X_i')^2 \tag{8.8}$$

The DB score is a commonly utilized quality assessment measure for developing unsupervised clustering models [307]. The fundamental premise of this measure is to ascertain the degree of similarity between clusters and their respective centers. In other words, the ratio between the similarity within a cluster and the dissimilarity between clusters is employed to calculate the average of the maximum similarity of each cluster set. A lower value indicates superior clustering performance,

with precisely separated clusters. Consequently, the objective of identifying the optimal number of clusters is to minimize the ratio value, while accounting for the issue of overfitting. At the point where the inertia value is undergoing a significant change, the optimal number of clusters is considered the most appropriate. The calculation of the DB score is given in Equation (8.9). C represents the number of clusters, D is the average distance between cluster data points and its centroid, and L is distance between the centroids of clusters c_i and c_j:

$$\text{DB} = \frac{1}{C}\Sigma max\frac{(D_i - D_j)}{L(c_i, c_j)} \tag{8.9}$$

In addition to the aforementioned evaluation metrics for unlabeled data, a number of quantitative evaluation measures are employed to assess the performance of the proposed model. Specifically, the precision, recall, and F1-score are computed based on a defined set of labeled test data in order to evaluate the fault-detection and clustering performance of the developed architecture, as detailed in the evaluation methodology proposed by [291]. Precision is the percentage of total classification results that were correctly predicted. Recall represents the correctly identified elements as a percentage of all elements to be identified. Finally, the harmonic mean between precision and recall is calculated using the F1-score. Mathematically, quantitative assessment metrics are expressed by Equation (8.10)–(8.12), based on the true positive (TP), false positive (FP) and false negative (FN) values:

$$Precision = \frac{TP}{TP + FP} \tag{8.10}$$

$$Recall = \frac{TP}{TP + FN} \tag{8.11}$$

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{8.12}$$

8.4.2 GRU-DAE Performance Compared to other AE Variants

The objective of the present study in this chapter is to demonstrate the superiority of the proposed model through a comparison of its performance with that of other AE variants. In order to accomplish this, three different models—namely the ANN-DAE, and CLSTM-DAE and GRU-based DAE have been implemented. All developed models were validated using the same test samples. As demonstrated in Table 8.4, the proposed model demonstrates high performance in extracting

noise-free features, with a low MSE of 0.0073. Notably, under noise-free conditions, the ANN-DAE model exhibits a high degree of performance, with an MSE value of 0.0212.

The applicability of the proposed model in denoising the data is demonstrated at different noise levels in comparison to the other models. In this instance, four distinct levels of Gaussian noise were introduced, i.e., 3%, 6%, 8%, and 10%. Although the MSE value increases with increasing noise in our proposed model, the MSE value at a high noise level is still within an acceptable range compared to that of the other models, i.e., 0.0618 at noise 10%. In contrast, the CLSTM-DAE model exhibits suboptimal performance, with an MSE value of 0.5641, likely due to its intricate structure. In addition to its simple structure, the ANN-DAE model also demonstrates satisfactory performance at various noise levels, including 0.0316, 0.0530, 0.0706, and 0.0766. Nevertheless, further enhancements are necessary to achieve higher accuracy in comparison to the proposed model.

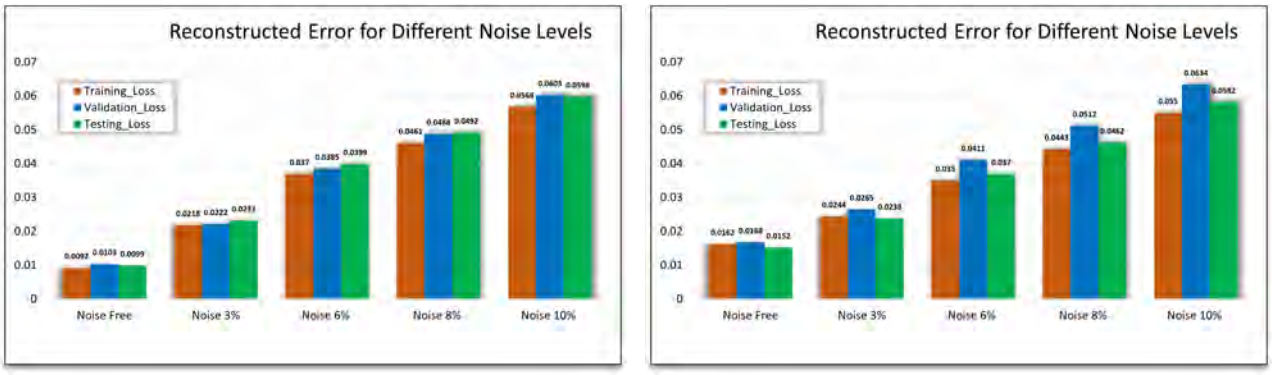
Table 8.4: Reconstruction error of GRU-DAE compared to other AE variants.

Noise Level	**ANN-DAE**	**CLSTM-DAE**	**Proposed GRU-DAE**
Noise-free	0.0212	0.5518	0.0073
Noise 3%	0.0316	0.5551	0.0234
Noise 6%	0.0530	0.5617	0.0421
Noise 8%	0.0706	0.5633	0.0504
Noise 10%	0.0766	0.5641	0.0618

8.4.3 Evaluation Results under Different Noise Levels and Fault Classes

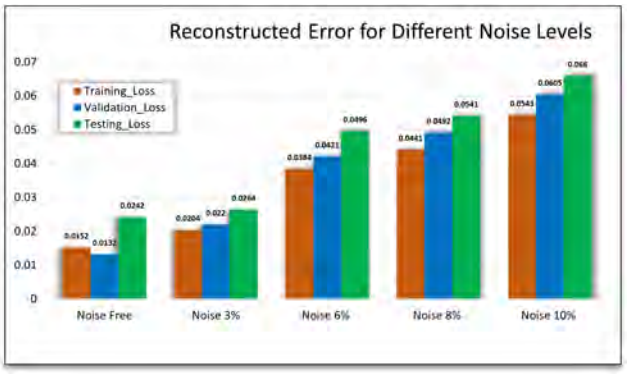
To illustrate the influence of data size and the number of fault classes on model performance, the proposed model is trained under distinct conditions. Consequently, the capacity of the proposed model to extract and reconstruct features is demonstrated under varying data sizes and numbers of fault classes. As illustrated in Figure 8.8, the model is trained on four, eight, twelve, and all CSV files under varying levels of noise, including noise-free, 3%, 6%, 8%, and 10%. The model demonstrates high performance in the presence of a limited number of fault classes, with an MSE of 0.00992 under fault-free conditions. However, as the level of noise increases, the MSE value also rises, reaching 0.0598 at 10% noise. Conversely, the impact of the size of the dataset, including

the faulty sample, can be observed in the fact that the MSE value increases as the number of fault classes increases. This phenomenon can be attributed to the increased diversity of faulty features in the case of combination faults, particularly when the data sample size is limited. Notwithstanding the presence of high noise and the inclusion of all fault classes, the model's performance remains satisfactory, with an MSE of 0.0618 at 10% noise. Consequently, the model's performance in the presence of a high degree of fault diversity can be guaranteed by increasing the size of the training data.

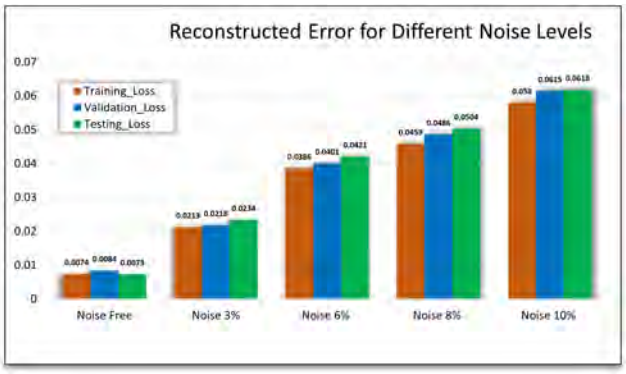


(a)

(b)



(c)



(d)

Figure 8.8: Effect of the data size and fault classes on the performance. (**a**) MSE using 25% of data with 3 fault classes. (**b**) MSE using 50% of data with 7 fault classes. (**c**) MSE using 75% of data with 11 fault classes. (**d**) MSE using 100% of data with all classes fault classes.

8.4.4 Clustering Results of Single and Concurrent Faults

In the event of the occurrence of a singular fault, as depicted in Figure 8.9, the extracted features undergo a division into two distinct clusters at the initial level of clustering, specifically, healthy and faulty. At the second clustering level, faulty features are grouped according to the faulty component. This can be either APP or RPM fault. Finally, at the third level of clustering, the features of APP faults and RPM faults are grouped into specific clusters that represent the nature of each fault. In a simultaneous fault, however, it is not possible to ascertain the precise characteristics of the faulty components. Subsequently, the faulty characteristics are grouped into subsets that represent the fault classes, which are then combined to form the concurrent faults. Figure 8.10 illustrates the cluster levels of the extracted features in the case of concurrent faults.

In order to identify the optimal number of clusters, the inertia value is employed. In this strategy, the inertia value is calculated in relation to the number of clusters. The objective is to select the optimal number of clusters such that the inertia value is minimized. Consequently, the optimal number is selected at the points where the change in inertia value is most pronounced, that is, at the elbow point. The selected number of clusters, contingent on the inertia value, is illustrated in Figures 8.11a,b for single and simultaneous faults, respectively.

As a means of assessing the efficacy of the cluster model, the DB index in Tables 8.5 and 8.6 illustrates the effectiveness with which the features are distinguished and grouped for single and concurrent faults, respectively. In the case of a single fault, the low value of DB is indicative of a high level of performance at all levels of the cluster. At the initial level, the DB score is 0.6442, whereas at the second level, it is 0.6875. This represents a marginal enhancement, likely attributable to the similarities in the features of the faulty components. The optimal clustering of faulty features of APP yielded to low DB index score of 0.5695.

In the case of concurrent faults, the complexity of the extracted features increases, leading to a significant increase in the DB score. This is in contrast to the single fault case, where the DB score is lower. The DB score in particular was 0.7838 at the initial stage, while in the subsequent stage, the DB score was 0.7859. Notwithstanding the aforementioned values illustrate the high performance of clustering, also in the presence of simultaneous combination faults, compared to the conventional K-means.

Clustering Level	GRU DAE + K-Means	Stand-Alone K-Means
Level 1	0.6442	1.2541
Level 2	0.6875	0.9560
Level 3	0.7170	0.9586
Level 3	0.5695	0.9312

Table 8.5: Davies–Bouldin scores of the proposed model for single fault.

Clustering Level	GRU DAE + K-Means	Stand-Alone K-Means
Level 1	0.7838	2.0676
Level 2	0.7859	1.2288

Table 8.6: Davies–Bouldin scores of the proposed model for concurrent faults.

In conclusion, the application of our proposed approach enables the training of robust representations against the noise of the input, thereby facilitating more accurate fault clustering and pattern identification, even in the presence of noise.

8.4.5 Fault-Detection and Clustering Results of Case Study 2

In the context of developing DL-based FDD models, ensuring the developed model is applicable in real industrial applications is of paramount importance. Consequently, the objective of this study is to record and categorize the dataset of system behavior under normal and anomalous conditions as a basis for testing, utilizing real-time manual driving as a representative scenario. The virtual driving tests recorded in this context include tests conducted in the presence of healthy, single, and simultaneous faults. It should be noted that due to the inherent variability in driver behavior and the complexity of real-world conditions, the test data is regarded as samples of uncertainty.

The test results indicate that the proposed model is an effective and superior approach for the reconstruction of original, denoised data with minimal loss, as evidenced by a MSE of 0.0955. The MSE demonstrates that the proposed model is capable of processing uncertain, real-world datasets containing noise with a low error. In addition, the clustering phase represents a notable achievement, as it enables the identification of faulty data, which is then clustered separately. In other words, the proposed model is capable of accurately detecting faults in single and concurrent occurrences with low DB scores of 0.4872 and 0.5617, respectively. The comparatively low DB

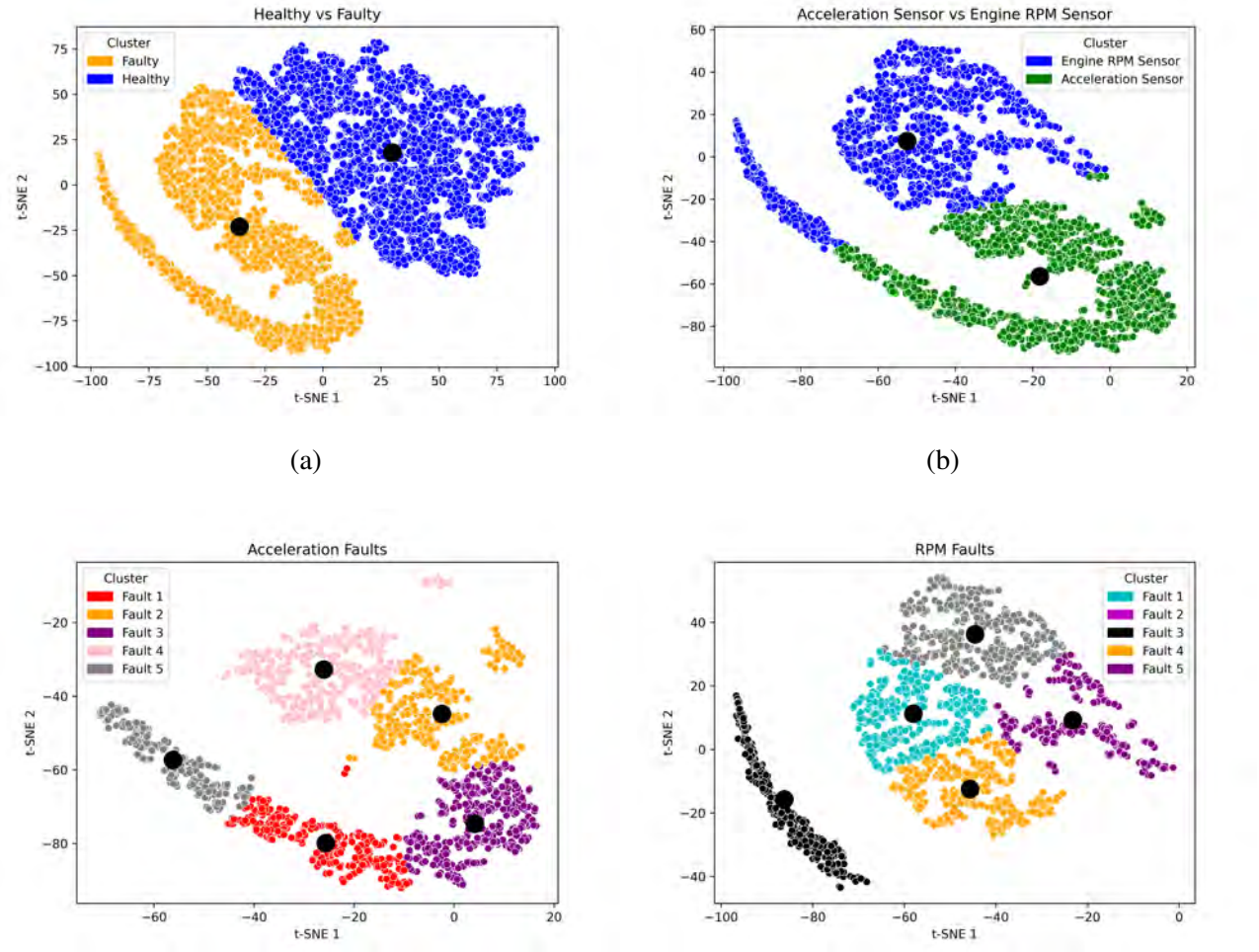
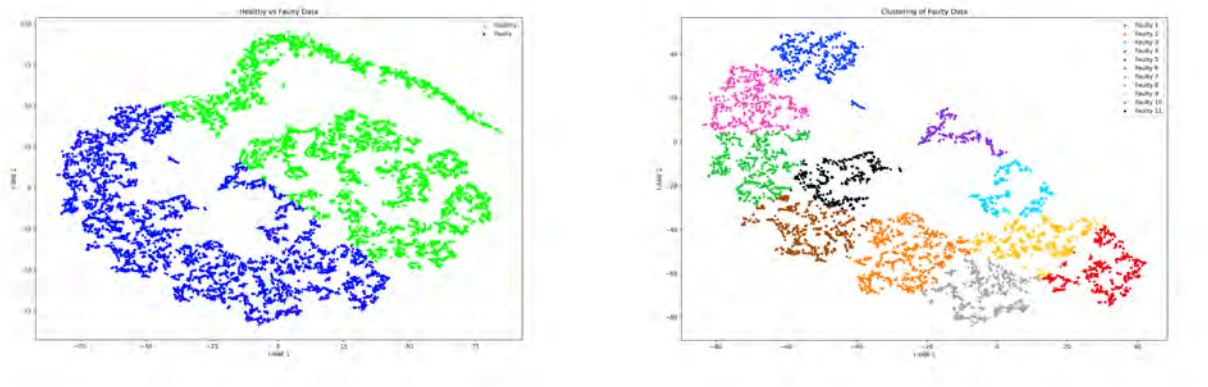
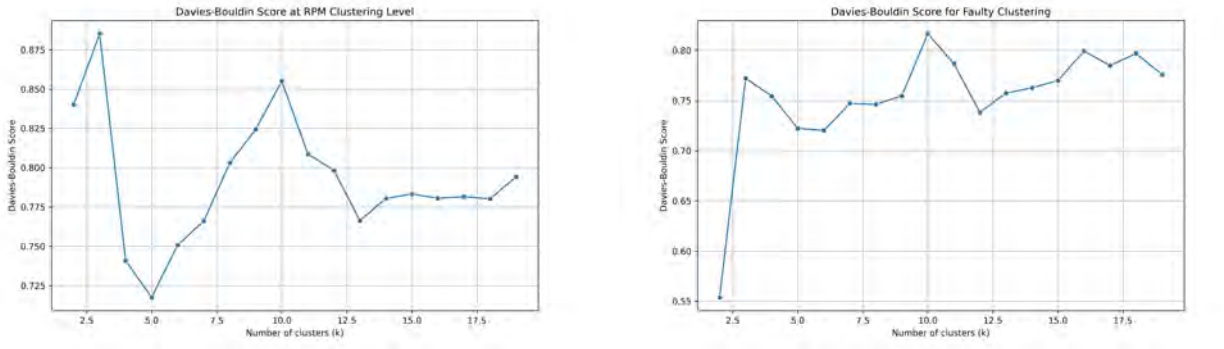


Figure 8.9: Feature visualization of the multi-level clustering for single fault including the clusters' centres. (**a**) Level 1 clustering of the faulty features. (**b**) Level 2 clustering of the faulty components features. (**c**) Level 3 clustering of the fault types features. (**d**) Level 3 clustering of the fault types features.



(a) (b)

Figure 8.10: Feature visualization of the multi-level clustering for concurrent faults. (**a**) Level 1 clustering of the faulty features. (**b**) Level 2 clustering of the fault types features.



(a) (b)

Figure 8.11: Optimization of number of clusters selection. (**a**) Clusters selection for single fault clustering. (**b**) Clusters selection for concurrent faults clustering.

score for detected faults implies that the detection performance of the proposed model is superior in the case of single faults compared to concurrent faults. This can be attributed to the considerable intricacy of the representative characteristics of simultaneous fault signal patterns.

In order to assess the efficacy of the proposed method in terms of detection and clustering, three quantitative performance criteria are employed, i.e., recall, precision and F1-score. To achieve this objective, the previously described dataset comprising real-time virtual test drive data labelled as healthy or faulty is employed. In the case of an individual fault, the method exhibits remarkable performance in terms of detection, with a precision of 99.17%, a recall of 95.23%, and an F1-score of 97.16%. The findings of the analysis indicate that single faults can be accurately identified with a high level of accuracy in terms of positive predictive value (PPV) and true positive rate (TPR). Moreover, the high detection accuracy in the case of concurrent faults demonstrates the reliability of the proposed model. The precision, recall, and F1-score are 92.83%, 98.09%, and 95.38%, respectively. It is essential to highlight the occurrence of false positives in the case of compound faults. This is due to a considerable number of test data samples being incorrectly clustered as faulty. This discrepancy can be attributed to the similarity of data samples in certain instances between the healthy and faulty samples. To address this issue more effectively, it is recommended to conduct experiments-based failure analysis in order to define the threshold for behavioral deviations that are indicative of faults. Furthermore, it is important to note that the aforementioned degree of accuracy is also influenced by the complexity of the fault patterns themselves. Consequently, data samples pertaining to fault types may exhibit similarities, which subsequently results in inaccurate diagnosis outcomes.

8.5 Conclusions

The objective of this chapter is to introduce a novel DL-based method designed to address a critical challenge encountered in the validation process of ASSs, i.e., the detection and grouping of unknown sensor faults. The core of the developed method is the adoption of a GRU-based DAE with the k-means algorithm. The denoised extracted features generated by the proposed method have been empirically validated to significantly enhance the detection and clustering process in the presence of noise. Consequently, the reliability and robustness of the clustering model for single

and simultaneous faults under different levels of noise can be guaranteed. When compared to alternative restructuring models, the proposed architecture exhibits a marked advantage in terms of reconstruction error under noisy conditions. The efficacy of the proposed method is validated by employing real-time simulation data from two automotive case studies. One case involves a gasoline engine and the other a dynamic vehicle system with traffic. The results of the experimental study demonstrate that the proposed model is not only capable of extracting representative features with greater efficiency than standalone techniques but also of clustering faults with greater accuracy than other techniques. Specifically, the results illustrate that the low value of DB index exhibits high performance in the clustering process of single and simultaneous faults, with values of 0.4872 and 0.5617, respectively. A quantitative evaluation was conducted to ascertain the effectiveness of the model in terms of diagnosis. This was achieved by employing metrics on a testing dataset derived from virtual testing drives. Both instances of fault occurrence, individually and simultaneously, were effectively identified and clustered by the model, with an average accuracy rate of 97.16% and 95.38% respectively. In conclusion, it can be stated that the proposed model is capable of identifying potential risks and vulnerabilities at an early stage of the system development process. This ultimately results in enhanced safety and reliability, as well as a reduction in the costs and resources required for real-time validation process of ASSs.

9 Detection and Identification of Known Simultaneous Faults

Solving the problem of detecting and classifying concurrent faults during system integration and testing, which is formulated in the research question RQ4, is the key point of this chapter. Therefore, an intelligent approach based on ensemble ML methods is proposed in this chapter, which aims at answering RQ4. The rest of the chapter is organized as follows. An overview of the chapter is given in Section 9.1. The architecture of the proposed ensemble FDD model is presented in Section 9.2. The dataset used and the implementation steps are presented in Section 9.3. The findings of the experimental evaluation are analyzed and discussed in detail in Section 9.4. Finally, the conclusion is presented in Section 9.5.

Note: It is important to highlight that the proposed methodology and the corresponding implementation outcomes presented in this chapter have been published in [308]

9.1 Chapter Overview

In compliance with the ISO 26262 standard, functional validation of ASSs under development represents a crucial aspect for ensuring the safety and reliability of these systems. HIL is a reliable, safe, and flexible test platform that enables the validation process in a timely manner. Conventional failure analysis techniques during real-time validation are time-consuming, arduous, and costly. Consequently, there is a pressing need for an intelligent solution that can overcome the aforementioned challenges. The development of DL methods for fault detection and classification is becoming increasingly prevalent. Despite their efficacy, the majority of previous studies have been conducted for single faults, with noise being largely overlooked. In this chapter, a novel method for identifying and classifying concurrent faults under noisy conditions is developed. This approach utilizes both LSTM and RF methods. Additionally, a GRU-based DAE is employed to enhance

model resilience against noise. Furthermore, to address the imbalanced nature of the data, an undersampling algorithm is employed. This approach facilitates the detection and identification of individual and concurrent sensor faults that occur throughout the system integration and testing of ASSs in an efficient and automatic manner. A preliminary evaluation of the efficacy and resilience of the proposed approach was conducted through a simulation of the dynamic vehicle system and driving conditions. The findings of the analysis indicate that the proposed model is capable of attaining a high degree of accuracy in the presence of noise, with an average detection accuracy of 99.43%. Moreover, the proposed ensemble learning architecture with DAE offers a more promising fault identification performance with enhanced accuracy and resilience when compared to the individual methods. The results of the testing indicate that the proposed model demonstrates greater effectiveness than other state-of-the-art methods in identifying simultaneous faults, achieving an F1-score of 91.2%.

9.2 Proposed Methodology

The proposed architecture is designed to facilitate the detection and identification of single and simultaneous faults in the sensors of ASSs, particularly in the presence of noisy and unbalanced data conditions. The proposed model is designed for use during the development phase of ASSs, specifically for real-time system validation. Accordingly, the process of failure analysis during the aforementioned test phase is enhanced. Consequently, the FDD of sensor faults is achieved in an efficient manner, thereby reducing both the time and effort required for the analysis process. The proposed FDD architectural design comprises four principal phases: data acquisition, data preparation, data noise reduction, and feature learning. These are depicted in Figure 9.1.

9.2.1 Data Acquisition

To assure the fidelity of the captured system behavior in a variety of circumstances, a real-time HIL simulation platform is employed. The interaction of the developed ECUs with other system components, including other ECUs, sensors and actuators, controlled devices, and the vehicle network, is simulated in order to ascertain their behavior. Moreover, a significant number of representative and pertinent test kilometers can be conducted at a relatively low cost and with a high level of

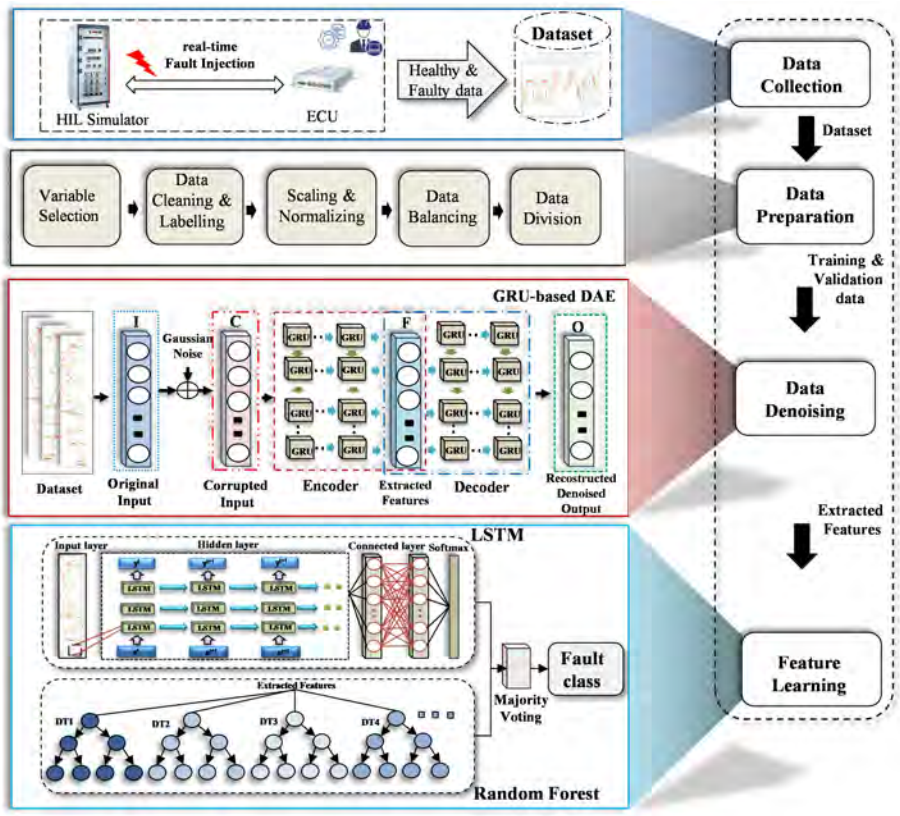


Figure 9.1: Proposed ensemble learning-based simultaneous FDD methodology

safety in comparison to real test drives. The HIL platform's logging system enables the recording of target system variables as multivariate time-series data during the virtual test drive. The HIL simulator, the developed target ECU, the real wheel and pedals, the CAN bus communication, and the real-time FI framework represent the principal elements of the HIL system in this study.

By running the system under fault-free conditions, i.e., desired/standard behavior, healthy data samples can be obtained. In order to achieve this objective, the signals emanating from the sensors and actuators that are accessible via the CAN bus are recorded. The use of a high-fidelity automotive simulation model within a HIL simulator enables the collection of high-quality datasets under real-time constraints. Furthermore, the real-time FI framework proposed in Chapter 6 facilitates the generation of representative faulty and healthy data in real-time. This is accomplished by executing the target system under conditions of random sensor/actuator fault. The aforementioned framework permits the injection of faults over the CAN bus without modifying the original system architecture. This guarantees the real-time execution of both the ECU and the plant as a blackbox

execution without the necessity for system design modification. It is important to note that in order for the FI process to be initiated, three attributes should be specified, i.e., the FI time, the fault types, and the fault locations. The potential location of fault occurrence, such as sensor, actuator, network, or controller, can be identified based on the system architecture components, as outlined in [158]. In addition, various sequential data-related faults, such as hard-over, noise, gain, offset, stuck-at, delay, drift, packet loss, and spike faults, can be introduced as fault types [309]. FI may be either permanent or temporary, and may occur individually or simultaneously as multiple faults over time. Consequently, the timing and duration of the FI are of critical importance in generating representative fault data. For instance, transient faults result in imbalanced data, with a disparate ratio of faulty to healthy samples. It is of particular note that simultaneous fault data is the result of the simultaneous injection of two different faults at different locations. A variety of factors may give rise to faults within or between system components.

9.2.2 Data Preparation

Once the representative dataset has been obtained, the data undergoes pre-processing during the data preparation phase. In this phase, the training process is enhanced by reducing the computational cost and preventing overfitting [310]. This is achieved by reducing the irrelevant data and correcting the missing samples. These include the selection of variables, the cleaning of data, the labelling of data, the scaling and normalization of data, the balancing of data, and the partitioning of data.

By employing a simulation that accurately represents the vehicle system, it is possible to measure a significant number of system variables. It is therefore crucial to select the variables that are pivotal in determining the status of the system, that is, whether it is healthy or malfunctioning. In this study, a selection of sensor signals at the system level were employed as inputs for the desired FDD model. The primary variables of the FDD model include engine speed, engine temperature, engine torque, intake manifold pressure, vehicle speed, throttle position, and rail pressure. In this manner, the model can be trained using relevant healthy and faults features. Subsequently, the dataset is subjected to filtration and cleansing procedures with the objective of enhancing its quality by eliminating any negative factors that may have arisen during its generation. In particular, the objective of the cleaning process is to remove any outliers and to compensate for the missing

samples in the dataset. Prior to the training process, data labeling takes place as the FDD model is developed using a supervised learning approach. This is the stage at which the classes are assigned to the corresponding data in order to facilitate classification. In the event of simultaneous faults, it is necessary to consider two classes simultaneously during the labeling process. Accordingly, in order to encompass the potential fault combinations, a multi-class, multi-label approach [311] was considered in this study. However, in order to circumvent the challenge of manual labeling, a data dictionary process has been developed for the automatic identification of all possible fault combinations. This approach allows for the creation of a unique numerical label with a specific index for each possible pair of fault types. It is important to note that the sensor signals in the ASSs exhibit different value ranges. Consequently, the amplitudes of the variables are standardized and scaled uniformly to the range [0,1] using the Z-scaling function. The mathematical scaling process of the input values is described in detail in [312].

The presence of imbalanced data, where the ratio of defective to healthy samples is disproportionate, also presents a challenge during the training process. This may result in a trained classifier exhibiting a tendency to favour the majority class, which may in turn lead to suboptimal predictive performance [313]. To address this issue, various approaches have been proposed in the literature, including augmentation-based, feature learning-based, and classifier design-based approaches [180]. To address the issue of imbalanced data, the random undersampling method was employed in this study. The fundamental concept underlying this methodology is the random selection and removal of instances from the majority class until the requisite class balance is achieved, as described in [314]. Given the abundance of healthy samples available for use in the HIL testing, the reduction of the dataset by the aforementioned balancing method does not negatively impact the training process. The rationale behind the selection of this technique, in comparison to other adjustment techniques such as random oversampling and SMOTE [315], is that they may potentially lead to overfitting due to their inability to align with the original patterns of the time series. Finally, the adjusted data are divided into a training, validation, and test subset. In particular, the dataset is divided into three subsets: 80% for training, 10% for validation, and 10% for testing.

9.2.3 DAE-based Data Denoising

In the real world, the presence of noise within datasets, along with the uncertainty that often accompanies such data sources, has a deleterious impact upon the decision-making processes that rely upon data-driven approach. It is notable that the majority of developed models are based on clean historical simulation data obtained under experimental laboratory conditions. Recently, the DAE has emerged as a highly effective approach to address the challenge of noisy data, as evidenced by the work of [144]. The DAE model is constructed on the basis of four distinct layers: the input layer, the corrupted layer, the hidden layer, and the output layer. The hidden layer architecture comprises two principal modules, i.e., the encoder and the decoder. In order to take advantage of the denoising function of the DAE, the original input data (I) should be corrupted with a certain level of noise, such as Gaussian noise. The coder's objective is to map the input data into a lower dimension, namely a latent code, where the extracted features (F) from the distorted input data (C) are stored. The reconstruction of the extracted features into the original data space is performed by the decoding module, which is referred to as the reconstructed output (O). Consequently, it is possible to employ a condensed representation of the input data for numerous applications, including the processes of dimensionality reduction, denoising of data, and generative modelling. Mathematically, the Equations (9.1) and (9.2) represent the encoder module and the decoder module, respectively.

$$F = f_{enc}(WC + b) \tag{9.1}$$

$$O = f_{dec}(W'F + b') \tag{9.2}$$

In this context, the notation f_{enc} and f_{dec} refer to the activation functions of the encoder and the decoder, respectively. The weight of the encoder and the decoder, respectively, are denoted by W and W'. b and b' represent the offset vectors of the encoder and decoder, respectively.

The training approach of the DAE model is founded upon a fundamental principle, i.e., the minimization of the reconstruction loss between the input data (I) and the reconstructed output (O). Thus, the encoder and decoder modules undergo training in which they represent the data effectively and reconstruct it without introducing any noise. The performance of the DAE can be

quantified by means of the reconstruction error, which is commonly referred to as a loss function. This can be demonstrated in Equation (9.3).

$$L(I,O) = ||O - I||^2 \tag{9.3}$$

Autoencoders can be designed based on different architectures, including AE based on DNN, AE based on CNN, or AE based on LSTM. In this study, a GRU-based DAE was employed to enhance the resilience of the proposed FDD model against noise in accordance with the notable accomplishments of GRU-based DAE in comparison to other AE variants [302]. Furthermore, the rationale for selecting the GRU cell was its capacity to fulfill the necessities of real-time applications with constrained resources and rapid inference times. The internal structure of the GRU, specifically the gate mechanisms, enables the flow of information to be controlled, resulting in a lower computational effort and lower training parameters compared to the LSTM cell. Further insight into the mathematical representation of the GRU cell Equations can be found in [302].

9.2.4 Ensemble Learning-based Fault Detection and Identification

Following the denoising of the data with the DAE algorithm, the correlation between the features of time series data is exploited in order to identify the respective fault type. This is significant because different fault characteristics can belong to two distinct classes simultaneously. To address the potential for biased classification accuracy inherent to a single classifier, ensemble learning, as detailed in [316], is employed in the proposed architecture. In order to achieve this, an ensemble learning approach based on the LSTM and RF classifiers is employed. Notably, the voter mechanism is used to select the predicted output with the highest probability from both classifiers. Consequently, the discrepancy in classification accuracy can be reduced, and the process of identifying faults can be enhanced.

It has been demonstrated that the LSTM is not only adept at establishing long-term relationships in sequential data, but also at resolving the vanishing gradient and exploding gradient issues commonly encountered in RNN models. Moreover, LSTM has demonstrated superior performance on large datasets for supervised classification problems in comparison to other techniques, such as CNN, MLP, and GRU [283]. Furthermore, LSTM is regarded as a highly effective technique

for processing complex nonlinear data in higher-dimensional, noisy spaces, offering high levels of accuracy, reliability, and effectiveness in the context of FDD [283]. Given the scope of the study and the nature of the data, LSTM was selected for the development of the FDD model. The fundamental structure of the model is the LSTM cell, which is constructed on the basis of three gates: the input gate, the forgetting gate, and the output gate, as illustrated in Figure 9.2.

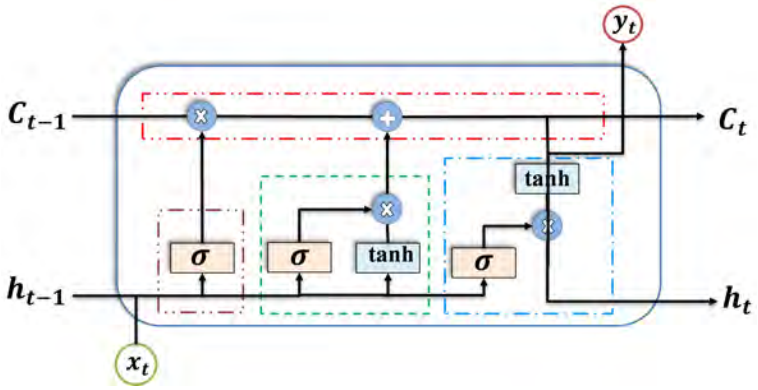


Figure 9.2: Internal structure of LSTM cell.

In addition to the aforementioned gates, a memory, referred to as the cell state, is employed by the LSTM network to enable the retention of information over extended sequences. The internal structure of the cell enables the precise regulation of information flow. Specifically, it determines the extent to which new information is discarded, retained, or reinforced. Subsequently, the gating mechanism and memory cell enable the effective addressing of tasks involving long-range dependencies. The output of the input gate, as represented by Equation (9.4), is the input to the forget gate, as determined by Equation (9.5), where ft is a value between 0 and 1. A value of 0 indicates that the data is removed, while a value of 1 indicates that the data is retained. Two distinct activation functions are employed to update the hidden state, namely (σ) and ($tanh$). The output of the current state, denoted by C_t, is determined by Equation (9.7), where the previous state output, represented by C_{t-1}, is utilized.

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_g) \tag{9.4}$$

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{9.5}$$

$$d_t = tanh(W_d[h_{t-1}, x_t] + b_d) \tag{9.6}$$

$$C_t = (f_t \times C_{t-1}) + (g_t \times d_t) \tag{9.7}$$

It is notable that the value of the gate is dependent upon the hidden state, denoted by h_{t-1}, and the input, denoted by x_t. Finally, the output cell and output gate can be mathematically represented by the Equation (9.9) and Equation (9.8), respectively.

$$O_t = \sigma(W_O.[h_{t-1}, x_t] + b_O) \tag{9.8}$$

$$h_t = O_t \times tanh(C_t) \tag{9.9}$$

The RF ensemble learning method, which involves multiple decision trees, is presented as an efficient classification technique that avoids overfitting problems. Among the various ML classifiers, RF is demonstrably superior to other techniques, such as k-nearest neighbor (KNN) and SVM, in terms of performance and running time. In addition to its resistance to overfitting, it exhibits outstanding performance in processing a large set of features, as evidenced by the findings of Belgiu et al. in [134]. The RF classifier has achieved remarkable success in a variety of applications due to its simplicity, speed, robustness, and ability to handle missing data [317, 318]. Although there are numerous advantages to the use of RF, one significant disadvantage is its inability to extract features and to capture the inherent time dependency present in time series data. Consequently, the representative features extracted by LSTM are employed in this study and provided to RF in order to guarantee optimal predictive performance. In essence, the fundamental principle underlying the operation of the RF is the aggregation of the predictions of all the trees that are formed, based on three distinct types of nodes, i.e., the root node, the decision node, and the leaf node. The ultimate determination is reached by aggregating the predictions of all the trees through a majority voting process. The mathematical representation of the prediction output is expressed in Equation (9.10).

$$T = Mode T_1(x), T_2(x), ..., T_i(x) \tag{9.10}$$

The predicted output is represented by the symbol T. The mode is the majority voting operation, and $T_i(x)$ represents the prediction of the i-th decision tree (DT) in the forest.

The construction of an RF tree is dependent upon the bagging technique, which entails the generation of multiple bootstrap samples from the original training dataset. This process serves to enhance the stability and accuracy of the resulting tree. The training process employs a random

subset of the data and features with the same ratio, and is performed separately for each individual tree. Upon reaching the predefined criterion, such as the maximum depth or minimum number of samples in a leaf node, the training process is concluded.

Finally, the classification decision with the highest score is made by combining the prediction results of the LSTM and the RF using the majority voting method [319]. Consequently, the combination of both classifiers enhances the prediction performance and robustness, and the risk of overfitting is mitigated. Furthermore, the uncertainty of the model can be accounted for by considering the individual predictions of each classifier.

9.3 Case Study and Experimental Implementation

This section presents the details of the system architectural of the case study, platform setup, and implementation procedures. The section also delineates the principal phases and steps in the development of the proposed FDD model, i.e., data creation, data preprocessing, model training, and testing.

It should be stated that the validation process of the approach proposed in this chapter, including the case study and the experimental setup, has been performed using the virtual test framework developed in chapter 10.

9.3.1 System Architecture of the Case Study

To validate the target FDD model and demonstrate its capability, two system models provided by dSPACE were used, i.e, the ASM Gasoline Engine and the ASM Vehicle Dynamics with Traffic [166]. Specifically, these models were modified and integrated to enable a virtual test drive, which was designed to ensure comprehensive engine system characteristics.

At the software level, in order to capture comprehensive characteristics of the system behavior, a comprehensive modeling and simulation of the engine system was conducted in the MATLAB/Simulink environment with high fidelity. Specifically, the model architecture incorporated detailed subsystems of the engine, encompassing their respective components and interconnections. The primary subsystems of the engine include the airflow system, the fuel system, the piston system, the exhaust system, and the radiator system, as illustrated in Figure 9.3. Furthermore, the

interaction of the target systems with their environment has been considered using the Model-Based Design (MBD) approach, whereby the powertrain, vehicle dynamics systems, and the driving environment have been modelled. This approach enables the identification of the most significant characteristics of the entire vehicle, including longitudinal driving, vehicle resistances, transmission and driver characteristics. Finally, the control algorithm, which is the SUT, was designed as a behavioral model called SoftECU. This is connected directly to the controlled system in a control loop.

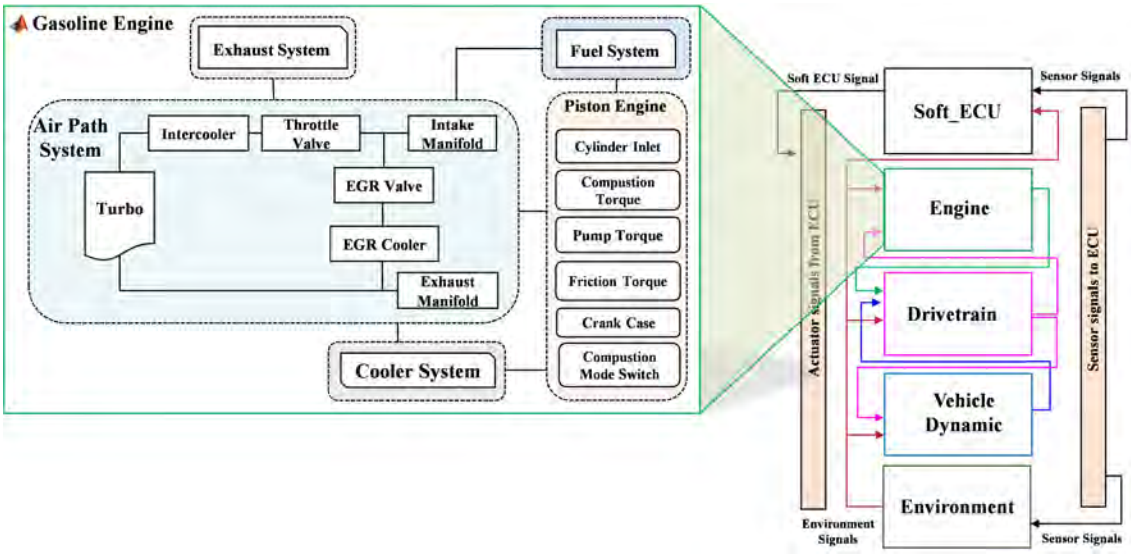
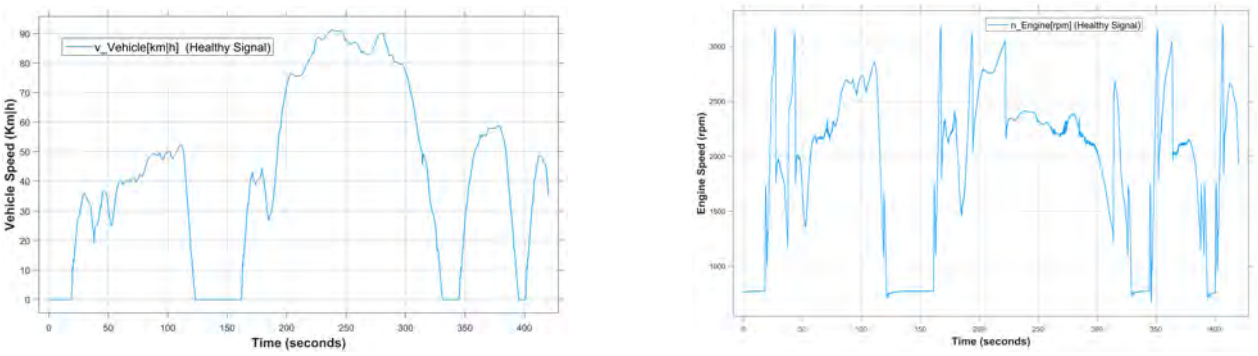


Figure 9.3: System architecture of the used case study.

At the hardware level, the MicroAutoBox II is employed in this study as a rapid control prototype (RCP) for the emulation of the real ECU functionality and the execution of the control algorithms. The RCP (DS1401 base board) is equipped with a 900 MHz processor, 6th Gen. Intel® CoreTM i7-6822EQ, 16 MB memory, and a boot time of 340 ms for a 3 MB application. dSPACE SCALEXIO is employed as a real-time simulator for comprehensive and accurate simulation of the complex controlled system, i.e., the gasoline engine with vehicle dynamics. The transfer of sensor and actuator signals between the real-time simulator and MicroAutoBox II is conducted via a CAN bus, with the host PC being connected to both devices via Ethernet. The real wheel and pedals are connected to the HIL system, thus enabling the virtual test drive to take the user's behavior into account. This functionality enables the machine to execute driving tasks in either an automated or manual mode at the discretion of the user. A virtual driving environment with dynamic traffic was



(a) (b)

Figure 9.4: Selected test drive scenario as a desired behavior (golden run). (**a**) Vehicle system behavior under fault-free conditions. (**b**) Engine system behavior under fault-free conditions.

designed and modeled using ModelDesk. Furthermore, a three-dimensional visualization of the environment was made possible with MotionDesk.

It is recommended that the parameters of the modeler be configured prior to running it on the target computer. This is accomplished through the use of ModelDesk, which enables the user to specify both the internal and external system specifications as required. The core specifications of the selected case study are presented in Chapter 6, 10.

9.3.2 Representative Dataset Generation

In accordance with the principles of golden run behavior, the system is executed in real-time under conditions that are free of faults. This process gathers the healthy dataset, which includes sensor and actuator signals, representing the normal behavior of the SUT. In order to select the desired driving scenarios, i.e., the "city" and "highway" scenarios, the ControlDesk tool was utilized. Figures 9.4a and 9.4b illustrate the vehicle and engine behavior under healthy and fault conditions, respectively.

By contrast, the faults dataset was generated via the injection of several types of faults, both individually and concurrently, into the sensor signal during the course of real-time execution. This process was carried out using a real-time FI framework. A methodology has been devised to define the attributes of the FI in such a way that representative and realistic sensor faults, both permanent and transient, can be injected into the driving cycle during the execution of the test. This approach permits the capture of the effect of critical faults that cause system-level faults in real-time.

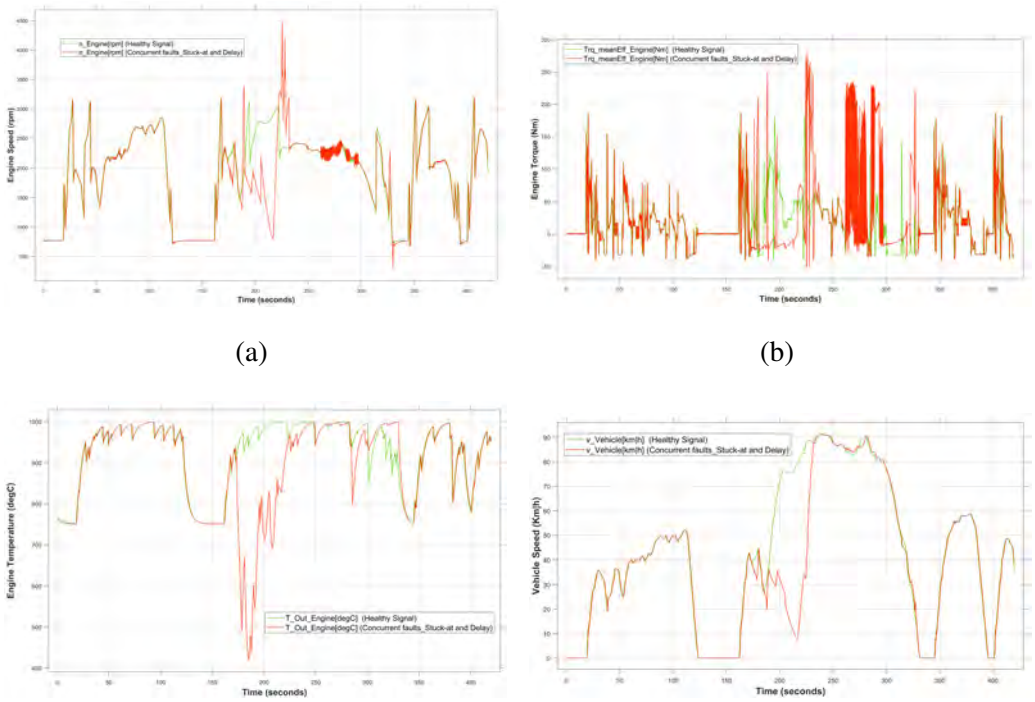
In this study, five types of faults and their combinations have been considered with the aim of injecting the majority of the sensor faults present in the time series data. In particular, the following fault types have been considered: drift, gain, stuck-at, noise, and delay faults. The occurrence of faults in the accelerator pedal position (APP) sensor and the engine speed (RPM) sensor can have a significant impact on the vehicle's safety [320]. Accordingly, the APP sensor and the RPM sensor have been identified as the most probable locations for faults in the target system. This study has focused on transient faults, as the objective of the target FDD model is to address data imbalance.

The combination of the aforementioned fault types was introduced into the target location with the objective of analyzing and capturing the system behavior under simultaneous faults. To illustrate, in each experiment, two distinct types of faults, such as gain and noise, were introduced into the APP and RPM sensor, respectively, for a specified duration. In this manner, the single and simultaneous faults were introduced for a specified duration and then deactivated after a brief interval. Based on the selected driving scenario, the faults were injected for a duration of 170 to 330 seconds. In total, 15 distinct classes of faults, including both single and combination faults, were collected from 15 separate FI experiments. The system's behavior under simultaneous faults can be observed in Figure 9.5. In particular, Figures 9.5 a,b,c and d illustrate the impact of simultaneous injection of stuck-at and delay faults on engine torque, engine temperature, engine speed, and vehicle speed.

9.3.3 Dataset Description and Prepossessing

The recorded data is stored in CSV format, which facilitates the subsequent data pre-processing steps. A total of 15 experiments of FI yielded 16 CSV files containing faults data, in addition to the healthy data samples. The sampling time in all experiments is 0.001 seconds. The total number of data samples collected is 44,800,000, with 2,800,000 samples per experiment.

Once the data has been collected from the sensors, it is subjected to pre-processing, as detailed in the methodology section. The objective of this phase is to clean and format the collected data in a manner that allows the target model to be trained in an efficient manner. The efficacy of the model resulting from the training process is contingent upon the quality of the data. In order to achieve this objective, a number of techniques were applied to the data that had been collected. First, the data were cleaned by identifying and treating the missing values, as well as removing the outliers



(a) (b)

(c) (d)

Figure 9.5: The effect of transient simultaneous faults on the system behavior. (**a**) Engine Speed behavior under stuck-at and delay faults. (**b**) Engine torque behavior under stuck-at and delay faults. (**c**) Engine temperature behavior under stuck-at and delay faults. (**d**) Vehicle Speed behavior under stuck-at and delay faults.

and duplicate samples in the dataset. Secondly, in order to define the data distribution, patterns, and relationships among the variables, the data are visualized and analysed using the Simulation Data Inspector from MATLAB. The most significant system variables that influence the performance of the model were identified, as the selection of features plays a pivotal role in the classification tasks performed by the trained model. The variables considered in this study are rail pressure (Bar), throttle position (%), engine torque (Nm), manifold pressure (Pa), engine temperature (degC), engine speed (rpm), and vehicle velocity (km/h), as illustrated in Figure 9.6. The subsequent stage of data labeling involves the conversion of categorical variables into numerical representations. The current study employs a Label Power Set (LPS) based on a multi-label, multi-class strategy for the labeling process. The aforementioned approach allows for the representation of all potential pairs of fault types. The Z-score function also normalizes and scales values within the range $[0-1]$.

Additionally, the random downsampling technique is employed to address the issue of imbalanced data. Finally, the dataset was divided into three sections: training, validation, and testing. A total of 80% of the dataset was allocated for the training phase, while 20% was allocated for the validation and testing phases, respectively. The detailed distribution of the collected dataset is presented in Table 9.1.To provide the encoder module with corrupted samples, a certain degree of noise is incorporated into the processed data, thereby enabling the DAE to be trained effectively.

	A	B	C	D	E	F	G	H
1	Time	Pos_Throttle[%]	T_Out_Engine[degC]	Trq_meanEff_Engine[Nm]	n_Engine[rpm]	p_InMan[Pa]	p_Rail[bar]	v_Vehicle[km|h]
2	0	100	766.1545971	1.001660417	773.430293	99828.20251	119.9560258	2.88739E-43
3	0.0010023	100	766.1533018	1.001438454	772.922836	99828.30085	119.9572804	2.85851E-43
4	0.0019988	100	766.1519049	1.000509247	772.3807066	99828.40558	119.9582441	2.82993E-43
5	0.0029996	100	766.1504	0.99902058	771.7953642	99828.51573	119.958902	2.80163E-43
6	0.0039989	100	766.1487788	0.99713677	771.1613802	99828.63306	119.9592337	2.77361E-43
7	0.0050007	100	766.147031	0.997499456	770.5228867	99828.75945	119.9592149	2.74588E-43
8	0.006002	100	766.1451458	1.008464126	770.0810469	99828.89622	119.9588339	2.71842E-43
9	0.007004	100	766.1431257	1.018724529	769.8285704	99829.03592	119.9581527	2.69123E-43
10	0.0080013	100	766.1410297	1.0282161	769.7560543	99829.14178	119.957283	2.66432E-43
11	0.0090023	100	766.1388969	1.036856349	769.8515836	99829.20757	119.9563308	2.63768E-43
12	0.0100007	100	766.1367643	1.044546959	770.1004704	99829.23369	119.9553956	2.6113E-43
13	0.0110003	100	766.1346663	1.051163337	770.4847562	99829.22231	119.9545681	2.58519E-43
14	0.0119994	100	766.1326339	1.056609454	770.9837285	99829.17644	119.9539295	2.55934E-43
15	0.013	100	766.130694	1.060770541	771.573572	99829.09983	119.9535491	2.53374E-43
16	0.0139996	100	766.1288692	1.063543846	772.2276007	99828.99686	119.9534831	2.5084E-43

Figure 9.6: Collected dataset from real-time HIL Simulation.

9.3.4 Training and Optimization

The method proposed has been implemented in Google Colab platform, utilizing the TensorFlow framework [293] and Python programming language. The DL model was trained on the basis of the balanced training data, fed in from the pre-processing phase. Figure 9.7 depicts a schematic overview of the data preprocessing and balancing stages, which serve as the foundation for the four subsequent phases of model development, i.e, model design, model training, model validation, and model evaluation. The model design phase involves the identification of the network architecture parameters and configurations. This also includes the initialization of the model hyperparameters, i.e., number of layers, epoch, learning rate, batch size, noise level, activation function, and optimizer.

It is important to note that in the context of DAE training, in addition to the hyperparameters previously mentioned, Gaussian noise levels are also determined. In this instance, the target model was trained using four noise levels, i.e., 3%, 6%, 8%, and 10%, in addition to the original dataset.

In contrast, RF training involves the specification of a number of decision trees, a maximum

Table 9.1: Description of the generated dataset with the fault classes.

Fault ID	Fault Type	Fault Duration (sec)	Training Samples	Testing Samples
H	Healthy	-	2240000	280000
F1	Gain	166–335	2240000	280000
F2	Stuck-at	179–328	2240000	280000
F3	Noise	174–340	2240000	280000
F4	Delay	170–318	2240000	280000
F5	Drift	176–333	2240000	280000
F1F2	Gain & Stuck-at	164–323	2240000	280000
F1F3	Gain & Noise	169–315	2240000	280000
F1F4	Gain & Delay	175–330	2240000	280000
F1F5	Gain & Drift	174–324	2240000	280000
F2F3	Stuck-at & Noise	176–334	2240000	280000
F2F4	Stuck-at & Delay	180–340	2240000	280000
F2F5	Stuck-at & Drift	171–338	2240000	280000
F3F4	Noise & Delay	166–325	2240000	280000
F3F5	Noise & Drift	177–338	2240000	280000
F4F5	Delay & Drift	173–342	2240000	280000

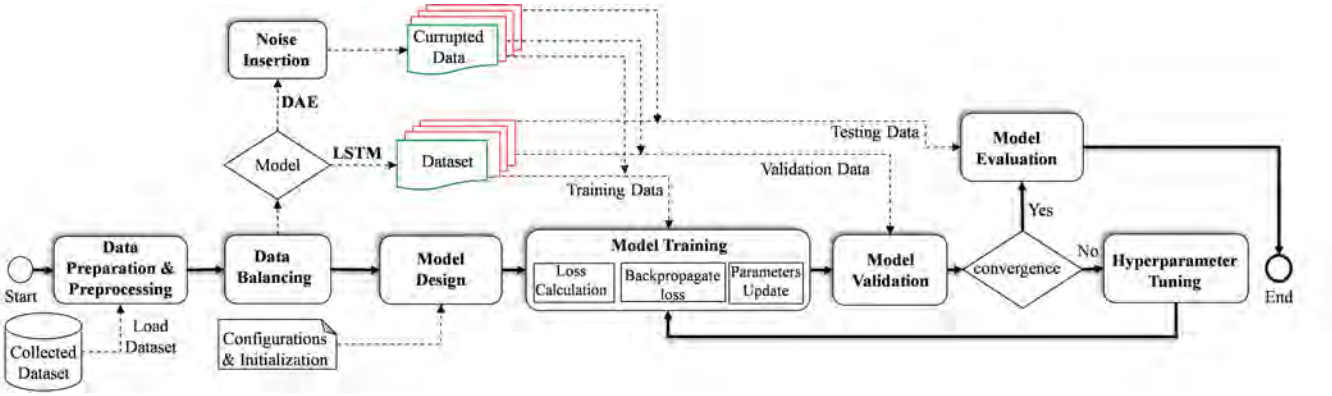


Figure 9.7: Flowchart of model training and optimization.

depth for each decision tree, and a randomizer. Furthermore, the out-of-bag (OOB) faults estimation feature and the warm start feature are enabled. Upon the commencement of the training process, the loss function is calculated and the performance of the model is subsequently monitored. The internal model parameters are updated in accordance with the propagation of the losses.

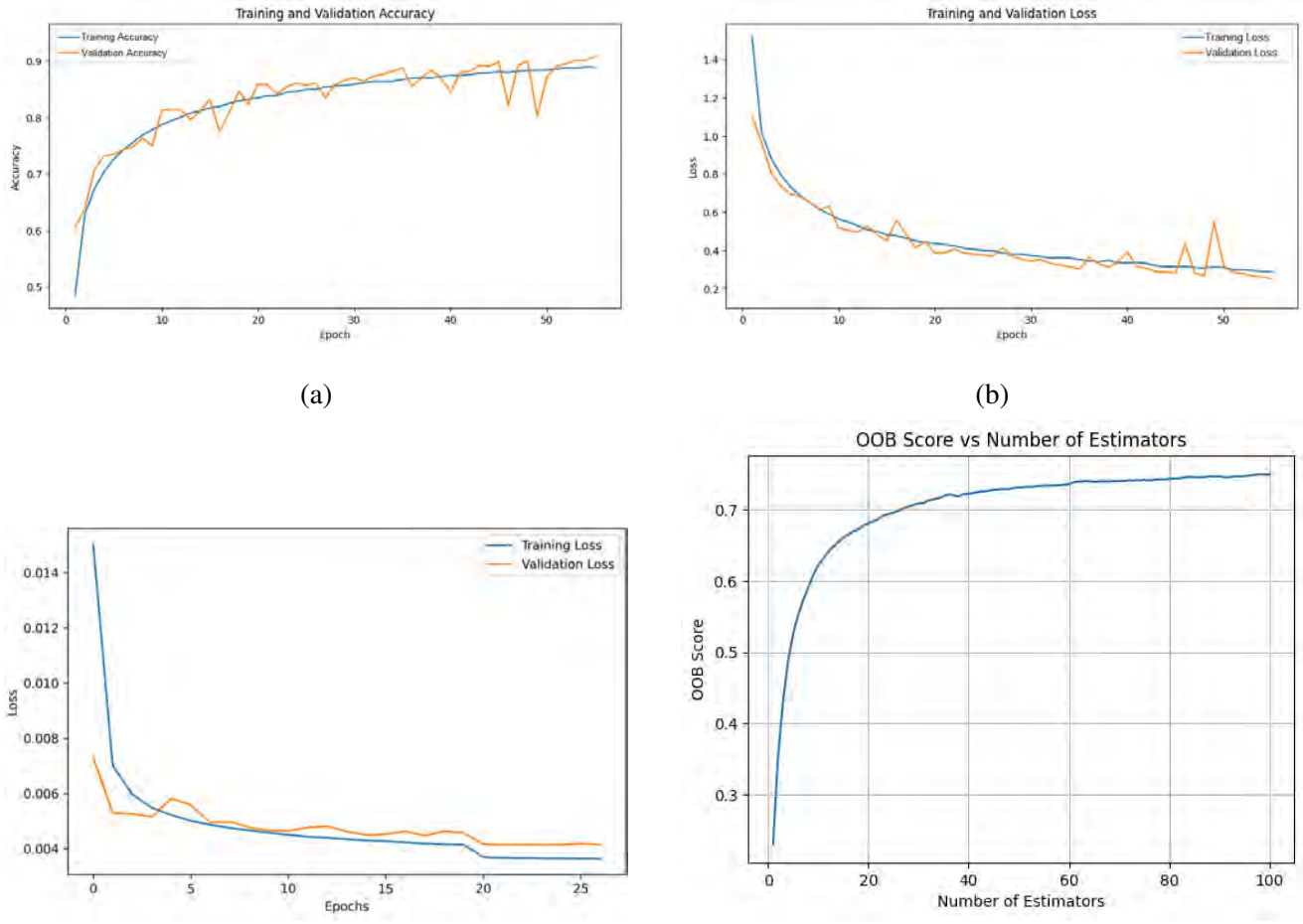
Table 9.2: Optimal hyperparameters of LSTM architecture.

Hyperparameter	**Optimal Values**
Hidden Layer Size	512
Epochs	60
Batch Size	1024
LSTM Layers	3
Learning Rate	0.0005
Activation Function	tanh
Optimizer	adam

Figure 9.8 illustrates the training process curve for the DAE, LSTM, and RF. It is of note that the performance of the trained model is significantly influenced by the defined hyperparameters. Consequently, the model optimization process is employed to optimize the trained model by adjusting the hyperparameters. To this end, the performance of the model is evaluated using validation data to ascertain whether convergence has been achieved. In particular, certain hyperparameters are adjusted in a manner that enhances the accuracy of the model in accordance with the optimized architecture. In this instance, a grid search mechanism has been employed to facilitate the exploration of diverse parameter configurations. The fundamental concept underlying this technique is the construction of a grid of potential hyperparameter values, encompassing all possible combinations. Consequently, the efficacy of the model is evaluated for each configuration through a cross-validation procedure, enabling the identification of optimal parameters that yield the highest performance. Nevertheless, given the intricate nature of the model, it is essential to contemplate the balance between the time and resources required for training and the potential for generalization to be achieved during the implementation phase. The selected optimal hyperparameters of the proposed LSTM are presented in Table 9.2.

9.4 Results and Discussion

The results of the experimental analysis of the proposed model are presented and discussed in this section. In particular, the efficacy of the proposed FDD model in terms of accuracy is demonstrated on the basis of a test dataset. Moreover, the superiority of the proposed architecture over standalone



(a) (b)

(c) (d)

Figure 9.8: Hyperparameter optimization results. (**a**) Training and validation accuracy of LSTM. (**b**) Training and validation Loss of LSTM. (**c**) Training and validation Loss of GRU-DAE. (**d**) OBB Score of RF.

techniques is demonstrated. Three evaluation metrics, i.e., precision, recall, and F1-score, are employed in this study to assess the detection and identification performance of the proposed model, as detailed in [291]. In parallel, the mean-square-error (MSE) [239] is employed to assess the performance and effectiveness of the DAE based on GRU.

9.4.1 GRU-based DAE Performance

The proposed model was evaluated under different levels of Gaussian noise, specifically 3%, 6%, 8%, and 10%, in order to demonstrate its anti-noise capability. The efficacy of the proposed model in reconstructing the original data under varying noise levels can be discerned in Figure 9.9. In particular, at low levels of noise, the developed model exhibits a high degree of performance in terms of a low reconstruction error, with an MSE of 0.023. However, this gradually increases at higher levels of noise. This indicates that minimal information is lost during the noise reduction and reconstruction process. It is notable that even at the highest noise level of 10%, the denoising performance of the model remains satisfactory, with an MSE of 0.0618. A comparison of the proposed model with other DAE structures, such as ANN-based DAE, reveals that it has superior performance, as demonstrated by its lower MSE. Furthermore, the proposed model demonstrates a significantly reduced reconstruction error at MSE values of 0.0234, 0.0421, 0.0504, and 0.0618, across noise levels of 3%, 6%, 8%, and 10%, respectively. The validation results demonstrate the efficacy of the DAE structure in reconstructing the original data without loss and in denoising the data with high performance. Nevertheless, the discrepancy between the reconstructed and denoised original data increases as the level of added noise increases.

9.4.2 Detection and Classification Results using Ensemble LSTM-RF

The efficacy of advanced ensemble classifiers with optimized architectures was assessed using a test dataset in terms of precision, recall, and F1-score. Figure 9.10 illustrates the model's fault identification performance for different fault classes, i.e., single and concurrent faults.

In the case of single faults, i.e., gain, stuck-at, and noise, the identification performance of the proposed model is evidently high, exceeding 97% in all evaluation measures. However, in case of the delay and drift faults, the performance of the classifiers is reduced to an F1-score of 92.57% and

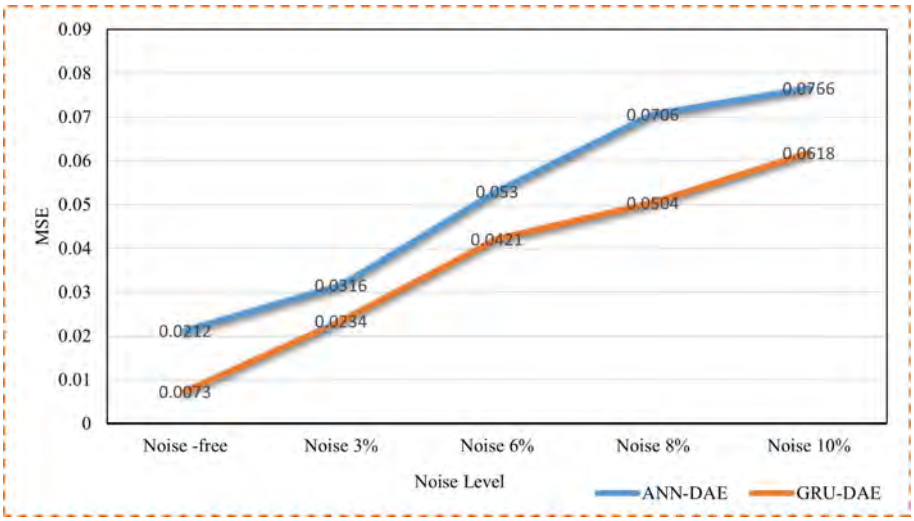
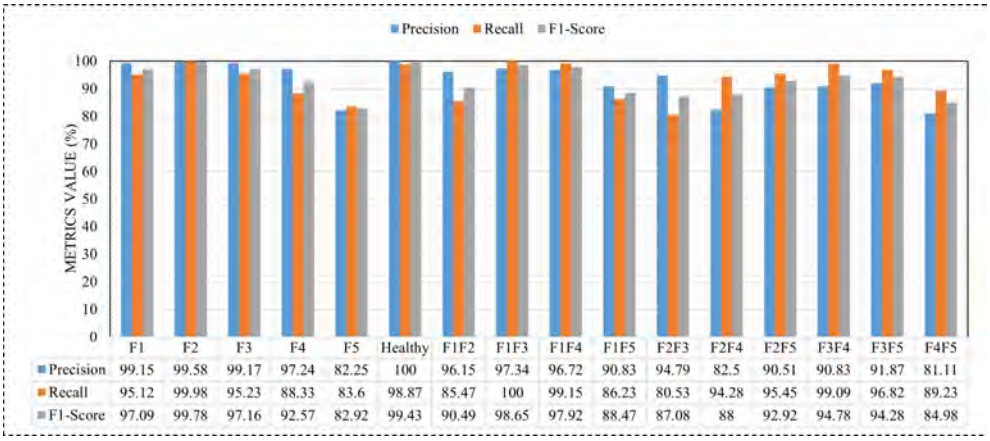


Figure 9.9: GRU-DAE performance under various level of noise.

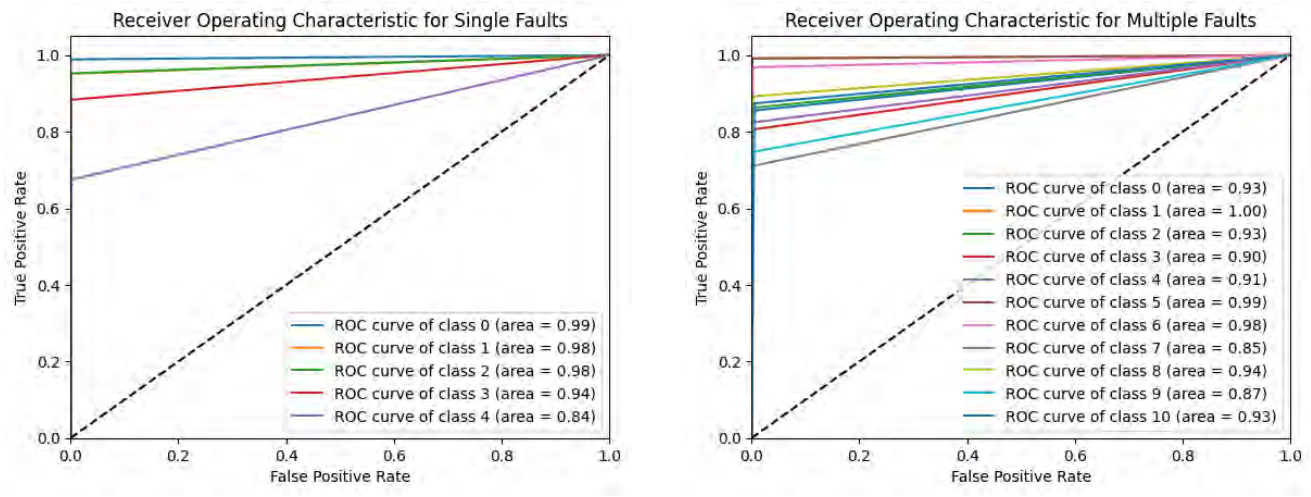


	F1	F2	F3	F4	F5	Healthy	F1F2	F1F3	F1F4	F1F5	F2F3	F2F4	F2F5	F3F4	F3F5	F4F5
Precision	99.15	99.58	99.17	97.24	82.25	100	96.15	97.34	96.72	90.83	94.79	82.5	90.51	90.83	91.87	81.11
Recall	95.12	99.98	95.23	88.33	83.6	98.87	85.47	100	99.15	86.23	80.53	94.28	95.45	99.09	96.82	89.23
F1-Score	97.09	99.78	97.16	92.57	82.92	99.43	90.49	98.65	97.92	88.47	87.08	88	92.92	94.78	94.28	84.98

Figure 9.10: Testing results of the proposed model for single and concurrent faults.

82.92%, respectively, due to the complexity of the pattern and the corresponding characteristics. The low sensitivity of the aforementioned types, with recall values of 88.33% and 83.33%, is attributed to the high rate of false positives. Notwithstanding, the system's performance in detecting faults states is noteworthy, with a harmonic mean F1-score of 99.43

The classification results of the simultaneous faults demonstrate that the ensemble learning approach is highly effective, with a mean accuracy exceeding 92%. The highest accuracy, as indicated by F1-scores of 98.65% and 97.92%, respectively, was obtained for the identification of the gain-noise and gain-delay classes. Conversely, the classifier's performance exhibited a slight decline, with F1-score of approximately 90% for the remaining classes. The lowest accuracy value



(a) (b)

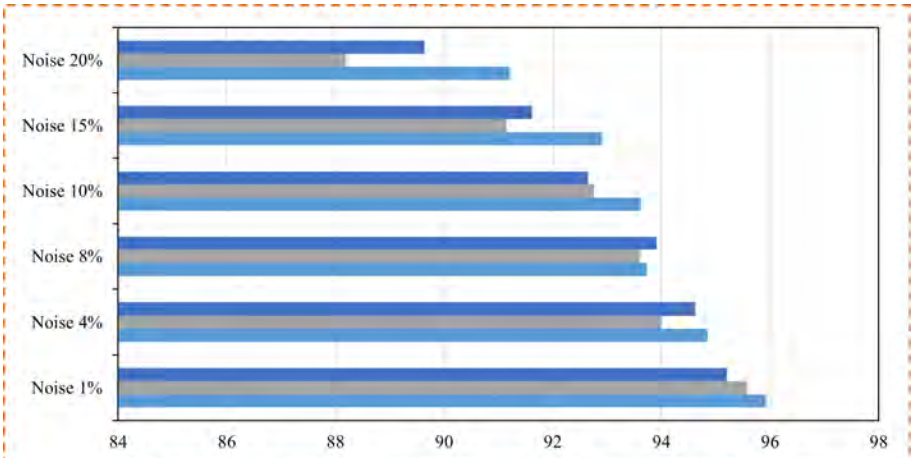
Figure 9.11: Fault identification performance in terms of AUC-ROC curve. (**a**) AUC-ROC curve of the proposed FDD model for single faults. (**b**) AUC-ROC curve of the proposed FDD model for concurrents faults.

observed for the F4F5 classifier is attributed to the misclassification of healthy samples as faults behavior, which is referred to as a false alarm. In conclusion, the model has been proven effective in detecting and identifying composite fault behaviors. The model's performance remains satisfactory even in the case of low values of the evaluation metrics, within an acceptable range.

The ROC curve is employed to graphically represent the performance of FDD with respect to each fault type. This enables the relationship between the true positive rate (TPR) and the false positive rate (FPR) to be plotted. Figures 9.11a and Figure 9.11b illustrate the extent to which the model is able to discriminate between types of faults, both individually and simultaneously. For instance, the ROC curve of class 0 (gain fault) with an ROC of 0.99 indicates that the model is capable of correctly identifying the gain fault with a probability of 99%. Conversely, the probability of the model correctly identifying simultaneous faults, such as F1F2, is 93%. The superior detection performance can be attributed to the tendency of the ROC value to the upper left corner.

9.4.3 Model Robustness against Noise

In order to assess the efficacy of the proposed model in the context of FDD, test data with varying levels of noise are employed. Thanks to the proposed GRU-DAE as the primary step before the FDD phase, the classifier demonstrated remarkable performance under noisy conditions. The high accuracy of the proposed FDD model in terms of precision, recall, and F1 value is evident from the evaluation results in Figure 9.12. In particular, under the first two noise levels, i.e., 1% and 4%, the model shows high accuracy with more than 94% F1-score. The robustness of the method is achieved by applying the DAE to reconstruct and denoise the original data, with a very low mean squared error (MSE), thereby eliminating the effects of any potential errors. However, the model's performance is negatively affected when the noise level is increased. At a noise level of 15%, the values of the evaluation parameters decrease to approximately 91%. Nevertheless, the model's capacity to address the challenge of noise remains satisfactory at high levels, specifically at 20% noise, with an accuracy of 89% for the F1-score. Consequently, it can be noted that the resilience of the proposed FDD model to noise is enhanced by the denoising process with GRU-DAE.



	Noise 1%	Noise 4%	Noise 8%	Noise 10%	Noise 15%	Noise 20%
F1-Score	95.21	94.62	93.91	92.65	91.62	89.64
Recall	95.58	94	93.61	92.76	91.15	88.19
Precision	95.92	94.85	93.73	93.62	92.91	91.21

Figure 9.12: Fault identification performance of the proposed model under various levels of noise.

Finally, the applicability of the proposed model was evaluated in two automotive cases, i.e., one involving an gasoline engine and vehicle dynamics. The two case studies demonstrate the high performance of the model with minimal reconstruction error. In addition to the dataset obtained

from automated driving, the test dataset obtained from manual virtual testing has been employed for the purpose of evaluating the proposed model. The results of the evaluation demonstrated that the proposed model is capable of denoising and reconstructing data with MSE values of 0.0421 and 0.0955, respectively, based on dataset 1 and dataset 2.

9.4.4 Classification Results Compared to Stand-alone Algorithms

The superiority of the proposed ensemble learning approach is evident from the comparison of the classification results achieved by the target model with those of individual techniques. The performance of the LSTM-RF, LSTM, and RF models for each fault class, including single and simultaneous fault types, is evaluated in terms of precision, recall, and F1-score. The results are presented in Tables 9.3, 9.4, and 9.5, respectively. The ensemble models demonstrated superior performance compared to the individual models, with an average precision of 93.57%. This is evident in the comparison results presented in Table 9.3. The mean accuracy for LSTM and RF is 92.3% and 70.16%, respectively. It is clear that the number of false positives in the proposed method decreased significantly when the results of the LSTM and RF algorithms were considered in the decision-making process. Nevertheless, in certain classes, such as delay fault, LSTM exhibits superior class identification performance, which, in turn, enhances F3F4 class identification performance. The normalized confusion matrix is depicted in Figure 9.13. A confusion matrix is presented herein to illustrate the identification performance achieved by the proposed model based on the test dataset. It is evident that a subset of the delay fault data samples have been erroneously identified as both a stuck-at fault and a delay fault simultaneously. A similarly poor prediction performance was obtained for the F2F3 class. This phenomenon can be attributed to the similarity in signal characteristics exhibited by the aforementioned fault classes. Furthermore, a few samples of class F1F2 were misclassified by the model as class F3F4. In contrast, the class F1-F3, gain and noise faults, exhibited the highest identification accuracy.

Similarly, the capacity of the proposed model to accurately identify all faults features is superior to that of the individual techniques. As illustrated in Table 9.4, the recall value of the ensemble learning approach exceeds 95% in the majority of the classes. Moreover, the proposed model demonstrates superior performance compared to LSTM and RF, even for complex fault patterns such as F4, F1F4, and F2F3. As illustrated in Table 9.5, the harmonic mean of recall and preci-

Table 9.3: Precision Score of the proposed FDD model compared to individual methods (%).

ID	Type	LSTM	RF	LSTM-RF
H	Healthy	99.99%	99.88%	100%
F1	Gain	96.94%	45.8%	99.15%
F2	Stuck-at	97.78%	85.98%	99.17%
F3	Noise	89.95%	53.65%	97.24%
F4	Delay	84.97%	70.24%	82.25%
F1F2	Gain-Stuck-at	94.58%	73.42%	96.15%
F1F3	Gain-Noise	97.98%	89.47%	97.34%
F1F4	Gain-Delay	88.65%	54.16%	90.83%
F2F3	Stuck-at-Noise	90.66%	53.09%	94.79%
F2F4	Stuck-at-Delay	90.65%	67.60%	90.51%
F3F4	Noise-Delay	83.95%	78.51%	91.87%

Table 9.4: Recall Score of the proposed FDD model compared to individual methods (%).

ID	Type	LSTM	RF	LSTM-RF
H	Healthy	98.77%	96.05%	98.87%
F1	Gain	92.33%	61.85%	95.12%
F2	Stuck-at	95.67%	78.63%	95.23%
F3	Noise	89.62%	71.54%	88.33%
F4	Delay	70.22%	65.89%	83.6%
F1F2	Gain-Stuck-at	86.38%	86.06%	85.47%
F1F3	Gain-Noise	98.64%	91.07%	100%
F1F4	Gain-Delay	84.37%	55.32%	86.23%
F2F3	Stuck-at-Noise	79.83%	46.87%	80.53%
F2F4	Stuck-at-Delay	96.24%	79.33%	95.45%
F3F4	Noise-Delay	97.92%	77.86%	96.82%

Table 9.5: F1-Score of the proposed FDD model compared to individual methods (%).

ID	Type	LSTM	RF	LSTM-RF
H	Healthy	99.38%	97.93%	99.43%
F1	Gain	94.58%	52.63%	97.09%
F2	Stuck-at	96.71%	82.14%	97.16%
F3	Noise	89.78%	61.32%	92.57%
F4	Delay	76.90%	68%	82.92%
F1F2	Gain-Stuck-at	90.30%	79.24%	90.49%
F1F3	Gain-Noise	98.31%	90.26%	98.65%
F1F4	Gain-Delay	86.46%	54.85%	88.47%
F2F3	Stuck-at-Noise	84.90%	49.79%	87.08%
F2F4	Stuck-at-Delay	93.36%	73.00%	92.92%
F3F4	Noise-Delay	90.40%	78.18%	94.28%

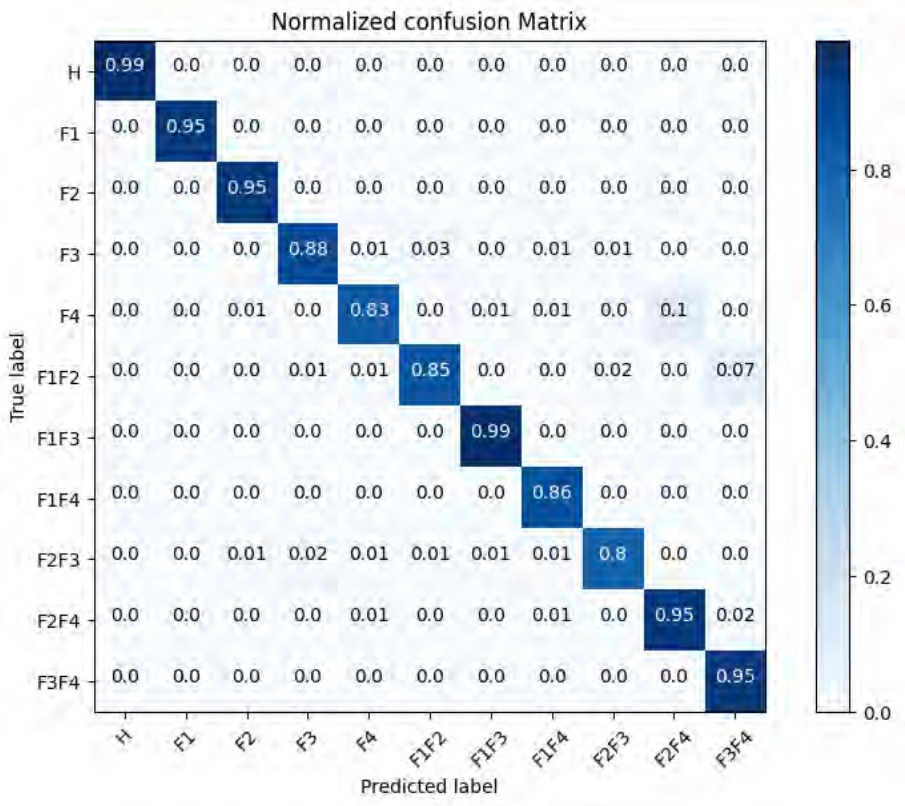


Figure 9.13: Confusion matrix of the proposed model with normalization.

sion demonstrates that the majority of fault classes are accurately detected and identified by the proposed method. The proposed model has demonstrated high performance for F1F2, F1F3, and

Table 9.6: Comprehensive analysis of FDD performance with different classifiers in terms of F1-Score (%).

Model	**Single Fault**	**Simultaneous Faults**
SVM	81.17%	77.93%
DT	79.48%	73.84%
MLP	90.66%	88.5%
1D-CNN	92.7%	87.44%
LSTM-RF	94.82%	91.2%

Table 9.7: Comparison between the results of the proposed method and other related works.

Reference	**Method**	**Accuracy**
[255]	PCA-multiclass SVM	84.93%
[244]	PCM	81.49%
[214]	DNN and 1D-CNNs	89%
[321]	MLBCL-EE	87.57%
[322]	HML-KNN	85.07%
[247]	SSAE	87.61%
Proposed work	DAE, LSTM-RF	91.2 %

F2F4, with F1-scores of 90.49%, 98.65%, and 92.92%, respectively. Nevertheless, the identification of faults states of the system in the case of delay faults remains to be improved. The reason for this shortcoming is the similarity between the healthy and faults features used to develop the model. In conclusion, the performance of each method-based classifier is significantly worse than that of the proposed model when evaluated on the same training and test data. This is due to the traditional method's inability to capture the complex relationship between fault classes, especially when faults occur simultaneously. Conversely, the proposed model is predicated upon a voting algorithm which incorporates the outcomes of the aforementioned models. In addition, several other ML-based classifiers were implemented and evaluated on the test dataset. Four classifiers were selected for this purpose, i.e., SVM, DT, MLP, and 1D-CNN. The performance of the proposed model in comparison to conventional classifiers in terms of F1-score is presented in Table 9.6, which presents the mean faults classification accuracy for single and concurrent faults. The aforementioned methods

were subjected to evaluation on a test dataset comprising both healthy and faults data samples. The ensemble LSTM-RF classifier exhibits superior performance compared to other traditional classifiers, despite the excellent performance of DL architecture-based classifiers, such as 1D-CNN and MLP. Table 9.7 compares the proposed model's performance in terms of accuracy with those obtained in other related works. It can be concluded that the proposed procedure is capable of enhancing the efficacy of concurrent FDDs in comparison to alternative procedures. Furthermore, the model's resilience to noise enables its application to FDD problems in diverse systems across different domains.

9.4.5 Computational Complexity Analysis

One of the primary considerations when developing DL models is the associated computational cost, particularly when dealing with voluminous datasets. Consequently, the requisite training and inferencing time of the target models is evaluated in consideration of the specifications of the training platform, i.e., Google Colab, as illustrated in the Table 9.8. The average training time for the development of a GRU-based DAE is 4445 seconds. Additionally, the requisite time for the testing procedure involving data reconstruction and denoising is notably low, with an average runtime of 0.339 seconds. In contrast, the training time for developing the proposed ensemble models was 5,319.48 seconds. The testing process requires 4.85 seconds to evaluate 10% of the total dataset. It is evident that the ensemble decision process necessitates a significant amount of testing time. The time required for training and testing increases with the number of classifiers employed in the ensemble learning process. Nevertheless, the computational time is tolerable in comparison to the traditional ML method, which necessitates additional time for manual feature extraction. This is due to the fact that the aforementioned features are extracted automatically. Nevertheless, the advent of more sophisticated computing resources can effectively address this shortcoming, thereby achieving a more balanced compromise between high performance and development time. Moreover, the potential for further enhancements with reduced computational demands is facilitated by the consideration of alternative, simplified DL architectures, such as GRU-based classifiers. In particular, it is possible to enhance the proposed methodology to make the identification of composite faults contingent upon the data of individual faults.

Table 9.8: Training and testing times of the proposed FDD model.

Model	Training Time (s)	Testing Time (s)
GRU-DAE	4445.17	0.339
LSTM-RF	5319.48	4.85

9.5 Conclusions

To overcome the challenge of detecting and classifying concurrent faults in the real-time validation of ASSs during system integration and testing, this chapter proposes a solution based on ensemble learning. In particular, two models were developed, i.e., LSTM model and an RF model. In contrast to conventional methods that focus on single faults, this study aims to develop a system-level approach for identifying both single and concurrent sensor faults in the presence of noisy and unbalanced data. In order to ensure the reliability and noise immunity of the proposed model, the present study employs a GRU-based DAE. Moreover, a random undersampling algorithm is utilized to address the difficulty of developing FDD models in the context of imbalanced data. To this end, a real-time FI method with HIL simulation is used to analyze the most crucial faults and to collect a representative dataset. In order to ascertain the efficacy and suitability of the proposed approach, a sophisticated, high-fidelity model of a gasoline engine system is employed as a case study. This encompasses the entirety of the vehicle's dynamic behavior within the operational context. A quantitative evaluation of the classification of individual faults yielded a F1-score of 94.82%, a recall of 93.52%, and a precision of 96.23%, according to an average value of the evaluation metrics. Another notable finding was the model's capacity to correctly identify concurrent faults, with a F1-score of 91.2%, a recall of 92.62%, and a precision of 90.26%. The results of the analysis, conducted using the same dataset, reveal a notable improvement in the FDD accuracy for single and concurrent faults when employing ensemble models in comparison to traditional single classifiers. The GRU-based DAE is capable of reconstructing and denoising the original data with an MSE of less than 0.05 at a noise level of 8%. In consequence, the effectiveness of the proposed model is enhanced even in the presence of a high noise level. The model exhibited excellent performance, with a F1-score of 92.65% at a noise level of 10%. Moreover, the model exhibited an accuracy rate of 99.43%, thereby demonstrating a high detection capability. In conclusion, the

combination of DAE with an ensemble predictive model offers a dependable and resilient FDD for the real-time ASSs validation process during HIL testing when compared to individual methods. Moreover, the integration of this approach results in enhanced safety and reliability of the target system. Furthermore, it reduces the time and effort required during the development process.

10 Virtual Test Drive Framework for Real-Time Validation of Automotive Software Systems

The primary objective of this chapter is to develop a real-time validation framework for conducting virtual test drives with the HIL system. This, in turn, contributes to answering research question RQ5. This is accomplished by examining the viability of utilizing real steering systems and pedals along with HIL simulation and FI method. In addition to the FI framework developed in Chapter 6, a variety of scenarios pertaining to road network topology and environmental conditions are considered. In this manner, the developed approach serves to generate the test dataset for the validation of the proposed models in Chapters 7, 8, and 9. This chapter is organized as follows: the chapter starts with an overview in Section 10.1. The aim of Section 10.2 is to introduce the layers and components of the proposed framework. The main phases of the implementation are presented in Section 10.3, including the development phases. The applicability and benefits of the proposed framework are demonstrated in Section 10.4 by analyzing and discussing the evaluation results. The chapter ends with a conclusion in Section 10.5.

Note: It is important to highlight that the proposed methodology and the corresponding implementation outcomes presented in this chapter have been published in [323].

10.1 Chapter Overview

The assessment of the safety and reliability of heterogeneous systems represents a significant challenge for the industry, due to the increasing complexity, e.g., ASSs. In accordance with the ISO 26262 standard, safety-critical real-time systems should undergo rigorous evaluation at each stage of the development process. Currently, the validation of ASSs pertaining to safety is conducted

via a number of experimental tests conducted at various stages within the V-model. These tests are referred to as 'X-in-the-loop' (XIL) methods. Nevertheless, these methods are associated with a number of significant drawbacks, in terms of cost, time, effort, and effectiveness. Consequently, the development of a testing approach that can bridge the aforementioned gap between the validation methods is required. The objective of this chapter is the development of a HIL-based real-time virtual test drive framework for the validation of system performance under normal and abnormal conditions.

The framework provides an approach for the real-time analysis of system behavior in the event of single and simultaneous sensor/actuator-related faults during virtual test drives. In contrast to traditional testing platforms, faults are injected in a programmed fashion, and the system's architectural constraints are met to ensure real-time functionality in such a way that no modifications are necessary. Besides, the framework incorporates a virtual environment with varying conditions, including weather, traffic, and roads.

In order to validate the efficacy of the proposed approach, a dynamic vehicle system with traffic has been used as a case study. The steering system has been the focus of the analysis process, with the objective of demonstrating the effect of single and simultaneous faults on the safety system behavior. A comparative analysis of the proposed methodology with existing approaches has demonstrated that it exhibits enhanced accuracy and performance. The developed framework is designed to provide a safe, dependable and realistic foundation for real-time assessment of complex vehicle systems at a low cost and with a minimal time and effort.

10.2 Proposed Methodology

This section provides an in-depth explanation of the key elements of the proposed methodology, encompassing the hardware, software components, and communication between them. It is necessary that the development process for automotive safety-critical systems comply with the relevant functional safety standard, i.e., ISO 26262. Furthermore, the proposed study considered a number of aspects of testing in accordance with ISO 26262. This comprises the definition of testing strategies, the assessment of the suitability of the test environment, and the validation of the overall suitability of the tool chain. The proposed approach to testing follows a predefined strategy, which

includes the delineation and definition of various components of the testing process, including the test focus, object, level, methodology, and environment. Given that both HIL simulation and FI are explicitly recommended by the ISO 26262 during the validation procedure, they were incorporated into this approach. In addition, the development process within the proposed approach made use of a software toolchain that has undergone comprehensive certification by TÜV SÜD, i.e., the dSPACE tools, in order to ensure compliance with all relevant standards.

The framework is comprised of three distinct layers, i.e., the hardware layer, the data analysis and FI layer, and finally, the system modeling and environment visualization layer. The layers of the proposed framework architecture are depicted in Figure 10.1.

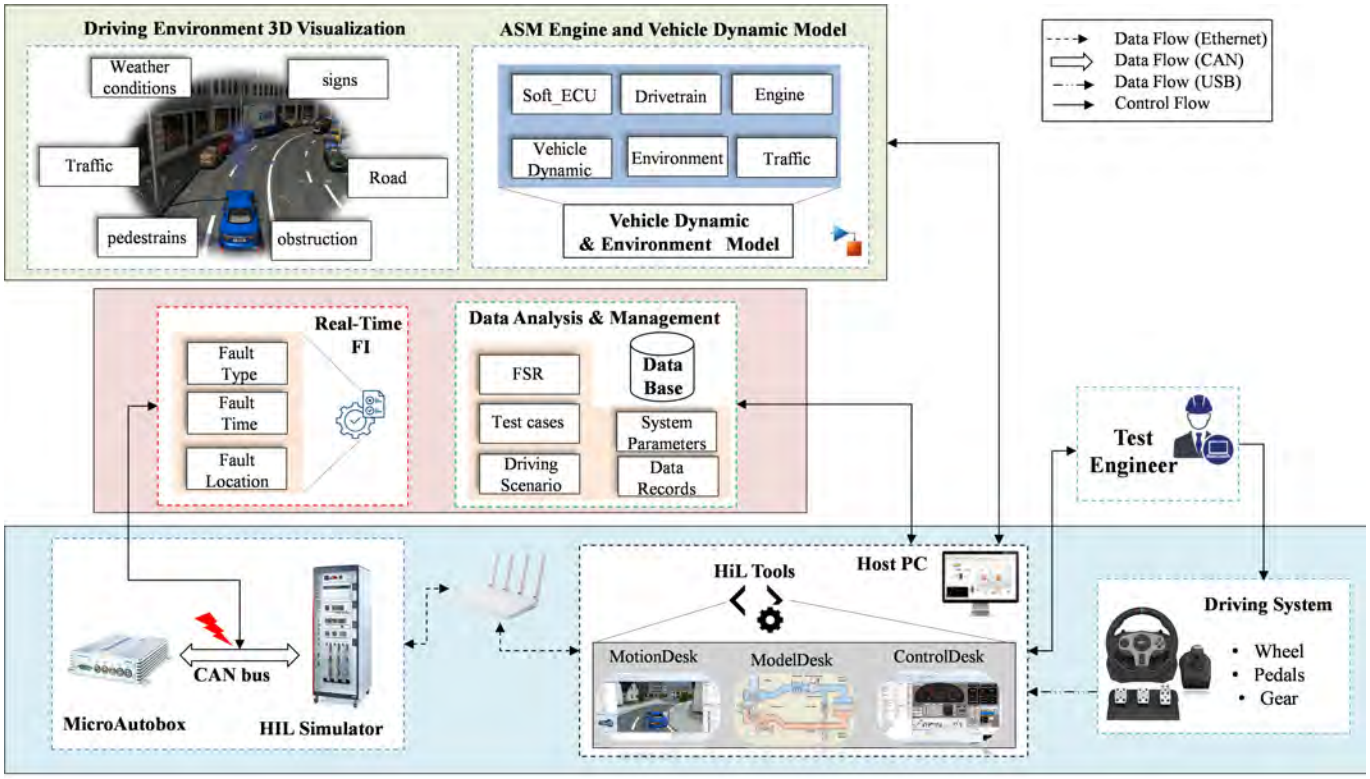


Figure 10.1: Proposed approach for developing real-time validation framework.

At the level of physical hardware components, the HIL simulation system, in which the HIL simulator is directly connected to the real ECU, is utilized for real-time simulation. In the course of this study, the entire vehicle model, which represents the controlled system, was executed in the HIL simulator. The control strategy was deployed and executed in the MicroAutoBox II, which

acted as the real ECU. The user is afforded the opportunity to operate the virtual vehicle manually via the steering wheel, gear shift, and pedals. It is noteworthy that the driving system in the proposed approach is capable of supporting both manual and automatic gear shifting during manual driving. The real-time test setup and control is performed using the dSPACE software tools, i.e., ModelDesk, MotionDesk, ControlDesk and AutomationDesk [167] on the host computer. The mentioned software tools are used to perform a plethora of tasks, including parameterization, configuration, instrumentation, execution, measurement, and analysis. The advantages of model-based design are exploited to automatically generate code from the models and implement it on the target device via the host computer. A total of three communication protocols are employed in the framework, i.e., Ethernet, CAN bus and USB, in order to interconnect the hardware components. The CAN bus, into which faults are injected, is employed to facilitate the transfer of information between the controller and the controlled system , i.e, between ECU and HIL simulator. The host PC establishes a connection to the HIL system via an Ethernet network, transferring the generated model code from the PC to the HIL elements. Finally, the USB communication protocol is employed for the transmission of driving commands from steering system to the HIL simulation controller.

In the second layer, the real-time FI framework developed in chapter 6 is employed to assess the resilience of the SUT and its functionality in the presence of faults during execution. To this end, three distinct FI attributes are identified, i.e., fault type, target location, and time of injection. In accordance with the specifications and test objectives, the fault injector furnishes a list of sensor and actuator faults, including those related to gain, offset, stuck, delay, data loss, hard-over, noise, drift, and spike. Further details on the various fault types can be found in [305]. The signals from the control unit and plant can be accessed during real-time simulation thanks to the capabilities of the CAN bus model. Consequently, the initiation of the FI at the target component can be accomplished programmatically as a black-box test, without necessitating modifications to the system model. Thus, the test engineer is able to select the timing and duration of the FI based on the standard system response, which is defined as the desired driving cycles.

In addition to the FI framework, the second layer is utilized for the analysis and management of data. This layer encompasses inputs and outcomes pertinent to targeted experiments. To put it another way, this layer is where functional and non-functional requirements are documented and

subjected to analysis. Moreover, it identifies fault test case specifications and functional test case specifications, including test data and the anticipated outcomes. In this context, the ModelDesk tool is employed for the purpose of designing the aforementioned TCs in accordance with the aforementioned test scenario. Upon completion of the tests, the recorded system performance is compared with that which was intended in accordance with the defined functional and non-functional requirements. This allows for the identification of any violations in the safety objective. In order to ascertain the system's response to faulty components, a further analysis is conducted to identify any deficiencies in the system design.

Finally, in the third layer, two phases are considered, i.e., the modelling of the system and the visualization of the environment. As previously stated, the ASM vehicle dynamics model, developed by dSAPCE [166], represents the target system from the automotive domain. A vehicle system model was created within the MATLAB/Simulink environment. Nevertheless, the system model only incorporates the fundamental engine model. This limitation precludes the ability to capture and analyze detailed characteristics of the engine in question. Consequently, a sophisticated model of the ASM gasoline engine has been incorporated into the dynamic model to address this issue. Moreover, in order to achieve comprehensive representation of system behavior, consideration was given to other subsystems of the vehicle, including the electrical system, the powertrain, the driver, and the environment. The code for all these models is generated and executed in the HIL simulator. Conversely, the control algorithm is modelled separately and connected to the control system through a CAN bus model in order that it may act as a signal interface model. It should be noted that the RTICANMM model is employed at this stage to configure the FI framework.

Moreover, the modelling and visualization of the driving environment in the form of a 3D visualization constitutes an additional element of this level. This is achieved through the utilization of the MotionDesk and ModelDesk tools. In addition to the typical conditions, the potential for irregularities in the road and in the surrounding environment have also been considered. Specifically, the road topology was modeled with surface irregularities in order to replicate the actual environment as closely as possible. Moreover, the dynamic objects on the road were incorporated into the model through the representation of other vehicles and pedestrians. Lastly, the ModelDesk tool is utilized to model traffic signs and obstacles. In order to accurately reflect the actual road topology, a Google map of the target area was utilized as the foundation for the developed environmental

model within the proposed framework. Moreover, in accordance with the standard dimensions of the real environment, the dimensions of the designed roads were calibrated.

10.3 Experimental Implementation and Setup

In addition to providing an overview of the framework's hardware components, this section also presents the case study software architecture. In addition, the most pivotal periods in the framework's development, including the communication between software and hardware, are delineated.

10.3.1 Case Study Architecture:

ASM Vehicle Dynamic model with traffic, provided by dSPACE, was employed to illustrate the applicability and advantages of the proposed framework. Figure 10.2 presents the architectural framework of the selected vehicle dynamics system. The target system was constructed and evaluated within the MATLAB/Simulink simulation environment. A comprehensive characteristics of the system's behavior can be captured by modeling the detailed aspects of the system and simulating it with a high degree of fidelity. The vehicle system was divided into subsystems, including the basic gasoline engine, vehicle dynamics, drivetrain, and environmental aspects. However, due to the limitations of the basic engine model employed in the case study, it was replaced by the ASM gasoline engine. In a similar manner, the gasoline engine system has been designed on the basis of multiple partial systems, i.e., the air path system, the fuel system, the piston engine system, the exhaust system and the cooling system. In order to consider the entire vehicle in real-time simulation, the aforementioned systems have be expanded to incorporate the powertrain, driver, and environmental electrical systems. For each system within the vehicle, there exists an ECU model that is responsible for regulating the corresponding system. Consequently, during the development phase, there is considerable leeway in selecting the SUT for each target system, based on the applications in question. It worth noting that, in this study, the ECU of the gasoline engine is selected as the SUT.

Mathematically, the aerodynamic forces and moments can be represented as follows:

$$F_{aero,i} = -\frac{1}{2} D_{air} . S^2 . A_i(T) . Z_i \tag{10.1}$$

$$F_{aero,j} = -\frac{1}{2} D_{air}.S^2.A_j(T).Z_i \tag{10.2}$$

$$F_{aero,k} = -\frac{1}{2} D_{air}.S^2.A_k(T).Z_i \tag{10.3}$$

$$Tr_{aero,i} = -\frac{1}{2} D_{air}.S^2.A_{Mi}(T).Z_i.Q_{ch} \tag{10.4}$$

$$Tr_{aero,j} = -\frac{1}{2} D_{air}.S^2.A_{Mj}(T).Z_i.Q_{ch} \tag{10.5}$$

$$Tr_{aero,k} = -\frac{1}{2} D_{air}.S^2.A_{Mk}(T).Z_i.Q_{ch} \tag{10.6}$$

$$S = (S_{vehicle} - S_{wind})ij \tag{10.7}$$

where $S_{vehicle}$ and S_{wind} are the vehicle and wind speeds [m/s], respectively. D_{air} is the air density [kg/m3]. The vehicle's longitudinal shadow area [m2] is represented by Z_i. While D_{air} is the characteristic length for torque calculation [m]. $A(T)$ represents the aerodynamic coefficients. Finlay, T is the angle of incidence [deg].

Equation (10.8) represents the wheel speeds V_{wheel}, where, Tr_{shaft}, Tr_{tire} and Tr_{brake} represent the driving torque, effective tire torque and effective brake torque, respectively. I_{wheel} is the moment of inertia about the wheel axis.

$$I_{wheel}.V_{wheel} = Tr_{shaft} + Tr_{tire} + Tr_{brake} \tag{10.8}$$

The motion for the movement of the steering rod can be represented mathematically as Equation (10.9):

$$N_{st} q_{st} = F_{st,fl} + F_{st,fr} + F_{st,gear} + F_{st,fric} \tag{10.9}$$

The generalized force due to the front left tire forces, the generalized force due to the front right tire forces, the generalized force on the steering gear from the steering column, and the generalized force due to friction in the steering rod are represented by the symbols $F_{st,fl}$, $F_{st,fr}$, $F_{st,gear}$, and $F_{st,fric}$, respectively. The total mass along the axis of q_{st} is determined by the wheel inertia and masses, which is denoted by N_{st}. The system parameters, including the specifications of the model employed for real-time simulation in this study, are summarized in Table 10.1.

The system architecture model for our case study is presented in Figure 10.2, which incorporates both the ECU and plant models. It is of note that this architecture permits the existence of

Table 10.1: system parameters of the used case study for real-time simulation.

Parameter	**Unit**	**Value**
Mass vehicle with wheel	kg	1880
Density Air	kg /m^3	1.188
Mass front wheel	kg	35
Mass left wheel	kg	32
Wheel base	m	2.9742
Inertia about the wheel ration	kg*m^2	1.5
Engine speed setpoint during start up	rpm	1100
Engine Displacement	cc	2900
Engine power	kW	185
Car type	-	MidSizeCar
Transmission type	-	Stepped Automatic
Tire model	-	Magic Formula

two distinct operational modes, i.e., online and offline modes. In particular, in offline mode, that is, with SoftECU, both the SUT model and the controlled model can be executed in the HIL simulator. Consequently, the online mode, where the control system is executed in a separate target machine, the control system is executed in the HIL simulator in the aforementioned mode. In the online mode, the CAN bus is employed to facilitate communication between the SUT and the plant. The signal interface is represented in a model and simulated in accordance with the Real-Time Interface CAN Multimessage Block Set (RTICANMM). The ECU and vehicle model code are generated and deployed on the target machine for real-time simulation. This study employs rapid control prototyping (RCP), i.e., a MicroAutoBox II embedded PC (DS1401 base board). This comprises a 900 MHz Intel® Core™ i7-6822EQ processor, 16MB memory, and a 340 ms boot-time for a 3MB application. The mentioned RCP is used to simulate the functionality of a real ECU, while the SCAELXIO DS6001 processor board serves as the hardware for real-time data processing (Figure 10.3).

This proposed approach is evaluated in terms of its runtime cost and complexity. In particular, the time and resources required for the creation and setup of the approach are considered. The greatest duration of any individual task, as indicated by the data obtained from the debugging run, is 0.738 milliseconds. This value was determined based on a 10-second protocol and a scheduled

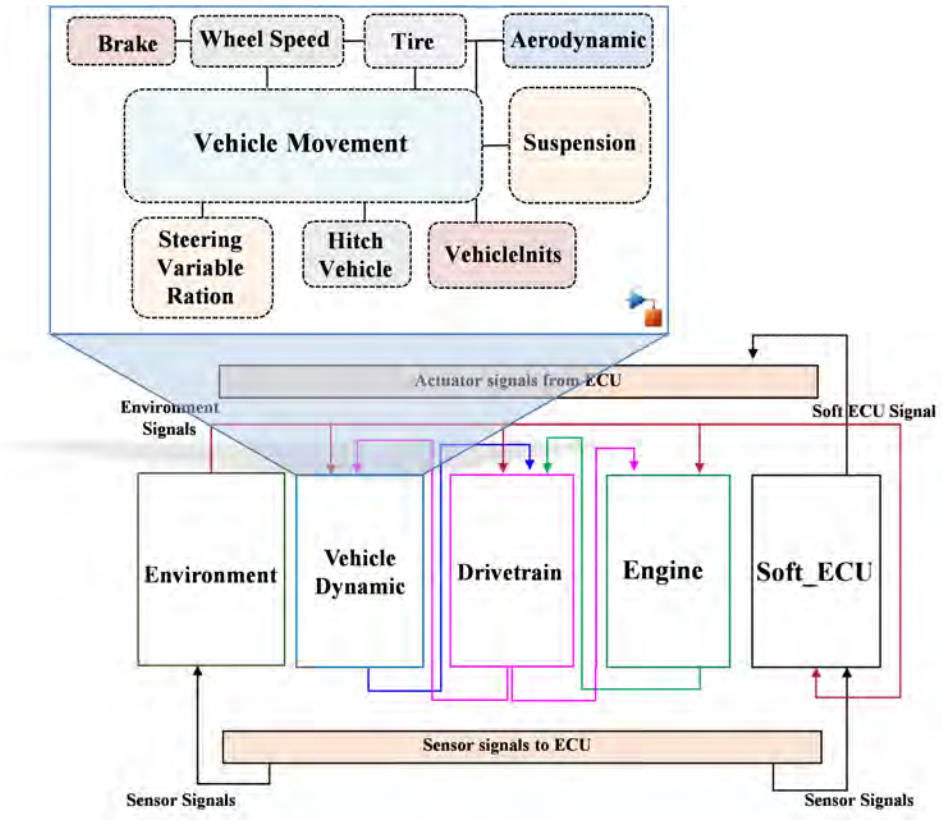


Figure 10.2: System architecture of the used case study.

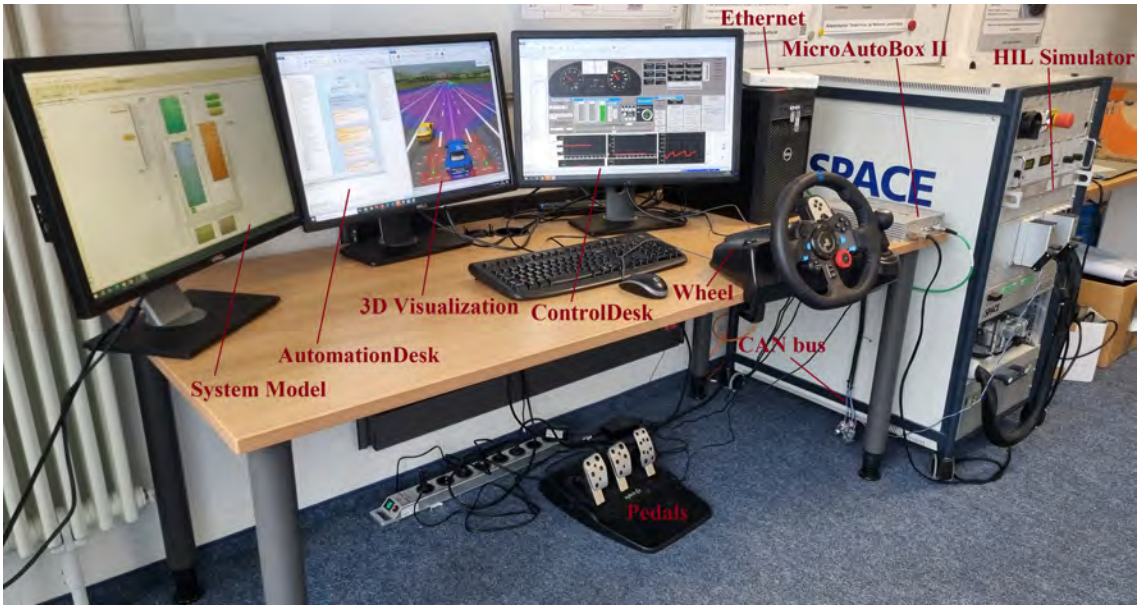


Figure 10.3: HIL real-time simulation-based virtual test drive.

sampling time of 1 ms. Conversely, the execution time for each test case is 402.426164 ms, inclusive of the evaluation phase. Nevertheless, the necessity to specify three distinct FI attributes during the setup phase implies that an infinite number of configurations could be created, thereby creating an infinite fault space. Achieving a balance between comprehensive testing and the identification of critical faults remains a significant issue in the context of complex software systems. Furthermore, in addition to the setup effort, a real-time system should be provided to ensure the real-time execution of both the system model and the framework under investigation. In order to ascertain the most effective fault TCs, three distinct factors were taken into consideration. The first factor represents the impact and speed of the injected fault. The second factor is the representative set of realistic faults that are injected. Finally, the diversity of fault types and environmental conditions constituted the third factor.

10.3.2 Experimental Implementation:

This section presents a detailed account of the primary stages that comprise the proposed framework implementation process. The development of the proposed framework is divided into four phases, as illustrated in Figure 10.4. In particular, along with the principal stages of the real-time HIL validation process, the following activities are conducted, i.e., the development of the FI framework, the configuration of the driving system, the test design, and the development of the driving environment. The software tools utilized throughout the process include dSPACE's ModelDesk, MotionDesk, and ControlDesk. [167].

1. **Real-time automated fault injection framework:**

   In accordance with the ISO 26262 standard, the FI method is strongly recommended for use in the validation of safety and reliability properties at various stages of the system development process. Moreover, this methodology is widely acknowledged as an effective approach for assessing the resilience of fault-tolerant systems in the presence of faults. The fundamental principle underlying this approach is the injection of faults into the SUT and subsequent assessment of its response to the resulting anomalous conditions. For further information, refer to Chapter 6.

   It is crucial to emphasise that the real-time constraints of the system execution in the target

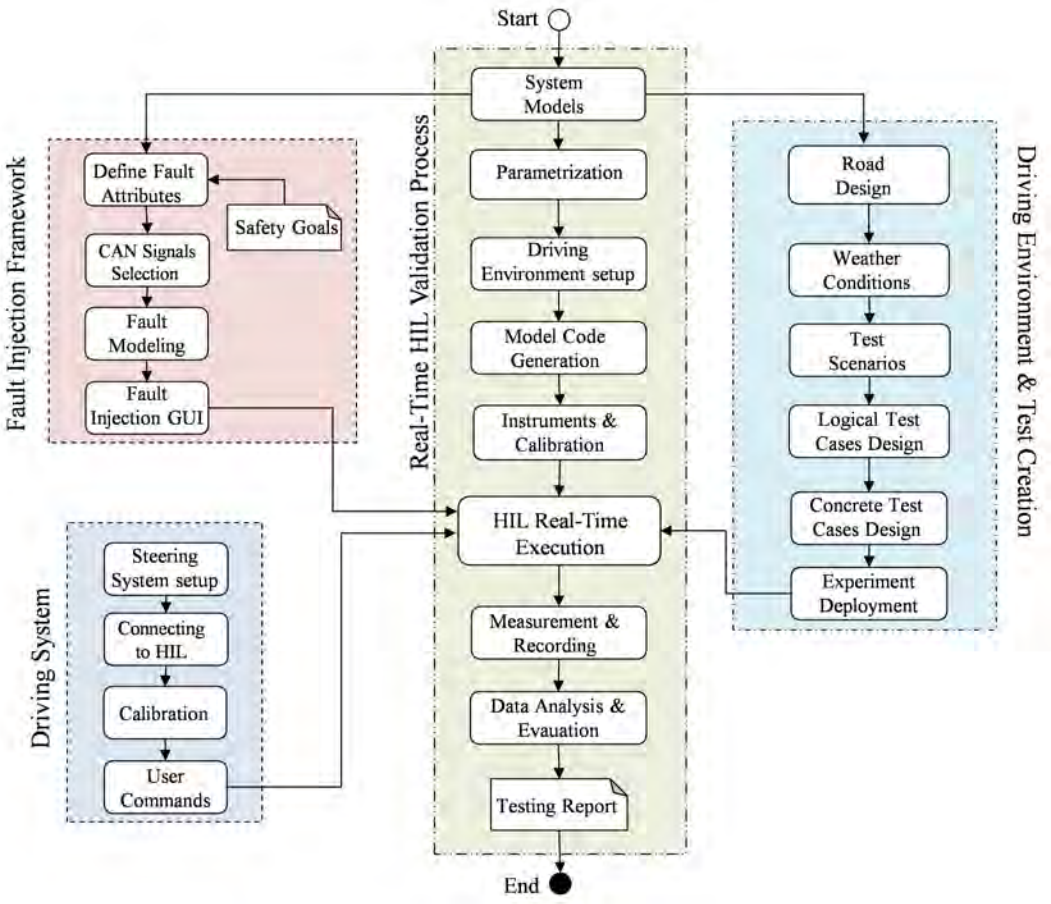


Figure 10.4: Flow chart of developing the proposed framework.

machine is guaranteed during the system integration test phase of the V-model, as outlined in ISO 26262-4. Consequently, the study performs the FI procedure at the signal interface between the SUT and the HIL simulation. The system then manipulates the selected fault locations based on the documented FI attributes, accessed and selected via the CAN bus network. Based on the signals accessed in the CAN bus network, the target system variables can be manipulated programmatically. This, consequently, ensures that the fault modes can be implemented in a black-box fashion, obviating the necessity to extend the system architecture model.

The graphical user interface for FI experiments allows users to modify and specify the configurations of FI experiments prior to their real-time execution , as shown in Figure 10.5. This is done according to the requirements and objectives of the test. Furthermore, the developed framework allows the injection of faults in either a single or a simultaneous manner.

Moreover, the framework permits the analysis of the impact of transient and permanent faults on system behavior in real-time. In conclusion, the framework offers two modes of execution, i.e., automated and manual, respectively. In the event that the machine is the entity responsible for the testing process, AutomationDesk enables the user to conduct FI test in an automated manner. In this instance, the execution process is comprised of a sequence of block hierarchies, comprising three phases, i.e., reading, signaling, and writing. Alternatively, faults may be introduced manually during real-time HIL execution via a user-based virtual test drive.

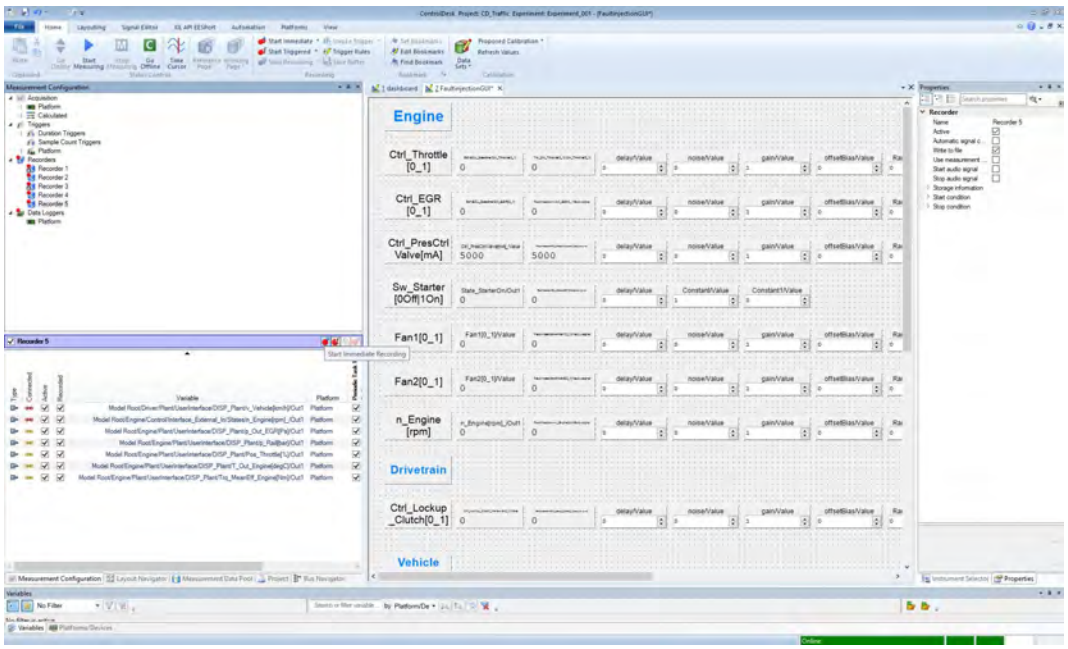


Figure 10.5: GUI of the real-time fault injection.

2. **Steering system for manual driving:**

The manual driving mode is executed via a real steering wheel, gear shift, and pedals. The aforementioned driving elements are linked to the host PC via the USB communication protocol. Prior to the connection of the driving elements to the HIL system, the configuration of the system is performed on the PC, where the devices are defined. Once the configurations have been completed, the ControlDesk tool, which facilitates the connection between the software steering controller and the physical steering wheel, is employed to establish the link between the driving system and the HIL simulation. In addition to enabling the adjust-

ment of variable values, the aforementioned controller also facilitates the calibration of the range in accordance with the specific experimental requirements. Once the steering device is connected, the calibration process with the real-time simulator is initiated. At this phase, the steering angle variable is calibrated manually to align with the three-dimensional virtual driving environment. In this study, the range is set to [300, -300]. Subsequently, the pedal variables, i.e., Pos_ACC, Pos_Brake and Pos_Clutch, are connected and calibrated according to the accelerations, decelerations, and clutch positions, with a range of [0-100]. In the final step, the gears were set up in ControlDesk to be given a specific numerical designation. During the manual test drive, the driving commands are received from the driving system as external inputs to the system in accordance with the user's actions. In the online mode, the vehicle system executes the driving scenario through a real-time simulation process, incorporating the driving instructions provided by the user.

3. **Driving environment and test creation:**

   Once the system models have been selected and imported into the test environment, the test scenarios are designed and created. ModelDesk enables the user to create the test environment, including the road, driving maneuvers, and traffic, in addition to configuring the vehicle. It is possible to create road networks and the associated characteristics of these networks, including lane, line, and sign characteristics; crossing points; elevation, slope, and pavement conditions. In this study, three sections were modeled to capture different conditions within and between cities. The initial segment of the route is from "ISSE" to the central bus station within the city of Clausthal (CLZ). This allows for a validation of the system in consideration of the characteristics typically found in urban areas with regards to roadways and the environment. The modeling process considers a number of scenarios, including pedestrians, traffic lights, and traffic with limited speed. The second zone of interest encompasses the streets situated outside the urban area. In particular, the characteristics of the road between the city of Clausthal (CLZ) and the city of Goslar (GS) are of interest. It was determined that the road characteristics differed significantly in terms of curvature and slope, given the mountainous terrain of the aforementioned direction. Furthermore, the road has two directions of traffic flow. In conclusion, the third domain encompasses a portion of the highway and the entrance

to the city of Brunswick (BS). Figures 10.6b, 10.6d, 10.6f, illustrate that the environmental design process encompassed a wide range of conditions. These included typical conditions within the urban and highway areas, as well as atypical conditions occurring outside of these areas.

The subsequent phase of the study involves the creation of a three-dimensional model of the testing environment, which incorporates various environmental conditions. In order to achieve this objective, the capabilities of the MotionDesk tool were employed. Four distinct environmental conditions were analyzed, i.e., a sunny day, a rainy day, a snowy day, and a foggy day. The aforementioned conditions are depicted in Figures 10.7a, 10.7b, 10.7c, 10.7d, respectively. In addition, the visualization procedure has been designed to accommodate both day and night vision.

The ModelDesk software is utilized for defining the motion of dynamic objects within the driving environment. Such objects may include vehicles or pedestrians with variety of both typical and atypical circumstances. In addition, obstructions and construction were included in the study to analyse and validate the system's behavior in the event of unexpected conditions.

A series of TCs are generated from the test scenario, which includes the input data, the desired output, and the evaluation functionality. In order to ascertain the functionality of the SUT with regard to the requirements specification, requirements-based testing method is employed for the design and implementation of the TCs. Consequently, the TCs are derived from the requirements in order to guarantee a comprehensive level of test coverage. The TCs are organized and carried out in a uniform manner, with the aim of comparing the actual performance of the SUT with the expected performance. The results of the TCs can be evaluated as either pass or fail, contingent upon their execution. It is possible to automatically design and execute a substantial number of TCs using an automation software tool, such as AutomationDesk. Once the test environment, test scenarios, and TCs have been constructed, the resulting experimental data is transferred for execution in real-time on the HIL simulator.

4. **Real-time HIL validation process:**

The ISO 26262 standard provides a clear recommendation regarding the use of HIL simula-

tion in the context of various testing methods during the development process. This is due to the fact that it is of paramount importance for ensuring the safe, flexible, reliable, and realistic simulation of a system [4]. Nevertheless, prior to commencing testing, it is essential to confirm the configuration of the test platform. As depicted in Figure 10.4, the validation process commences with the importation of the system models, i.e., the SUT and the model of the controlled system, which were developed in the preceding phase. The system parameters are defined through the use of ModelDesk, which facilitates the specification of static system variables, commonly referred to as the parameterization process. The Simulink environment facilitates the generation and deployment of system model code to the designated machine. In this case, the object code of the SUT is deployed and executed on the real ECU. Conversely, the executable object code of the controlled system model is executed in the HIL simulator, including the RTICANMM.

Once the model code generated and the test environment developed have been deployed on the target machine, the software application is executed on the target machine through ControlDesk's layout instrumentation. The real-time system execution can be displayed online in the form of online 3D animation in MotionDesk using synchronized software tools. This image also illustrates the system's behavior within a test platform. In other words, the target system is represented in its movement and interaction with other dynamic objects within the three-dimensional virtual environment. The user has the option of executing the driving scenario with complete control over the driving functions, particularly when operating in the manual driving mode. In particular, the driving commands are received from the external driving device, which is under the user's control. Notably, the manual driving does not necessitate adherence to a predefined procedure. Instead, the users have the opportunity to devise their own scenario based on the interactions between humans and machines within the test environment. In the context of online execution, faults are injected into the accessed system signals over the CAN bus network between the ECU and the HIL simulator, contingent upon the configuration of the user-defined fault attributes.

ControlDesk provides users with an extensive array of features for measurement, data recording, calibration, and diagnostics. Upon defining the requisite parameters for the recording system, including the sampling time and target signals, the selected system signals are

recorded as time-series data while the real-time simulator is operational.

During the assessment phase, the recorded behavior of the system is evaluated in comparison with the expected behavior using an evaluation function. This comparison is conducted in the context of automated driving. Any divergence of the intended signals from the anticipated values that is outside the defined tolerance range is considered a fault in accordance with the established evaluation criteria. Consequently, in the event of a fault, the executed TCs are evaluated as failed. In cases where the actual measurement signals fall within the specified expected range, the test results are considered to indicate a successful completion of the test. Conversely, the act of manual driving results in the acquisition of uncertainty data, including noise. In such instances, any anomalous behavior observed by the test driver is identified as a potential failure. Subsequently, the inspector analyses the record of anomalous performance against the intended performance requirements and safety criteria. It is important to note that the process of analyzing the recordings is conducted manually, based on the expertise of the analyst, with the objective of identifying the type and location of the faults that have been detected. Finally, a test report is generated, which contains all findings, observations, and evaluation results. The report is then subjected to evaluation in both driving modes. Thereafter, the report is forwarded to the development team with a view to resolving and mitigating the faults that have been identified.

10.4 Results and Discussion

The following sections present and discuss the findings of the evaluation of the proposed framework. To ascertain the system's behavior under varying internal and external conditions, three distinct test environments were employed, i.e., a non-urban area, an urban environment, and a highway. In order to illustrate the validation process of the system in the presence of disparate environmental conditions, the aforementioned TCs are employed. Furthermore, the internal abnormal condition of the targeted system in the presence of individual and simultaneous faults have been examined.

10.4.1 Validation of the Real-Time Simulation Process:

The efficacy of the proposed framework was evaluated by comparing the outcomes of the real-time simulation of the HIL system to those of the non-real-time simulation, i.e., the MIL, conducted in the MATLAB/Simulink software. To accomplish this objective, the RoadWork_Highway scenario was conducted at the aforementioned test levels (MIL and HIL), utilizing the identical test specifications. A theoretical behavior that aligns with the desired outcome was established as the benchmark against which to gauge the simulation results. As illustrated in Figure 10.8a, a curve colored brown represents this reference behavior. The scenario, designed to simulate a driving situation on a highway, involved the vehicle encountering a construction site. The desired vehicle speed should reach two maximum values, i.e., 100 and 120 km/h, following the commencement of the test at 15 and 70 seconds, respectively. However, according to the vehicle speed reference profile, the vehicle system exhibits a distinct behavioral pattern, maintaining a speed within the range of [100-60] for a period of approximately [19-65] seconds, traversing the construction site on the highway.

As illustrated in Figure 10.8a, the system's behavior in response to the proposed framework is more aligned with the intended outcome than the MIL results. This phenomenon is attributable to the precise execution of the control strategy within the HIL environment, which accounts for the constraints imposed by real-time conditions. In consequence, the system state undergoes a continuous transition with a minimal variation in engine speed (RPM) in comparison to the MIL results, as illustrated in Figure 10.8b. To calculate the relative error, the discrepancy between the desired theoretical behavior and the actual results from the real-time simulation in the HIL environment is calculated. Figure 10.8c illustrates the calculated error with minimal discrepancy while driving in the construction zone. At a point in time 34.8 seconds into the test, the maximum value of the relative error reached 13, while the average value was found to be 2.52. Notwithstanding, the absolute error value increases to 17.6 seconds at the 70-second point due to the abrupt alteration of the system velocity from 60 to 120 km/h. However, it is evident that the HIL system offers greater accuracy and precision in the desired behavior than the MIL results.

Figures 10.8d,e, and f provide a comparison between the results obtained from the model in the MIL and the model in the HIL. It is evident that both simulation results display comparable be-

Table 10.2: Comparison between the results of the proposed method and other related works.

Reference	Average error	Fault injection	Real-time constrains
[268]	4	Not Considered	Considered
[274]	8	Not Considered	Considered
[324]	-	Not Considered	Considered
[325]	5	Not Considered	Considered
[275]	-	Not Considered	Considered
Proposed work	2.52	Considered	Considered

havior, except for instances where the HIL system demonstrates more precise and stable behavior, particularly in the context of driving between 20 and 62 seconds, as illustrated in Figure 10.8e. Figure 10.8f illustrates the status of the brake pedal, demonstrating an accurate response to the control command throughout the driving time. It can be noted that the utilization of an HIL system as a testing framework for virtual test drives results in high accuracy and a reduction in the occurrence of relative errors in comparison to non-real-time simulation platforms.

In order to illustrate the superiority of the proposed framework in comparison to alternative platforms, a Table 10.2 is presented which outlines the relative error rates, the capability of handling FI, and the consideration of real-time constraints for the aforementioned approaches. It can be concluded that the proposed approach exhibits a significantly lower error rate than related works. In addition, the capability to examine the functioning of the system when certain components are faulty during the simulation process illustrates the superiority and applicability of the proposed framework when compared to related studies.

10.4.2 Validation Results under Various Roads and Weather Conditions:

The consideration of irregularities in roadway conditions, such as mountainous roads with pronounced gradients and sharp turns, represents a pivotal element in the examination of the system's conduct in the context of critical scenarios, particularly when the system in question is designated as safety-critical. In this research study, the characteristics of the rural road between GS and CLZ were simulated in order to identify potential faults and problems that might otherwise have gone undetected. Figure 10.9 presents the vehicle system behavior during a virtual test drive performed

manually by the user in real-time for 450 seconds. Figure 10.9a, 10.9b, 10.9c, 10.9d illustrate the recorded system variables, i.e., vehicle speed, engine speed, throttle position, and engine temperature. It should be noted that this scenario was executed in an environment under free-fault conditions. It is evident that the greater the number of turns in the road, the more pronounced the system's reaction. In consideration of these circumstances, the vehicle speed exhibits a multimode behavior, varying between 20 and 100 km/h. It lacks the capacity to maintain a stationary condition. Specifically, in order to traverse curves, the vehicle's speed undergoes a significant decline from 100 km/h to 40 km/h. The speed of the engine, the position of the throttle, and the temperature of the engine frequently vary depending on the specific scenario being tested. It is crucial to recognize that the intricacy of system behavior renders analysis and troubleshooting more challenging.

The operation of vehicles in inclement weather, such as snow or fog, presents a risk at any time of day or night. This framework is designed to encompass the scenarios deemed most critical to the test environment. Its objective is to address the financial and safety constraints associated with real-world testing under weather conditions. The proposed framework encompasses the scenarios that present the most significant risk to the test environment, with the objective of overcoming the financial and safety constraints associated with real-world testing under weather conditions. Furthermore, the developed system has been validated under a range of inclement weather conditions, including sunny, rainy, foggy, and snowy conditions. In such circumstances, the user experiences difficulties in maintaining a smooth and controlled driving technique due to the adverse effects of the weather conditions on vision. The influence of the previously described weather conditions on the user's driving behavior is demonstrated in Figure 10.10. A slight variation in vehicle speed is observed between 0–50 s and 100–150 s, as compared to the fault-free behavior illustrated in Figure 10.9a. In a similar manner, Figure 10.10a demonstrates that the engine speed exhibits a dynamic non-linear behavior, with an uncertain pattern.

10.4.3 Validation Results in Occurrence of Single Faults:

In complex system architectures, unexpected faults in sensors and actuators have the potential to propagate to other components and subsystems. This phenomenon is designated as fault propagation [13]. This propagation can result in significant deficiencies in the system's ability to perform as intended , as shown in Figure 10.12. Failures due to fault propagation constitute a violation of

safety objectives for vehicular functions. This is particularly evident in the case of ADAS and autonomous driving. Such failures can have serious consequences. Consequently, it is of paramount importance to conduct simulations that accurately reflect the system's behavior in the presence of faults in order to effectively mitigate these risks. A comprehensive analysis of the nature and causes of safety risks associated with fault propagation can be conducted through the implementation of such simulations. By employing this approach, the potential dangers can be avoided, and the desired functionality of the system maintained, thus ensuring the overall safety of the system.

The proposed framework considers nine distinct categories of time-series related faults, as detailed in [164]. One such illustration is depicted in Figure 10.11, which depicts the impact of injecting a noise fault into the steering wheel angle sensor. The critical effect on safety-relevant automotive systems, in particular the steering system, was the primary motivation for selecting this sensor for the FI process. The fault was injected at five seconds during manual driving in the urban scenario, i.e., in CLZ City scenario. In comparison to the fault-free behavior, which is represented by the green curve in Figure 10.11b, the injected fault resulted in a divergence from the intended behavior between the times [5 to 30] seconds. However, since the target system is capable of performing the intended function, the resulting departure can be considered to be a tolerable anomaly which does not represent a significant risk.

As depicted in Figure 10.11a, the rationale behind this phenomenon is that the SUT mitigates the fault injected during this temporal window with the aim of achieving the desired behavior. However, the system is no longer able to follow the defined scenario when the ratio of deviation reaches 30 seconds. Consequently, the fault is transmitted from the sensor to the system components, resulting in a system-level failure occurring at 58 seconds. This phenomenon is illustrated in Figure 10.11a. The results of the FI are duly recorded in the test protocol, where comments are also included regarding the effect of the faults and the respective amplitudes. This allows for the analysis process to be repeated by varying one or more configurations of FI attributes. These include the type of faults, its location and the time at which it occurs. The proposed framework provides an efficient solution to safety analysis in a secure and controlled environment. It enables the conduct of reproducible experiments under controlled conditions in real-time. Specifically, the analysis of system behavior under fault conditions during virtual test drives enables the identification of system vulnerabilities and the enhancement of safety mechanisms, thus improving overall safety.

Consequently, this enables the enhancement of the safety characteristics of the target system.

It is essential to emphasize that the proposed framework can be applicable to other vehicle subsystems, including those with internal combustion engines (ICE) and those with electrical or hybrid configurations. This is attributable to the distinctive characteristics of the underlying development and validation processes. Given that the FI process occurs at the CAN bus interface, it is only necessary to provide specifications for the configurations of those attributes associated with the FI. This is because it is sufficient for the purpose of analyzing the fault effects on other systems to determine which attributes are involved.

10.4.4 Validation Results in Occurrence of Concurrent Faults:

In accordance with the ISO 26262 standard, the occurrence of two faults, each in a disparate location, may potentially compromise the safety objectives of a system. The analysis process is considered critical and challenging during testing because it involves the examination of two faults that contribute to the resulting system-level behavior. The integrated FI, which forms a part of the proposed framework, enables concurrent faults to be injected into target components in real-time. This allows for the analysis of safety and reliability characteristics.

In light of the fact that concurrent transients are more significant than permanent faults, this study aims to assess the effect of two sensor-related faults on the system's behavior. Specifically, a gain fault and a noise fault were simultaneously injected into the accelerator pedal sensor and the engine speed sensor, respectively. Both faults were initiated at 170–330 seconds, as illustrated in Figure 10.13. Once the faults were injected, the deviation of the system response from the normal response (green curve) was displayed as a red curve. In this case, the simultaneous rectification of the concurrent faults by the SUT is regarded as an additional complicating factor. Each fault contributed to the erroneous system behavior in a distinct manner. Consequently, the system was temporarily unable to perform its intended functionality. As demonstrated in Figure 10.13d, the engine is incapable of supplying the requisite torque to the propulsion system during fault conditions. Additionally, 10.13b and Figure 10.13c illustrate that engine temperature and rail pressure are also affected by the fault.

The observed change in driving behavior was correlated with the deactivation of vehicle functionality, indicating an alteration in the driving pattern in relation to vehicle speed. Notwithstand-

ing, the cessation of this fault at precisely 330 seconds allows the SUT to resume an optimal operational state. This response is in alignment with the desired outcome. This approach allows the efficient identification of critical concurrent faults and their corresponding attributes, such as type, location, and time. On the other hand, as the number of parameters increases, the number of experiments that can be conducted in order to achieve a high level of test coverage also grows. Consequently, an automation tool has been integrated into the proposed framework to address this issue. This tool automatically performs the injection process. Consequently, according to the stipulations governing the validation of safety requirements, the majority of the analysis process can be completed.

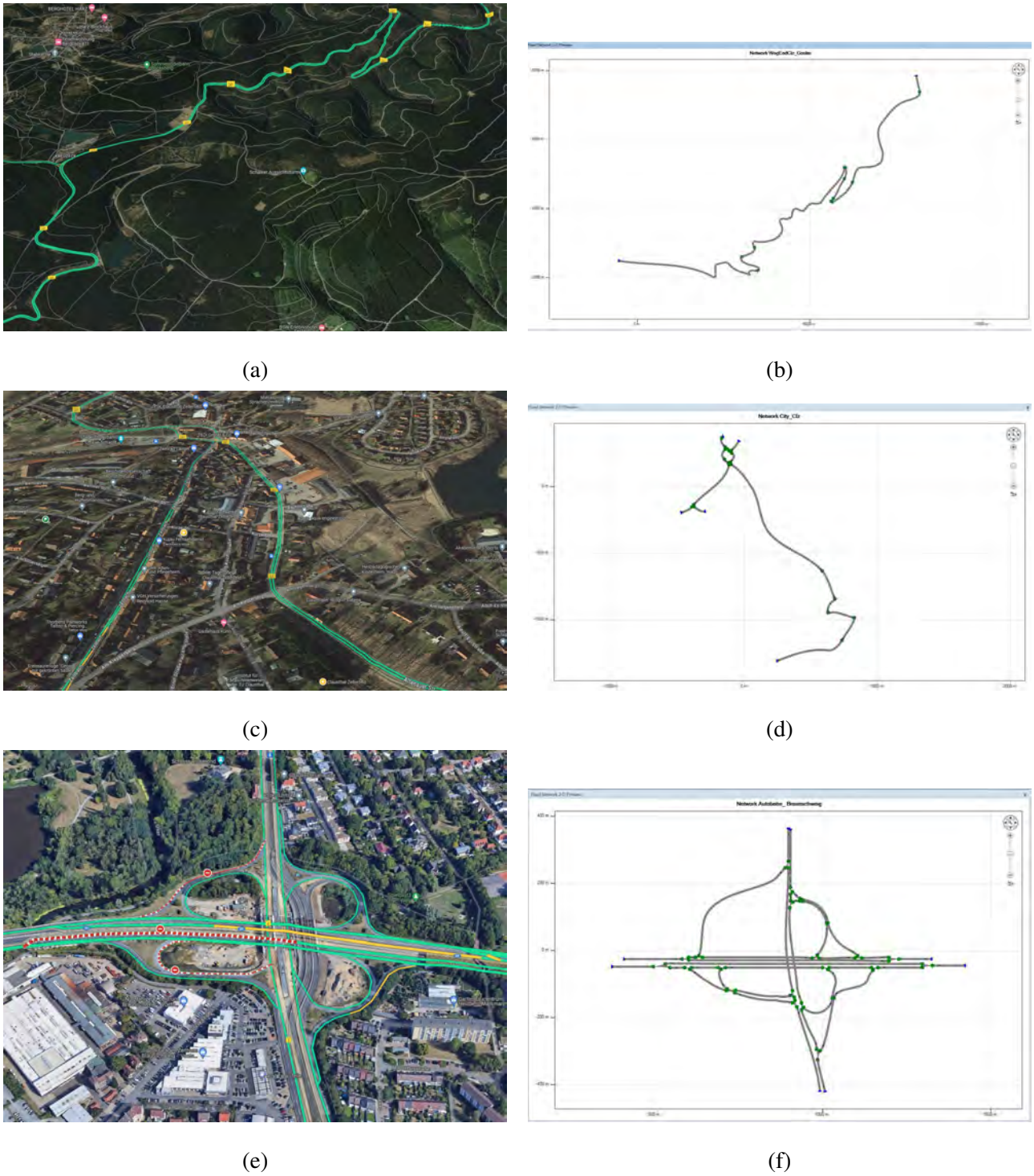
10.5 Conclusions

This chapter introduces a novel real-time testing framework for validating ASSs, which represents a significant advance in the field of automotive safety system engineering. The primary objective of this proposed approach is to facilitate the functionality and safety validation process at the system integration stage of the V-model development approach. In particular, it is designed to align with ISO 26262-4. A virtual test drive is enabled based on HIL real-time simulation, taking into account user behavior. Furthermore, manual driving can be executed along with a real physical steering system, including steering wheel, pedals, and a gearshift. The proposed methodology allows for the analysis of system performance in the presence of faults. In the real-time execution of the system within a target machine, it is possible to inject not only individual faults but also simultaneous faults. Notably, sensor and actuator-related faults have been covered in the proposed approach, including gain, offset/bias, noise, hard-over, spike, stuck-at, packet loss, delay, and drift. The aforementioned faults can be injected into the system components either by manually input or in automated procedure. Furthermore, the faults can be either permanent or transient in nature.

The proposed framework incorporates consideration of a variety of environmental factors, including environmental conditions, traffic and road infrastructure. In particular, the system's behavior can be validated in a range of environmental conditions, including sunny, rainy, foggy, and snowy conditions. In consideration of user behaviors and real-time requirements, the results demonstrate the applicability of the proposed framework to analyzing system behavior under faulty

conditions. Moreover, the framework provides a comprehensive range of test scenarios encompassing diverse weather and road conditions.

In conclusion, the proposed framework allows for the validation of safety-critical systems using a variety of test methods in accordance with the ISO 26262–4 standard. Consequently, the enhanced safety and reliability of the developed system can be achieved while the cost and effort associated with field testing is reduced.

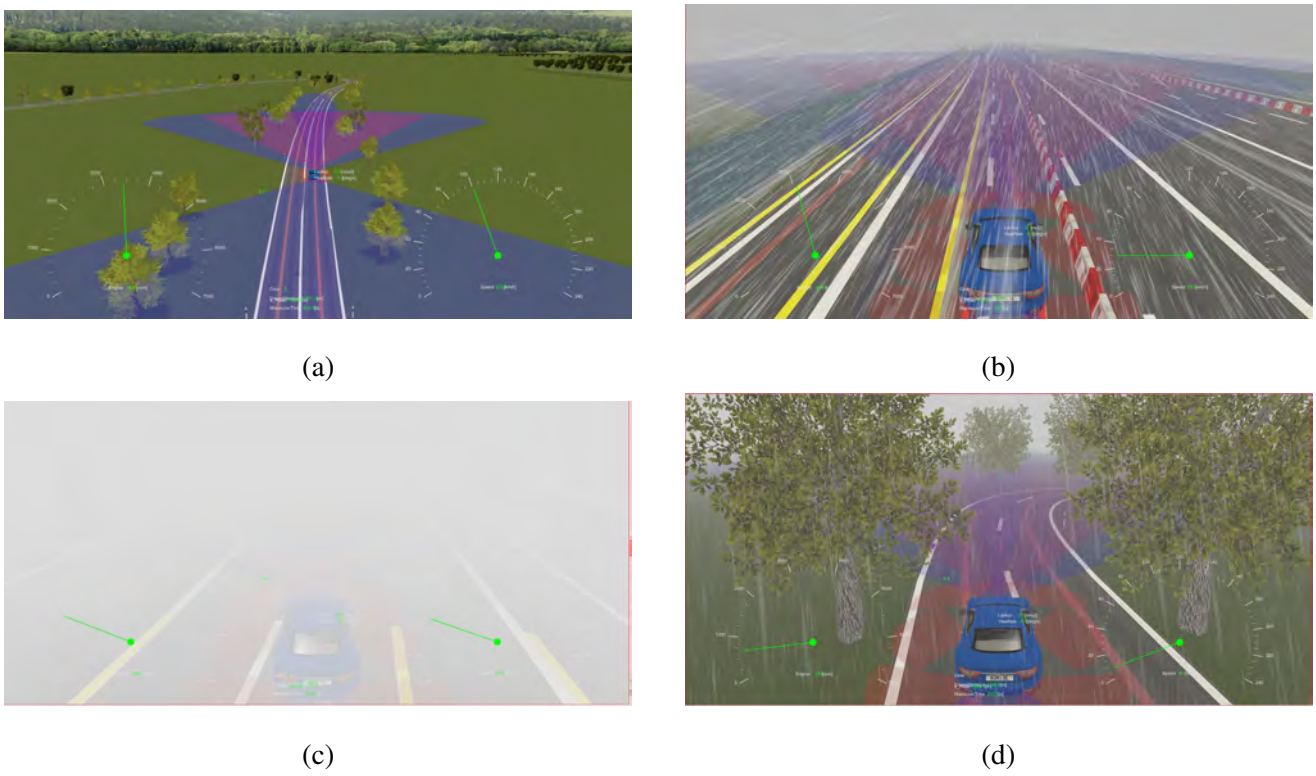


(a) (b)

(c) (d)

(e) (f)

Figure 10.6: Road topology and traffic infrastructure design. (**a**) Real non-urban area (CLZ-GS). (**b**) Designed non-urban area (CLZ-GS). (**c**) Real urban area (CLZ). (**d**) Designed urban area (CLZ).(**e**) highway area (BS). (**f**) Designed highway area (BS).



(a) (b)

(c) (d)

Figure 10.7: Testing environment under various environmental conditions. (**a**) Sunny weather. (**b**) Rainy weather. (**c**) Snowy weather. (**d**) Foggy weather.

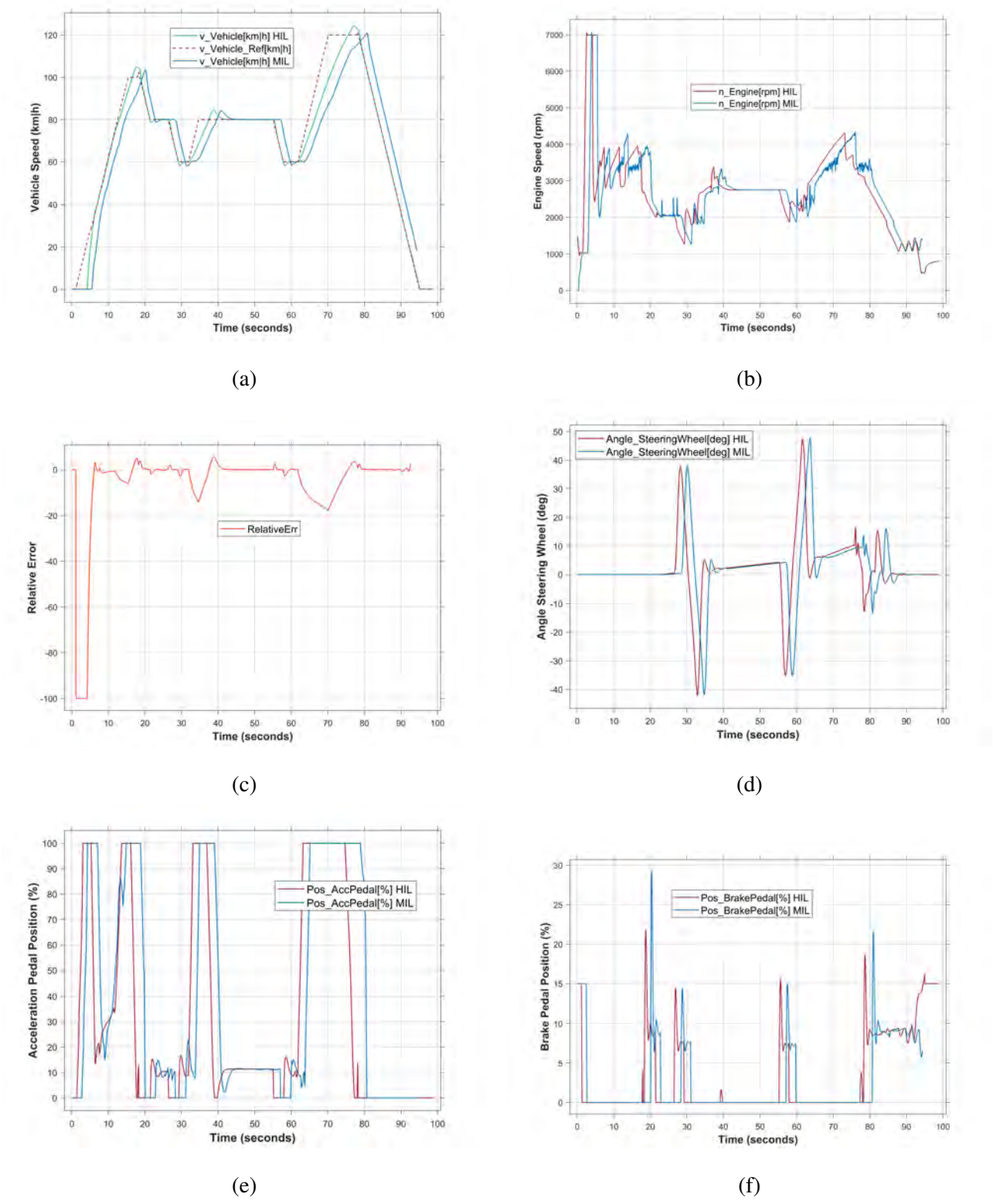
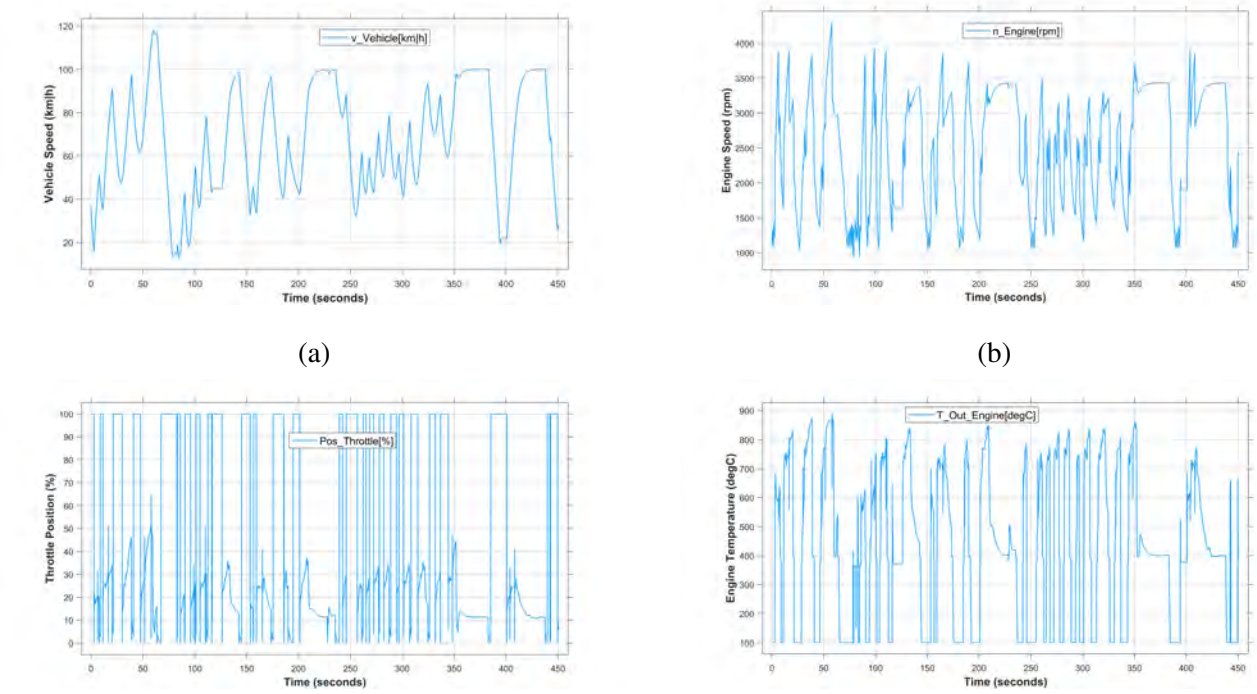


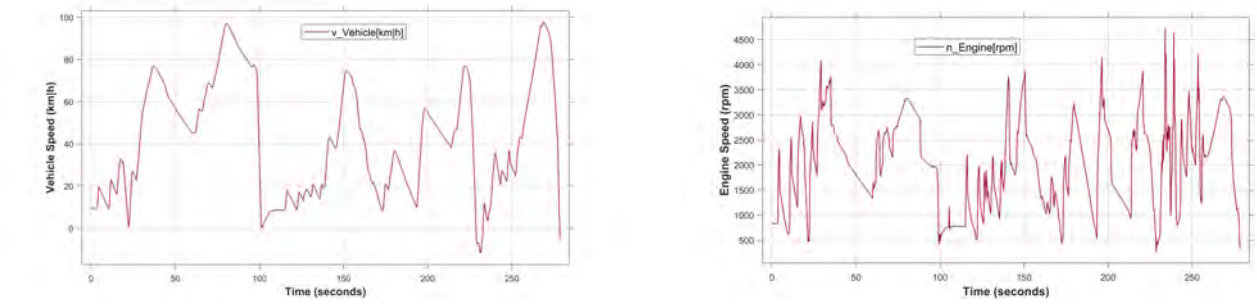
Figure 10.8: Comparison results of the real-time HIL and non-real-time MIL simulation. (**a**) Vehicle speed over the driving time. (**b**) Engine speed over the driving time. (**c**) Relative error of the system performance in term of vehicle speed. (**d**) Angle steering position.(**e**) Acceleration pedal position. (**f**) Brake pedal position.



(a) (b)

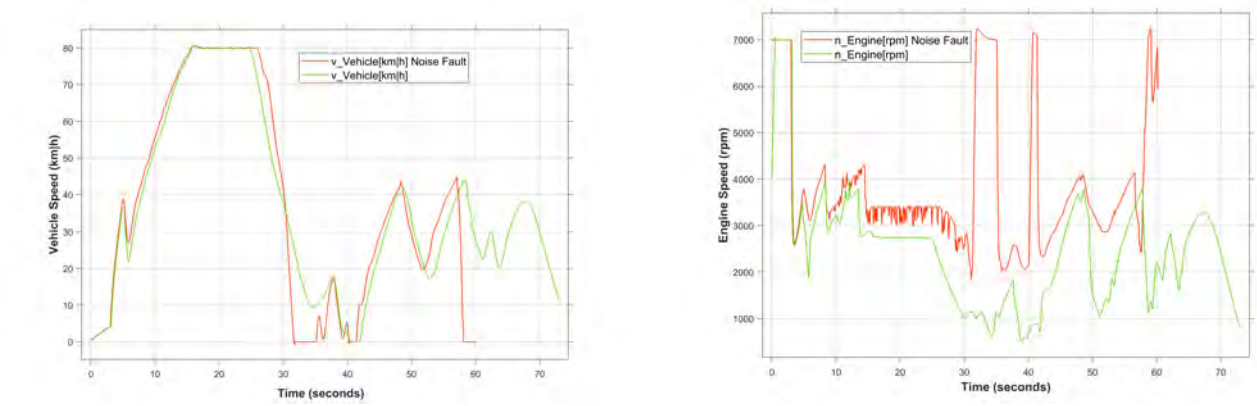
(c) (d)

Figure 10.9: Vehicle system behavior during a real-time virtual test drive. (**a**) vehicle speed. (**b**) engine speed. (**c**) throttle position . (**d**) engine temperature.



(a) (b)

Figure 10.10: User-based driving behavior under foggy weather condition. (**a**) Vehicle speed under abnormal vision condition. (**b**) Engine speed under abnormal vision condition.



(a) (b)

Figure 10.11: System behavior under single fault condition in steering wheel angle. (**a**) Vehicle speed in the occurrence of noise fault. (**b**) Engine speed in the occurrence of noise fault.

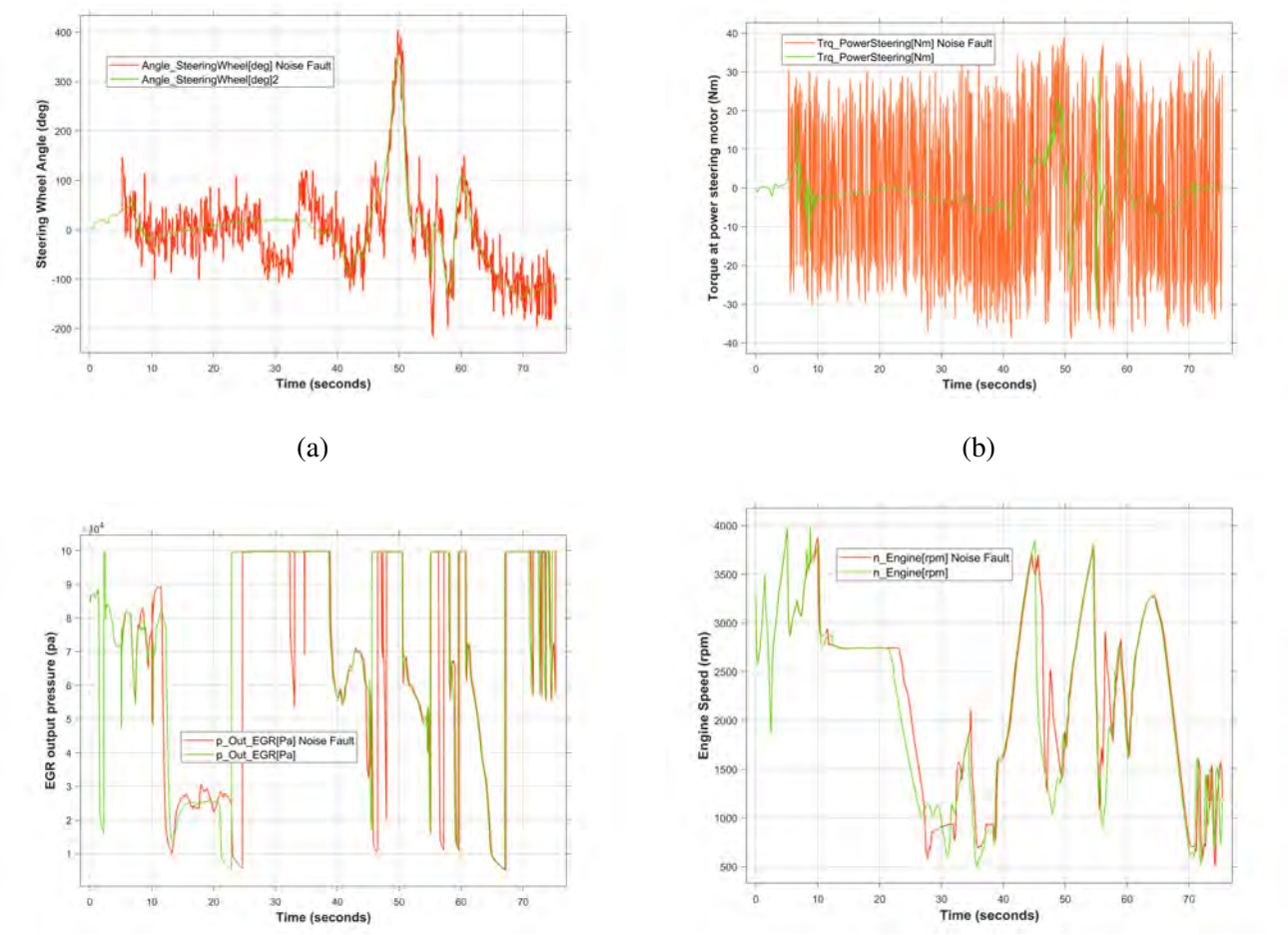


Figure 10.12: The effect of fault propagation on system behavior under single fault condition in steering wheel angle. (**a**) Noise fault injection in the signal of steering wheel angle sensor. (**b**) Torque at power steering motor under propagated noise fault. (**c**) EGR output pressure under propagated noise fault . (**d**) Engine speed under propagated noise fault.

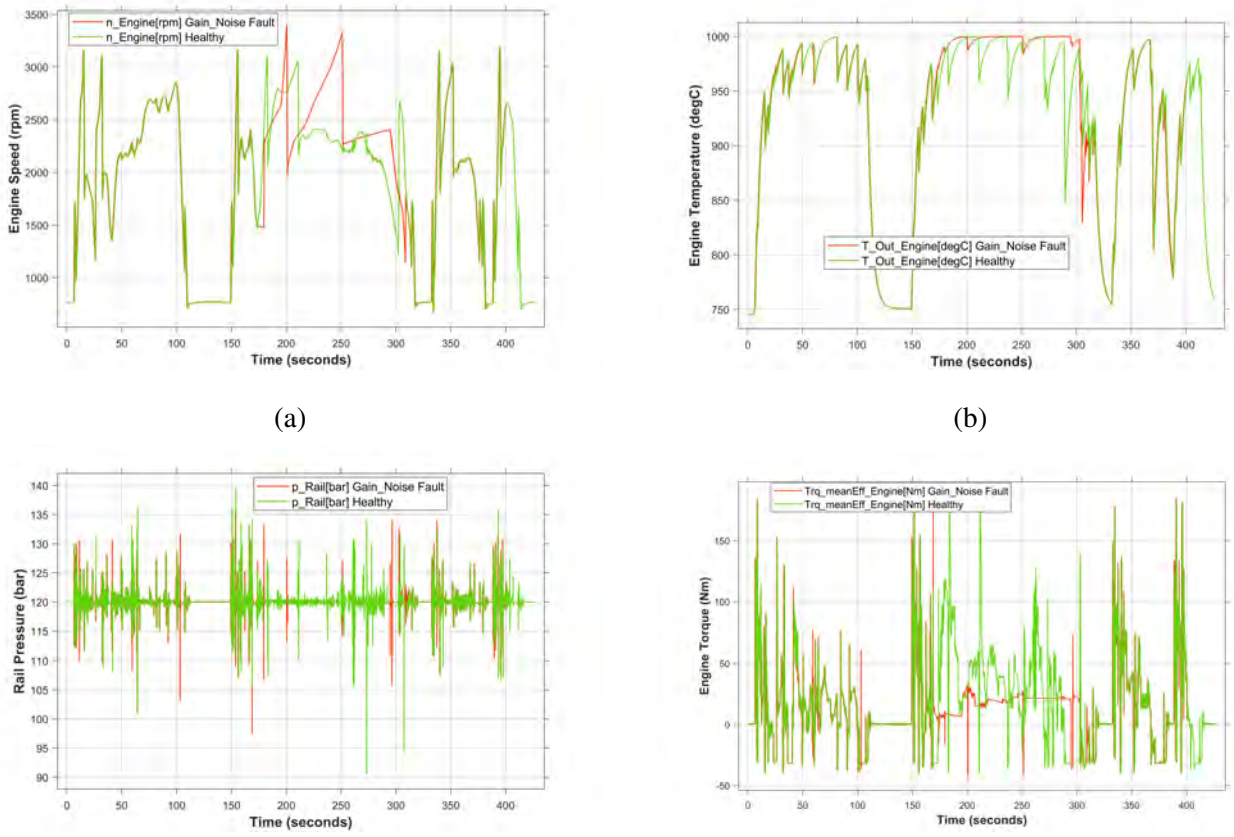


Figure 10.13: System behavior under simultaneous faults condition. (**a**) Engine speed under gain and noise faults. (**b**) Engine temperature under gain and noise faults. (**c**) Rail pressure under gain and noise faults . (**d**) Engine torque under gain and noise faults.

11 Summary and Conclusion

The findings of the research conducted for this thesis are presented in this chapter, with a particular focus on the key contributions and limitations of the proposed approach. In addition, the primary research questions addressed in the dissertation are addressed and discussed. Finally, recommendations for future research directions that build upon the current work are presented.

11.1 Discussion of Results

This study addressed the challenge of failure analysis in the test drive records during the real-time validation of ASSs at the system integration phase of the development process. Given the voluminous nature of the test records, which encompass multivariate nonlinear system behavior, this study endeavors to address the challenge of detecting and classifying critical sensor-related faults at the system level. The objective was achieved by proposing a novel approach integrated with the HIL test process, which depended on the generated representative faulty dataset and used data-driven ML and DL methods. In contrast to conventional FDD methods, the proposed approach is not limited to the detection of a single known fault, but also encompassed the identification of known and unknown simultaneous faults. Furthermore, the potential impact of noisy conditions and imbalanced data on the performance of the FDD models was considered during the models development process. The limitations of existing industrial failure analysis tools have been overcome by developing a model that considers critical faults, which are the root cause of system-level failures.

In this study, based on the mentioned challenges in Chapter 3, Section 3.4, five research questions to be addressed in this study are derived and formulated as the following:

RQ1. How can the system behavior under faulty conditions be captured in real-time without changing the system architecture to generate representative dataset?

RQ2. How can the diagnosis model of single sensor faults be improved while maintaining comprehensive class coverage?

RQ3. How can a robust model be developed for diagnosing unknown simultaneous sensor faults under noisy conditions?

RQ4. How can a known simultaneous FDD model be developed for the recordings analysis considering the noisy and imbalanced data?

RQ5. How can a demonstration prototype be developed to evaluate the SUT and validate the proposed concept based on real-time HIL simulations?

The contributions to the mentioned research questions defined in the scope of this study's approach are discussed in the following subsections

11.1.1 Main Contributions

1. **Hardware-in-the-Loop-Based Real-Time Fault Injection Framework for Dynamic Behavior Analysis of ASSs and representative real-time dataset generation:** (Chapter 6)

The objective of this dissertation is to propose a real-time FI framework for ASSs testing, with a view to addressing the first research question (RQ1). In order to achieve this objective, a HIL simulation platform is employed to develop and implement the proposed framework. The functionality of the real ECU and the entire system model, along with its surrounding environment, are taken into account. The objective is to analyze the dynamic behavior of complex ASSs during the V-cycle development process, i.e., the integration phase. Thus, the critical fault that resulted in the violation of the safety goal can be identified. Consequently, the corresponding behavior is captured providing a representative dataset for the application of ML.

A synopsis of the principal contributions of this study is provided below: (1) The proposed framework is designed to analyze the impact of faults on complex systems under realistic operational conditions and generate faulty datasets in real-time. (2) The framework facilitates the automated analysis of system behavior, thereby enabling the identification of critical faults that may potentially lead to a violation of functional safety requirements. Furthermore, the framework

allows the validation of a real or virtual ECU's performance under both normal and faulty operational conditions, utilizing intricate examination scenarios. Subsequently, the majority of the most common faults in the signals of ASSs are considered, including gain, offset/bias, noise, hard-over, spike, stuck-at, packet loss, delay and drift faults. (5) Moreover, the framework allows for the simulation and injection of not only single faults but also multiple faults, thus enabling an analysis of the impact of simultaneous faults on the system. (6) Finally, fault are programmatically injected in real-time through the HIL platform, which, in turn, obviates the need to modify the original system model or add additional blocks.

To demonstrate the benefits and applicability of the proposed framework, a comprehensive simulation of a gasoline engine system is employed as a case study. Furthermore, vehicle dynamics, environmental conditions, driver model, and powertrain models have been incorporated into the framework to facilitate the detailed characterization of the vehicle during the data acquisition phase.

- **Intelligent Fault Detection and Classification Based on Hybrid DL Methods for system integration Test of ASSs:** (Chapter 7)

In order to overcome the limitations of a single DL-based FDD and address the second research question (RQ2), this dissertation proposes a novel FDD approach for a system integration test of the ASS development, as a classification problem. In particular, an intelligent FDD model is developed and implemented based on hybrid DL techniques. The objective of the developed model is to accurately identify and classify fault types in automotive sensors and communication signals. To address the issue of representative training datasets, a novel real-time FI framework was developed along with an HIL simulation system. Consequently, a faults dataset is generated in real-time, which is subsequently utilized for the training, validation, and assessment of the model. The proposed methodology is demonstrated through the use of a complex automotive gasoline engine system as a case study, which considers the vehicle dynamics along with the environmental system model. The efficiency of the model was assessed in terms of precision, recall, and F1-score, utilizing an unseen testing dataset.

The key findings of the study are presented below. (1) A novel approach for developing a FDD model of automotive sensor signals has been proposed for utilization during the V-cycle of ASSs development. To this end, the integration of the LSTM and 1D-CNN is employed with the objective

of capitalizing on the respective strengths of each while mitigating their respective limitations. (2) To address the paucity of fault datasets, fault modes are simulated along with intricate test scenarios without modifying the original system model. Consequently, the fault types were injected in programmed manner in real-time using a real-time FI framework with HIL platform. In addition, a total of eight distinct types of sensor and communication faults in vehicle signals were taken into consideration as distinct fault classes. This included the spike, offset/bias, noise, hard-over, delay, gain, stuck-at, and drift faults. (4) In order to demonstrate the superiority of our proposed model, a comparison has been made of the classification performance in respect of each class with that of a standalone DL technique-based model, i.e., CNN and LSTM technique.

- **GRU-based Denoising Autoencoder for Detection and Clustering of Unknown Concurrent Faults during System Integration Testing of ASSs:** (Chapter 8)

In addressing the third research question (RQ3), the objective was to fill the gaps in current research on simultaneous FDD for ASSs development under noisy conditions. This dissertation presents a novel intelligent method for the detection and clustering of unknown faults during the system development process. The developed model is able to detect and categorize both known and unknown faults in the sequential data of sensor signals during real-time test drive.

The principal contributions of the proposed study are outlined below. (1) A novel DL architecture based on deep features extraction and multi-level clustering has been developed. In essence, this architecture effectively addresses the problem of detecting and clustering both single and simultaneous faults in time-series data. The proposed method employs a hybrid GRU-based DAE and K-means for the detection and clustering of faults in time-series data. Moreover, the combination of the proposed techniques enables the detection of both known and unknown faults, under a range of different noise conditions. This renders it an appropriate choice for real-world industrial use in conditions where noise levels are high. (3) Furthermore, an investigation has been conducted into the comparative analysis and discussion of various alternative DAE architectural variants utilizing both healthy and faults validation datasets. (4) A high-fidelity simulation model of a complex gasoline engine and vehicle dynamic system has been employed as a case study. The applicability and robust performance of the proposed method have been demonstrated based on testing data from manual and automated virtual test drives.

- **Simultaneous Fault Detection and Classification using DL Methods for Recordings Analysis:** (Chapter 9)

Although the current FDD models have yielded encouraging results, the majority of previous studies have been conducted for a single fault, without consideration of the potential impact of concurrent multiple faults. Furthermore, the majority of current methodologies depend on the quality and availability of labelled datasets, which should include a predefined set of known fault classes. Furthermore, the occurrence of multiple faults simultaneously makes it impossible to identify a vast number of combination classes to encompass all fault types within noisy and imbalanced data. In order to address this challenge and answer to the fourth research question (RQ4), this study proposes an ensemble DL classifier that can detect and classify single and concurrent faults. A synthesis of the findings from this study yields the following conclusions: (1) This study proposes an innovative and effective ensemble learning-based simultaneous fault detection and identification approach. Specifically, a multi-label ensemble of LSTM and RF-based classifiers is developed. (2) In order to enhance the efficacy and resilience of the developed FDD model in the presence of varying levels of noise, a novel framework based on GRU-based DAE has been implemented. (3) The challenge of FDD model development in the context of datasets with inherent imbalance is overcome through the application of the random undersampling algorithm. Consequently, the classification accuracy of the minority classes is enhanced. (4) The efficacy of the proposed methodology was assessed through the utilization of real-time automotive simulation data across a spectrum of noise levels, with the outcomes also subjected to a comparative analysis with standalone methodologies.

- **Real-Time Validation Platform for Automotive Embedded Software Systems based on HIL and FI:** (Chapter 10)

This dissertation proposes a novel real-time virtual test drive based on HIL simulation to address the limitations of current test drive platforms. The proposed solution addresses the fifth research question (RQ5). Specifically, the SUT is validated through the consideration of the driver's behavior under normal and abnormal operating conditions within the system components. Furthermore, to ensure a high level of realism, a dynamic traffic environment has been created as a three-dimensional visualization. In order to illustrate the benefits and capabilities of the proposed

platform, a highly realistic automotive system model is employed as a case study. This model, which is a complex gasoline engine model with vehicle dynamics, serves to demonstrate the advantages of the platform.

In essence, this research contributes to the following: (1) A novel driving simulator has been developed which is capable of real-time analysis of the system's behavior in the event of a fault. The simulator is specifically designed to analyze faults in sensors and actuators, which are common causes of malfunction in automotive systems. (2) To include the tester's behavior, physical drive elements were connected to the HIL, including the steering wheel and pedals, for real and virtual ECU validation under normal and abnormal conditions. (3) In contrast to traditional test methods, the HIL enables the injection of fault types programatically at the components level. This approach allows the testing of different fault scenarios without necessitating changes to the original system model. (4) The construction of a three-dimensional visualization of the driving environment is modelled, with the inclusion of dynamic traffic, obstacles, weather conditions, and traffic congestion. This is being conducted in a manner analogous to the real environment, specifically in the regions of Clausthal, Goslar, and Braunschweig. (5) A sophisticated gasoline engine model has been incorporated with the highly-fidelity full-vehicle models in order to accurately capture the characteristics and properties of the system.

11.1.2 Limitations

In order to provide a comprehensive definition of the scope of this work, this section offers an overview of other challenges that have not been addressed in this thesis. In accordance with the research questions delineated in the preceding sections, this study concentrates on the FDD aspects as a classification and clustering task. Nevertheless, other aspects, such as fault localization and fault forecasting, could be considered as potential avenues for future research. Furthermore, this section concludes with a description of the types of faults and potential locations that do not fall within the scope of this study.

- **Fault Localization/ Isolation:**

The process of determining the locations of faults within a system, i.e., the faulty components, is known as fault localization or isolation. The process of isolating the faulty components that can

lead to a system-level failure is of vital importance in ensuring the safety and reliability of the overall system. However, this task falls outside the scope of the present study. This study addresses the FDD task, which concerns determining the nature of the detected faults based on the patterns in the time series data.

- **Fault Forecasting:**

In certain engineering applications, such as the development of autonomous vehicles, it is of paramount importance to predict the occurrence of faults in a system before they occur. This enables the development of a proactive model, which in turn allows for the implementation of preventive action and timely maintenance. Consequently, the overall reliability of the system can be guaranteed. Nevertheless, this aspect of fault prediction is not addressed in this study.

- **Fault Locations:**

Given the random and systematic faults that can occur during system integration testing, this work has been limited to sensor-related faults as a potential fault location. Nevertheless, other system components may also be susceptible to faults, including ECUs and the network. Specifically, this study does not consider network-related faults, such as wiring harness, physical topology, and network configuration. Similarly, faults in the software and hardware of the ECU are not relevant to this work. A number of examples of ECU software faults can be identified, including those in application software, base software, and parameterization and versioning. ECU hardware faults include ECU layout and ECU internal components.

11.2 Future Work

The opportunities for future research can be summarized as the following:

- Future research should aim to expand the scope of the proposed real-time FI framework to encompass the potential faults in other system components, such as the ECU and network. Furthermore, incorporating DL techniques into the framework would enhance its capabilities during the validation process with virtual test drive. This would enable the automated, effective, and systematic generation of fault TCs, aligned with specific requirements, thus advancing the framework's overall effectiveness and utility, as demonstrated in [326].

- In light of the findings in this study, it can be noted that the proposed fault classification methodology may also prove useful for the analysis of real-world test drives conducted on public roads. This would permit the intelligent detection and classification of unexpected faults without the necessity for human intervention. In light of these considerations, it is recommended that during the development of the FDD model, uncertain data with advanced processing techniques be considered as a potential area for further investigation into advanced feature engineering approaches. In addition, further development of the proposed model is required to enhance its performance in the presence of high noise.

- The proposed methodology for clustering unknown faults in the presence of noise could be extended to include the identification of faulty components in developed systems and the underlying causes of those faults. In order to attain this objective, it is essential to use the historical data encompassing representative faulty sensors and actuators to facilitate the development of the target model. It is therefore necessary to develop novel techniques for intelligent fault detection and localization which take into account the problem of the unavailability of representative faulty data.

- It is possible to enhance the precision and efficiency of the proposed ensemble models in detecting and identifying simultaneous faults. Furthermore, it may be beneficial to examine the applicability of other straightforward DL architectures in the construction of ensemble classifiers for subsequent development stages of ASSs. Besides, the adaptability and applicability of the proposed model to FDD issues in systems originating from other industrial sectors, e.g., railway and aviation, can be further explored.

- Finally, the proposed virtual test drive framework can be extended to encompass other test methods, such as interface tests and resource usage tests. In addition, the incorporation of AI-driven intelligent models into the aforementioned framework facilitates the real-time analysis of recorded data. The proposed approach can also be enhanced by integrating real vehicle components and subsystems into the framework. This integration can be achieved by incorporating real fuel pressure sensors and throttle valves, for instance. By doing so, the uncertainty of the data caused by real elements can be accounted during the validation process under realistic real-time conditions.

Bibliography

[1] "Iso 26262-10:2012 - road vehicles — functional safety — part 10: Guideline on iso 26262," https://www.iso.org/standard/54591.html, (Accessed on 11/22/2021).

[2] S. Fürst, J. Mössinger, S. Bunzel, T. Weber, F. Kirschke-Biller, P. Heitkämper, G. Kinkelin, K. Nishikawa, and K. Lange, "Autosar–a worldwide standard is on the road," in *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, vol. 62, no. 5. Citeseer, 2009.

[3] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, 1997.

[4] A. Himmler, K. Lamberg, and M. Beine, "Hardware-in-the-loop testing in the context of iso 26262," SAE Technical Paper, Tech. Rep., 2012.

[5] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.

[6] J. D'Ambrosio and G. Soremekun, "Systems engineering challenges and mbse opportunities for automotive system design," in *2017 IEEE international conference on systems, man, and cybernetics (SMC)*. IEEE, 2017, pp. 2075–2080.

[7] C. Ebert and J. Favaro, "Automotive software," *IEEE Software*, vol. 34, no. 03, pp. 33–39, 2017.

[8] L. L. Bello, R. Mariani, S. Mubeen, and S. Saponara, "Recent advances and trends in on-board embedded and networked automotive systems," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1038–1051, 2018.

[9] P. Panaroni, G. Sartori, F. Fabbrini, M. Fusani, and G. Lami, "Safety in automotive software: An overview of current practices," in *2008 32nd Annual IEEE International Computer Software and Applications Conference*. IEEE, 2008, pp. 1053–1058.

[10] E. Bringmann and A. Krämer, "Systematic testing of the continuous behavior of automotive systems," in *Proceedings of the 2006 international workshop on Software engineering for automotive systems*, 2006, pp. 13–20.

[11] K. Ding, A. Morozov, and K. Janschek, "Classification of hierarchical fault-tolerant design patterns," in *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2017, pp. 612–619.

[12] E. Balaban, A. Saxena, P. Bansal, K. F. Goebel, and S. Curran, "Modeling, detection, and disambiguation of sensor faults for aerospace applications," *IEEE Sensors Journal*, vol. 9, no. 12, pp. 1907–1917, 2009.

[13] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE transactions on dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004.

[14] M. Gentile and A. E. Summers, "Random, systematic, and common cause failure: How do you manage them?" *Process safety progress*, vol. 25, no. 4, pp. 331–338, 2006.

[15] "Iso 26262-10:2018 road vehicles functional safety," https://www.iso.org/standard/68392.html, (Accessed on 01/06/2024).

[16] A. R. Plummer, "Model-in-the-loop testing," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 220, no. 3, pp. 183–199, 2006.

[17] A. Bittar, H. V. Figuereido, P. A. Guimaraes, and A. C. Mendes, "Guidance software-in-the-loop simulation using x-plane and simulink for uavs," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2014, pp. 993–1002.

[18] J. Mina, Z. Flores, E. López, A. Pérez, and J.-H. Calleja, "Processor-in-the-loop and hardware-in-the-loop simulation of electric systems based in fpga," in *2016 13th International Conference on Power Electronics (CIEP)*. IEEE, 2016, pp. 172–177.

[19] R. Isermann, J. Schaffnit, and S. Sinsel, "Hardware-in-the-loop simulation for the design and testing of engine-control systems," *Control Engineering Practice*, vol. 7, no. 5, pp. 643–653, 1999.

[20] T. Bokc, M. Maurer, and G. Farber, "Validation of the vehicle in the loop (vil); a milestone for the simulation of driver assistance systems," in *2007 IEEE Intelligent vehicles symposium*. IEEE, 2007, pp. 612–617.

[21] W. Damm, E. Möhlmann, T. Peikenkamp, and A. Rakow, "A formal semantics for traffic sequence charts," *Principles of Modeling: Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday*, pp. 182–205, 2018.

[22] V. Garousi, M. Felderer, Ç. M. Karapıçak, and U. Yılmaz, "Testing embedded software: A survey of the literature," *Information and Software Technology*, vol. 104, pp. 14–45, 2018.

[23] G. Tibba, C. Malz, C. Stoermer, N. Nagarajan, L. Zhang, and S. Chakraborty, "Testing automotive embedded systems under x-in-the-loop setups," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, 2016, pp. 1–8.

[24] O. Gietelink, J. Ploeg, B. De Schutter, and M. Verhaegen, "Development of a driver information and warning system with vehicle hardware-in-the-loop simulations," *Mechatronics*, vol. 19, no. 7, pp. 1091–1104, 2009.

[25] R. Kiefer, D. LeBlanc, M. Palmer, J. Salinger, R. K. Deering, M. Shulman *et al.*, "Development and validation of functional definitions and evaluation procedures for collision warning/avoidance systems," United States. Department of Transportation. National Highway Traffic Safety, Tech. Rep., 1999.

[26] A. Theissler, "Detecting anomalies in multivariate time series from automotive systems," Ph.D. dissertation, Brunel University School of Engineering and Design PhD Theses, 2013.

[27] Z. Szalay, M. Szalai, B. Tóth, T. Tettamanti, and V. Tihanyi, "Proof of concept for scenario-in-the-loop (scil) testing for autonomous vehicle technology," in *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*. IEEE, 2019, pp. 1–5.

[28] K. Athanasas, C. Bonnet, H. Fritz, C. Scheidler, and G. Volk, "Valse-validation of safety-related driver assistance systems," in *IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No. 03TH8683)*. IEEE, 2003, pp. 610–615.

[29] S. Otten, J. Bach, C. Wohlfahrt, C. King, J. Lier, H. Schmid, S. Schmerler, and E. Sax, "Automated assessment and evaluation of digital test drives," in *Advanced Microsystems for Automotive Applications 2017: Smart Systems Transforming the Automobile*. Springer, 2018, pp. 189–199.

[30] Y. Chen, S. Chen, T. Zhang, S. Zhang, and N. Zheng, "Autonomous vehicle testing and validation platform: Integrated simulation system with hardware in the loop," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 949–956.

[31] Z. Filipi, H. Fathy, J. Hagena, A. Knafl, R. Ahlawat, J. Liu, D. Jung, D. Assanis, H. Peng, and J. Stein, "Engine-in-the-loop testing for evaluating hybrid propulsion concepts and transient emissions–hmmwv case study," *SAE Transactions*, pp. 23–41, 2006.

[32] V. V. Nair and B. P. Koustubh, "Data analysis techniques for fault detection in hybrid/electric vehicles," in *2017 IEEE Transportation Electrification Conference (ITEC-India)*. IEEE, 2017, pp. 1–5.

[33] C. V. Jordan, F. Hauer, P. Foth, and A. Pretschner, "Time-series-based clustering for failure analysis in hardware-in-the-loop setups: an automotive case study," in *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2020, pp. 67–72.

[34] A. Theissler and I. Dear, "An anomaly detection approach to detect unexpected faults in recordings from test drives," *International Journal of Computer and Information Engineering*, vol. 7, no. 7, pp. 958–965, 2013.

[35] X. Liu, Q. Zhou, J. Zhao, H. Shen, and X. Xiong, "Fault diagnosis of rotating machinery under noisy environment conditions based on a 1-d convolutional autoencoder and 1-d convolutional neural network," *Sensors*, vol. 19, no. 4, p. 972, 2019.

[36] A. Theissler, "Multi-class novelty detection in diagnostic trouble codes from repair shops," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*. IEEE, 2017, pp. 1043–1049.

[37] S. Radack, "The system development life cycle (sdlc)," National Institute of Standards and Technology, Tech. Rep., 2009.

[38] S. T. Acuña and X. Ferré, "Software process modelling." in *ISAS-SCI (1)*. Citeseer, 2001, pp. 237–242.

[39] S. Shylesh, "A study of software development life cycle process models," in *National Conference on Reinventing Opportunities in Management, IT, and Social Sciences*, 2017, pp. 534–541.

[40] Z. Bi, L. Da Xu, and C. Wang, "Internet of things for enterprise systems of modern manufacturing," *IEEE Transactions on industrial informatics*, vol. 10, no. 2, pp. 1537–1546, 2014.

[41] R. R. Young, *Effective requirements practices*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[42] G. J. Myers, *The art of software testing*. John Wiley & Sons, 2006.

[43] C. Szyperski, D. Gruntz, and S. Murer, *Component software: beyond object-oriented programming*. Pearson Education, 2002.

[44] N. B. Ruparelia, "Software development lifecycle models," *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 3, pp. 8–13, 2010.

[45] L. Pintard, J.-C. Fabre, M. Leeman, K. Kanoun, and M. Roy, "From safety analyses to experimental validation of automotive embedded systems," in *2014 IEEE 20th Pacific Rim International Symposium on Dependable Computing*. IEEE, 2014, pp. 125–134.

[46] J. C. Jensen, D. H. Chang, and E. A. Lee, "A model-based design methodology for cyber-physical systems," in *2011 7th international wireless communications and mobile computing conference*. IEEE, 2011, pp. 1666–1671.

[47] B. W. Boehm, "Improving software productivity," *Computer*, vol. 20, no. 09, pp. 43–57, 1987.

[48] T. Erkkinen and M. Conrad, "Verification, validation, and test with model-based design," SAE Technical Paper, Tech. Rep., 2008.

[49] R. E. Shannon, "Introduction to the art and science of simulation," in *1998 winter simulation conference. proceedings (cat. no. 98ch36274)*, vol. 1. IEEE, 1998, pp. 7–14.

[50] "Matlab," https://www.mathworks.com/products/matlab.html, (Accessed on 06/12/2023).

[51] S. T. Karris, *Introduction to Simulink with engineering applications*. Orchard Publications, 2006.

[52] S. Mathur and S. Malik, "Advancements in the v-model," *International Journal of Computer Applications*, vol. 1, no. 12, pp. 29–34, 2010.

[53] S. Balaji and M. S. Murugaiyan, "Waterfall vs. v-model vs. agile: A comparative study on sdlc," *International Journal of Information Technology and Business Management*, vol. 2, no. 1, pp. 26–30, 2012.

[54] L. Pintard, "From safety analysis to experimental validation by fault injection-case of automotive embedded systems," Ph.D. dissertation, 2015.

[55] R. Isermann and P. Balle, "Trends in the application of model-based fault detection and diagnosis of technical processes," *Control engineering practice*, vol. 5, no. 5, pp. 709–719, 1997.

[56] A. Dorling, "Spice: Software process improvement and capability determination," *Software Quality Journal*, vol. 2, pp. 209–224, 1993.

[57] R. Messnarz, H.-L. Ross, S. Habel, F. König, A. Koundoussi, J. Unterrreitmayer, and D. Ekert, "Integrated automotive spice and safety assessments," *Software Process: Improvement and Practice*, vol. 14, no. 5, pp. 279–288, 2009.

[58] N. Ehsan, A. Perwaiz, J. Arif, E. Mirza, and A. Ishaque, "Cmmi/spice based process improvement," in *2010 IEEE International Conference on Management of Innovation & Technology*. IEEE, 2010, pp. 859–862.

[59] "Iso 26262-1:2018(en), road vehicles — functional safety — part 1: Vocabulary," https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-2:v1:en, (Accessed on 12/13/2021).

[60] J. Birch, R. Rivett, I. Habli, B. Bradshaw, J. Botham, D. Higham, P. Jesty, H. Monkhouse, and R. Palin, "Safety cases and their role in iso 26262 functional safety assessment," in *Computer Safety, Reliability, and Security: 32nd International Conference, SAFECOMP 2013, Toulouse, France, September 24-27, 2013. Proceedings 32*. Springer, 2013, pp. 154–165.

[61] M. Staron, M. Staron, and D. Durisic, "Autosar standard," *Automotive Software Architectures: An Introduction*, pp. 81–116, 2017.

[62] H. Fennel, S. Bunzel, H. Heinecke, J. Bielefeld, S. Fürst, K.-P. Schnelle, W. Grote, N. Maldener, T. Weber, F. Wohlgemuth *et al.*, "Achievements and exploitation of the autosar development partnership," SAE Technical Paper, Tech. Rep., 2006.

[63] P. Folkesson, F. Ayatolahi, B. Sangchoolie, J. Vinter, M. Islam, and J. Karlsson, "Back-to-back fault injection testing in model-based development," in *Computer Safety, Reliability, and Security: 34th International Conference, SAFECOMP 2015, Delft, The Netherlands, September 23-25, 2015, Proceedings 34*. Springer, 2015, pp. 135–148.

[64] M. Horauer, M. Zauner, and H. Schuster, "A system-level test for automotive communication subsystems," *Elektrotechnik und Informationstechnik*, vol. 128, no. 6, p. 215, 2011.

[65] H. Ziade, R. A. Ayoubi, R. Velazco *et al.*, "A survey on fault injection techniques," *Int. Arab J. Inf. Technol.*, vol. 1, no. 2, pp. 171–186, 2004.

[66] A. Floridia, E. Sanchez, and M. S. Reorda, "Fault grading techniques of software test libraries for safety-critical applications," *IEEE Access*, vol. 7, pp. 63 578–63 587, 2019.

[67] A. Benso and P. Prinetto, *Fault injection techniques and tools for embedded systems reliability evaluation*. Springer Science & Business Media, 2003, vol. 23.

[68] P. Marwedel, *Embedded system design: embedded systems foundations of cyber-physical systems, and the internet of things*. Springer Nature, 2021.

[69] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell, "Fault injection for dependability validation: A methodology and some applications," *IEEE Transactions on software engineering*, vol. 16, no. 2, pp. 166–182, 1990.

[70] M. Cukier, D. Powell, and J. Ariat, "Coverage estimation methods for stratified fault-injection," *IEEE Transactions on Computers*, vol. 48, no. 7, pp. 707–723, 1999.

[71] R. Natella, D. Cotroneo, and H. S. Madeira, "Assessing dependability with software fault injection: A survey," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, pp. 1–55, 2016.

[72] J. Carreira, H. Madeira, and J. G. Silva, "Xception: A technique for the experimental evaluation of dependability in modern computers," *IEEE Transactions on Software Engineering*, vol. 24, no. 2, pp. 125–136, 1998.

[73] Z. Segall, D. Vrsalovic, D. Siewiorek, D. Ysskin, J. Kownacki, J. Barton, R. Dancey, A. Robinson, and T. Lin, "Fiat-fault injection based automated testing environment," in *Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995,'Highlights from Twenty-Five Years'*. IEEE, 1995, p. 394.

[74] D. T. Stott, B. Floering, D. Burke, Z. Kalbarczpk, and R. K. Iyer, "Nftape: a framework for assessing dependability in distributed systems with lightweight fault injectors," in *Proceedings IEEE International Computer Performance and Dependability Symposium. IPDS 2000*. IEEE, 2000, pp. 91–100.

[75] S. Dawson, F. Jahanian, and T. Mitton, "Orchestra: A probing and fault injection environment for testing protocol implementations," in *Proceedings of IEEE International Computer Performance and Dependability Symposium*. IEEE, 1996, p. 56.

[76] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham, "Ferrari: A tool for the validation of system dependability properties." in *FTCS*, 1992, pp. 336–344.

[77] D. Gil, J. C. Baraza, J. Gracia, and P. J. Gil, "Vhdl simulation-based fault injection techniques," in *Fault injection techniques and tools for embedded systems reliability evaluation*. Springer, 2003, pp. 159–176.

[78] V. Sieh, O. Tschache, and F. Balbach, "Verify: Evaluation of reliability using vhdl-models with embedded fault descriptions," in *Proceedings of IEEE 27th International Symposium on Fault Tolerant Computing*. IEEE, 1997, pp. 32–36.

[79] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault injection into vhdl models: the mefisto tool," in *Predictably Dependable Computing Systems*. Springer, 1995, pp. 329–346.

[80] Ó. Ruano, F. García-Herrero, L. A. Aranda, A. Sánchez-Macián, L. Rodriguez, and J. A. Maestro, "Fault injection emulation for systems in fpgas: Tools, techniques and methodology, a tutorial," *Sensors*, vol. 21, no. 4, p. 1392, 2021.

[81] M. Mauritz, "Engineering of safe autonomous vehicles through seamless integration of system development and system operation," Ph.D. dissertation, Dissertation, Clausthal-Zellerfeld, Technische Universität Clausthal, 2019, 2019.

[82] S. Samuel, L. Austin, and D. Morrey, "Automotive test drive cycles for emission measurement and real-world emission levels-a review," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 216, no. 7, pp. 555–564, 2002.

[83] E. Coelingh, J. Nilsson, and J. Buffum, "Driving tests for self-driving cars," *IEEE Spectrum*, vol. 55, no. 3, pp. 40–45, 2018.

[84] S. Khastgir, G. Dhadyalla, S. Birrell, S. Redmond, R. Addinall, and P. Jennings, "Test scenario generation for driving simulators using constrained randomization technique," SAE Technical Paper, Tech. Rep., 2017.

[85] A. Lauber, H. Guissouma, and E. Sax, "Virtual test method for complex and variant-rich automotive systems," in *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. IEEE, 2018, pp. 1–7.

[86] J.-H. Oetjens, N. Bannow, M. Becker, O. Bringmann, A. Burger, M. Chaari, S. Chakraborty, R. Drechsler, W. Ecker, K. Grüttner *et al.*, "Safety evaluation of automotive electronics using virtual prototypes: State of the art and research challenges," in *Proceedings of the 51st annual design automation conference*, 2014, pp. 1–6.

[87] S. Papagiannidis, E. See-To, and M. Bourlakis, "Virtual test-driving: The impact of simulated products on purchase intention," *Journal of Retailing and Consumer Services*, vol. 21, no. 5, pp. 877–887, 2014.

[88] J. Mina, Z. Flores, E. López, A. Pérez, and J.-H. Calleja, "Processor-in-the-loop and hardware-in-the-loop simulation of electric systems based in fpga," in *2016 13th International Conference on Power Electronics (CIEP)*. IEEE, 2016, pp. 172–177.

[89] A. Spillner and T. Linz, *Software Testing Foundations: A Study Guide for the Certified Tester Exam-Foundation Level-ISTQB® Compliant*. dpunkt. verlag, 2021.

[90] E. Bringmann and A. Krämer, "Model-based testing of automotive systems," in *2008 1st international conference on software testing, verification, and validation*. IEEE, 2008, pp. 485–493.

[91] A. R. Plummer, "Model-in-the-loop testing," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 220, no. 3, pp. 183–199, 2006.

[92] A. Bittar, H. V. Figuereido, P. A. Guimaraes, and A. C. Mendes, "Guidance software-in-the-loop simulation using x-plane and simulink for uavs," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2014, pp. 993–1002.

[93] R. Isermann, J. Schaffnit, and S. Sinsel, "Hardware-in-the-loop simulation for the design and testing of engine-control systems," *Control Engineering Practice*, vol. 7, no. 5, pp. 643–653, 1999.

[94] "dspace - targetlink," https://www.dspace.com/de/gmb/home/products/sw/pcgs/targetlink.cfm, (Accessed on 11/22/2021).

[95] "Real-time simulation and testing," https://www.speedgoat.com/, (Accessed on 01/22/2024).

[96] T. Bokc, M. Maurer, and G. Farber, "Validation of the vehicle in the loop (vil); a milestone for the simulation of driver assistance systems," in *2007 IEEE Intelligent vehicles symposium*. IEEE, 2007, pp. 612–617.

[97] B. Tabbache, M. E. H. Benbouzid, A. Kheloui, and J.-M. Bourgeot, "Virtual-sensor-based maximum-likelihood voting approach for fault-tolerant control of electric vehicle powertrains," *IEEE transactions on vehicular technology*, vol. 62, no. 3, pp. 1075–1083, 2012.

[98] G. Abaei and A. Selamat, "A survey on software fault detection based on different prediction approaches," *Vietnam Journal of Computer Science*, vol. 1, no. 2, pp. 79–95, 2014.

[99] F. Corno, F. Esposito, M. S. Reorda, and S. Tosato, "Evaluating the effects of transient faults on vehicle dynamic performance in automotive systems," in *2004 International Conferce on Test*. IEEE, 2004, pp. 1332–1339.

[100] T. Muhammed and R. A. Shaikh, "An analysis of fault detection strategies in wireless sensor networks," *Journal of Network and Computer Applications*, vol. 78, pp. 267–287, 2017.

[101] K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, and M. Srivastava, "Sensor network data fault types," *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, no. 3, pp. 1–29, 2009.

[102] S. U. Jan, Y.-D. Lee, J. Shin, and I. Koo, "Sensor fault classification based on support vector machine and statistical time-domain features," *IEEE Access*, vol. 5, pp. 8682–8690, 2017.

[103] A. Abbaspour, P. Aboutalebi, K. K. Yen, and A. Sargolzaei, "Neural adaptive observer-based sensor and actuator fault detection in nonlinear systems: Application in uav," *ISA transactions*, vol. 67, pp. 317–329, 2017.

[104] U. Saeed, S. U. Jan, Y.-D. Lee, and I. Koo, "Fault diagnosis based on extremely randomized trees in wireless sensor networks," *Reliability Engineering & System Safety*, vol. 205, p. 107284, 2021.

[105] F. Salehpour-Oskouei and M. Pourgol-Mohammad, "Risk assessment of sensor failures in a condition monitoring process; degradation-based failure probability determination," *International Journal of System Assurance Engineering and Management*, vol. 8, no. 3, pp. 584–593, 2017.

[106] J. A. Crossman, H. Guo, Y. L. Murphey, and J. Cardillo, "Automotive signal fault diagnostics-part i: signal fault analysis, signal segmentation, feature extraction and quasi-optimal feature selection," *IEEE Transactions on Vehicular Technology*, vol. 52, no. 4, pp. 1063–1075, 2003.

[107] M. Lucke, A. Stief, M. Chioua, J. R. Ottewill, and N. F. Thornhill, "Fault detection and identification combining process measurements and statistical alarms," *Control Engineering Practice*, vol. 94, p. 104195, 2020.

[108] R. Isermann, "Model-based fault-detection and diagnosis–status and applications," *Annual Reviews in control*, vol. 29, no. 1, pp. 71–85, 2005.

[109] M. Zhong, Y. Song, and S. X. Ding, "Parity space-based fault detection for linear discrete time-varying systems with unknown input," *Automatica*, vol. 59, pp. 120–126, 2015.

[110] P. M. Frank, "Enhancement of robustness in observer-based fault detection," *International Journal of control*, vol. 59, no. 4, pp. 955–981, 1994.

[111] A. K. Sood, A. A. Fahs, and N. A. Henein, "Engine fault analysis: Part ii—parameter estimation approach," *IEEE Transactions on Industrial Electronics*, no. 4, pp. 301–307, 1985.

[112] A. Scacchioli, G. Rizzoni, and P. Pisu, "Hierarchical model-based fault diagnosis for an electrical power generation storage automotive system," in *2007 American Control Conference*. IEEE, 2007, pp. 2991–2996.

[113] N. Weinhold, S. Ding, T. Jeinsch, and M. Schultalbers, "Embedded model-based fault diagnosis for on-board diagnosis of engine control systems," in *Proceedings of 2005 IEEE Conference on Control Applications, 2005. CCA 2005.* IEEE, 2005, pp. 1206–1211.

[114] D. Gonzalez-Jimenez, J. Del-Olmo, J. Poza, F. Garramiola, and P. Madina, "Data-driven fault diagnosis for electric drives: A review," *Sensors*, vol. 21, no. 12, p. 4024, 2021.

[115] Z. Gao, C. Cecati, and S. X. Ding, "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*, vol. 62, no. 6, pp. 3757–3767, 2015.

[116] X. Dai and Z. Gao, "From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2226–2238, 2013.

[117] W. Zhou, T. G. Habetler, and R. G. Harley, "Bearing fault detection via stator current noise cancellation and statistical control," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 12, pp. 4260–4269, 2008.

[118] S. Nandi and H. A. Toliyat, "Novel frequency-domain-based technique to detect stator interturn faults in induction machines using stator-induced voltages after switch-off," *IEEE Transactions on industry applications*, vol. 38, no. 1, pp. 101–109, 2002.

[119] R. Yan, R. X. Gao, and X. Chen, "Wavelets for fault diagnosis of rotary machines: A review with applications," *Signal processing*, vol. 96, pp. 1–15, 2014.

[120] G. Zhiwei, C. Cecati, and S. X. Ding, "A survey of fault diagnosis and fault-tolerant techniques—part ii: Fault diagnosis with knowledge-based and hybrid/active approaches," 2015.

[121] B. Dowdeswell, R. Sinha, and S. G. MacDonell, "Finding faults: A scoping study of fault diagnostics for industrial cyber–physical systems," *Journal of systems and software*, vol. 168, p. 110638, 2020.

[122] D. L. Nunez and M. Borsato, "An ontology-based model for prognostics and health management of machines," *Journal of Industrial Information Integration*, vol. 6, pp. 33–46, 2017.

[123] S. Yin, S. X. Ding, X. Xie, and H. Luo, "A review on basic data-driven approaches for industrial process monitoring," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 11, pp. 6418–6428, 2014.

[124] A. Theissler, J. Pérez-Velázquez, M. Kettelgerdes, and G. Elger, "Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry," *Reliability engineering & system safety*, vol. 215, p. 107864, 2021.

[125] S. W. Choi, C. Lee, J.-M. Lee, J. H. Park, and I.-B. Lee, "Fault detection and identification of nonlinear processes based on kernel pca," *Chemometrics and intelligent laboratory systems*, vol. 75, no. 1, pp. 55–67, 2005.

[126] G. R. Naik and D. K. Kumar, "An overview of independent component analysis and its applications," *Informatica*, vol. 35, no. 1, 2011.

[127] L. H. Chiang, E. L. Russell, and R. D. Braatz, "Fault diagnosis in chemical processes using fisher discriminant analysis, discriminant partial least squares, and principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 50, no. 2, pp. 243–252, 2000.

[128] Y. Lei, B. Yang, X. Jiang, F. Jia, N. Li, and A. K. Nandi, "Applications of machine learning to machine fault diagnosis: A review and roadmap," *Mechanical Systems and Signal Processing*, vol. 138, p. 106587, 2020.

[129] Y. Lei, F. Jia, J. Lin, S. Xing, and S. X. Ding, "An intelligent fault diagnosis method using unsupervised feature learning towards mechanical big data," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 5, pp. 3137–3147, 2016.

[130] Y. Reddy, P. Viswanath, and B. E. Reddy, "Semi-supervised learning: A brief review," *Int. J. Eng. Technol*, vol. 7, no. 1.8, p. 81, 2018.

[131] A. Singh, N. Thakur, and A. Sharma, "A review of supervised machine learning algorithms," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. Ieee, 2016, pp. 1310–1315.

[132] Y. Ding, L. Ma, J. Ma, M. Suo, L. Tao, Y. Cheng, and C. Lu, "Intelligent fault diagnosis for rotating machinery using deep q-network based health state classification: A deep reinforcement learning approach," *Advanced Engineering Informatics*, vol. 42, p. 100977, 2019.

[133] W. S. Noble, "What is a support vector machine?" *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[134] M. Belgiu and L. Drăguţ, "Random forest in remote sensing: A review of applications and future directions," *ISPRS journal of photogrammetry and remote sensing*, vol. 114, pp. 24–31, 2016.

[135] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[136] T. M. Kodinariya, P. R. Makwana *et al.*, "Review on determining number of cluster in k-means clustering," *International Journal*, vol. 1, no. 6, pp. 90–95, 2013.

[137] F. Lv, C. Wen, Z. Bao, and M. Liu, "Fault diagnosis based on deep learning," in *2016 American control conference (ACC)*. IEEE, 2016, pp. 6851–6856.

[138] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao, "Deep learning and its applications to machine health monitoring," *Mechanical Systems and Signal Processing*, vol. 115, pp. 213–237, 2019.

[139] S. R. Saufi, Z. A. B. Ahmad, M. S. Leong, and M. H. Lim, "Challenges and opportunities of deep learning models for machinery fault detection and diagnosis: A review," *Ieee Access*, vol. 7, pp. 122 644–122 662, 2019.

[140] S. Shafiq, A. Mashkoor, C. Mayr-Dorn, and A. Egyed, "A literature review of using machine learning in software development life cycle stages," *IEEE Access*, vol. 9, pp. 140 896–140 920, 2021.

[141] H. Badihi, Y. Zhang, B. Jiang, P. Pillay, and S. Rakheja, "A comprehensive review on signal-based and model-based condition monitoring of wind turbines: Fault diagnosis and lifetime prognosis," *Proceedings of the IEEE*, 2022.

[142] D. Neupane and J. Seok, "Bearing fault detection and diagnosis using case western reserve university dataset with deep learning approaches: A review," *IEEE Access*, vol. 8, pp. 93 155–93 178, 2020.

[143] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pp. 353–374, 2023.

[144] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096–1103.

[145] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[146] Q. Zhang, J. Zhang, J. Zou, and S. Fan, "A novel fault diagnosis method based on stacked lstm," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 790–795, 2020.

[147] B. Wang, Z. Wang, L. Liu, D. Liu, and X. Peng, "Data-driven anomaly detection for uav sensor data based on deep learning prediction model," in *2019 Prognostics and System Health Management Conference (PHM-Paris)*. IEEE, 2019, pp. 286–290.

[148] H. Pan, X. He, S. Tang, and F. Meng, "An improved bearing fault diagnosis method using one-dimensional cnn and lstm." *Strojniski Vestnik/Journal of Mechanical Engineering*, vol. 64, 2018.

[149] P. Wolf, A. Mrowca, T. T. Nguyen, B. Bäker, and S. Günnemann, "Pre-ignition detection using deep neural networks: A step towards data-driven automotive diagnostics," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 176–183.

[150] Y. Wu, Q. Xue, J. Shen, Z. Lei, Z. Chen, and Y. Liu, "State of health estimation for lithium-ion batteries based on healthy features and long short-term memory," *IEEE Access*, vol. 8, pp. 28 533–28 547, 2020.

[151] J. Zuo, H. Lv, D. Zhou, Q. Xue, L. Jin, W. Zhou, D. Yang, and C. Zhang, "Deep learning based prognostic framework towards proton exchange membrane fuel cell for automotive application," *Applied Energy*, vol. 281, p. 115937, 2021.

[152] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1d convolutional neural networks and applications: A survey," *Mechanical systems and signal processing*, vol. 151, p. 107398, 2021.

[153] D. Kolar, D. Lisjak, M. Pająk, and D. Pavković, "Fault diagnosis of rotary machines using deep convolutional neural network with wide three axis vibration signal input," *Sensors*, vol. 20, no. 14, p. 4017, 2020.

[154] O. Abdeljaber, O. Avci, S. Kiranyaz, M. Gabbouj, and D. J. Inman, "Real-time vibration-based structural damage detection using one-dimensional convolutional neural networks," *Journal of Sound and Vibration*, vol. 388, pp. 154–170, 2017.

[155] T. Ince, S. Kiranyaz, L. Eren, M. Askar, and M. Gabbouj, "Real-time motor fault detection by 1-d convolutional neural networks," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 11, pp. 7067–7075, 2016.

[156] K. B. Lee, S. Cheon, and C. O. Kim, "A convolutional neural network for fault classification and diagnosis in semiconductor manufacturing processes," *IEEE Transactions on Semiconductor Manufacturing*, vol. 30, no. 2, pp. 135–142, 2017.

[157] N. R. Storey, *Safety critical computer systems*. Addison-Wesley Longman Publishing Co., Inc., 1996.

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

[159] P. Sarhadi and S. Yousefpour, "State of the art: hardware in the loop modeling and simulation with its applications in design, development and implementation of system and control software," *International Journal of Dynamics and Control*, vol. 3, pp. 470–479, 2015.

[160] T. Schulze and J.-E. Stavesand, "Hardware-in-the-loop test process for modern e/e systems," in *Simulation and Testing for Vehicle Technology: 7th Conference, Berlin, May 12-13, 2016*. Springer, 2016, pp. 343–360.

[161] V. K. Solanki and R. Dhall, "An iot based predictive connected car maintenance approach," 2017.

[162] K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, and M. Srivastava, "Sensor network data fault types," *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, no. 3, pp. 1–29, 2009.

[163] J.-l. Yang, Y.-s. Chen, L.-l. Zhang, and Z. Sun, "Fault detection, isolation, and diagnosis of self-validating multifunctional sensors," *Review of scientific instruments*, vol. 87, no. 6, p. 065004, 2016.

[164] U. Saeed, S. U. Jan, Y.-D. Lee, and I. Koo, "Fault diagnosis based on extremely randomized trees in wireless sensor networks," *Reliability engineering & system safety*, vol. 205, p. 107284, 2021.

[165] J. A. Crossman, H. Guo, Y. L. Murphey, and J. Cardillo, "Automotive signal fault diagnostics-part i: signal fault analysis, signal segmentation, feature extraction and quasi-optimal feature selection," *IEEE Transactions on Vehicular Technology*, vol. 52, no. 4, pp. 1063–1075, 2003.

[166] "Automotive simulation models - dspace," https://www.dspace.com/en/pub/home/products/sw/automotive_simulation_models.cfm#176_26302_2, (Accessed on 11/04/2023).

[167] "Automotive simulation models - dspace," https://www.dspace.com/en/pub/home/products/products.cfm, (Accessed on 20/03/2023).

[168] D. Velásquez, E. Pérez, X. Oregui, A. Artetxe, J. Manteca, J. E. Mansilla, M. Toro, M. Maiza, and B. Sierra, "A hybrid machine-learning ensemble for anomaly detection in real-time industry 4.0 systems," *IEEE Access*, vol. 10, pp. 72 024–72 036, 2022.

[169] X. Dai and Z. Gao, "From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2226–2238, 2013.

[170] F. Zhang, N. Saeed, and P. Sadeghian, "Deep learning in fault detection and diagnosis of building hvac systems: A systematic review with meta analysis," *Energy and AI*, p. 100235, 2023.

[171] S. Qiu, X. Cui, Z. Ping, N. Shan, Z. Li, X. Bao, and X. Xu, "Deep learning techniques in intelligent fault diagnosis and prognosis for industrial systems: A review," *Sensors*, vol. 23, no. 3, p. 1305, 2023.

[172] S. F. Lokman, A. T. Othman, S. Musa, and M. H. Abu Bakar, "Deep contractive autoencoder-based anomaly detection for in-vehicle controller area network (can)," *Progress in Engineering Technology: Automotive, Energy Generation, Quality Control and Efficiency*, pp. 195–205, 2019.

[173] S. Longari, D. H. N. Valcarcel, M. Zago, M. Carminati, and S. Zanero, "Cannolo: An anomaly detection system based on lstm autoencoders for controller area network," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1913–1924, 2020.

[174] D. Rengasamy, M. Jafari, B. Rothwell, X. Chen, and G. P. Figueredo, "Deep learning with dynamically weighted loss function for sensor-based prognostics and health management," *Sensors*, vol. 20, no. 3, p. 723, 2020.

[175] J. Yang, Y. Guo, and W. Zhao, "Long short-term memory neural network based fault detection and isolation for electro-mechanical actuators," *Neurocomputing*, vol. 360, pp. 85–96, 2019.

[176] T. Zehelein, T. Hemmert-Pottmann, and M. Lienkamp, "Diagnosing automotive damper defects using convolutional neural networks and electronic stability control sensor signals," *Journal of Sensor and Actuator Networks*, vol. 9, no. 1, p. 8, 2020.

[177] A. Kumar, C. Gandhi, Y. Zhou, G. Vashishtha, R. Kumar, and J. Xiang, "Improved cnn for the diagnosis of engine defects of 2-wheeler vehicle using wavelet synchro-squeezed transform (wsst)," *Knowledge-Based Systems*, vol. 208, p. 106453, 2020.

[178] E. Min, X. Guo, Q. Liu, G. Zhang, J. Cui, and J. Long, "A survey of clustering with deep learning: From the perspective of network architecture," *IEEE Access*, vol. 6, pp. 39 501–39 514, 2018.

[179] S. Zhang, S. Zhang, B. Wang, and T. G. Habetler, "Deep learning algorithms for bearing fault diagnostics—a comprehensive review," *IEEE Access*, vol. 8, pp. 29 857–29 881, 2020.

[180] T. Zhang, J. Chen, F. Li, K. Zhang, H. Lv, S. He, and E. Xu, "Intelligent fault diagnosis of machines with small & imbalanced data: A state-of-the-art review and possible extensions," *ISA transactions*, vol. 119, pp. 152–171, 2022.

[181] A. Theissler, J. Pérez-Velázquez, M. Kettelgerdes, and G. Elger, "Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry," *Reliability engineering & system safety*, vol. 215, p. 107864, 2021.

[182] V. K. Kukkala, J. Tunnell, S. Pasricha, and T. Bradley, "Advanced driver-assistance systems: A path toward autonomous vehicles," *IEEE Consumer Electronics Magazine*, vol. 7, no. 5, pp. 18–25, 2018.

[183] M. Fernandes, J. M. Corchado, and G. Marreiros, "Machine learning techniques applied to mechanical fault diagnosis and fault prognosis in the context of real industrial manufacturing use-cases: a systematic literature review," *Applied Intelligence*, vol. 52, no. 12, pp. 14 246–14 280, 2022.

[184] R. Svenningsson, H. Eriksson, J. Vinter, and M. Törngren, "Model-implemented fault injection for hardware fault simulation," in *2010 Workshop on Model-Driven Engineering, Verification, and Validation*. IEEE, 2010, pp. 31–36.

[185] M. Moradi, B. Van Acker, K. Vanherpen, and J. Denil, "Model-implemented hybrid fault injection for simulink (tool demonstrations)," in *Cyber Physical Systems. Model-Based Design*. Springer, 2018, pp. 71–90.

[186] S. Mohan, C. Thirumalai, and G. Srivastava, "Effective heart disease prediction using hybrid machine learning techniques," *IEEE access*, vol. 7, pp. 81 542–81 554, 2019.

[187] T. Shon and J. Moon, "A hybrid machine learning approach to network anomaly detection," *Information Sciences*, vol. 177, no. 18, pp. 3799–3821, 2007.

[188] Y. Shen and K. Khorasani, "Hybrid multi-mode machine learning-based fault diagnosis strategies with application to aircraft gas turbine engines," *Neural Networks*, vol. 130, pp. 126–142, 2020.

[189] S. Winter, M. Tretter, B. Sattler, and N. Suri, "simfi: From single to simultaneous software fault injections," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2013, pp. 1–12.

[190] P. Peng, W. Zhang, Y. Zhang, H. Wang, and H. Zhang, "Non-revisiting genetic cost-sensitive sparse autoencoder for imbalanced fault diagnosis," *Applied Soft Computing*, vol. 114, p. 108138, 2022.

[191] P. Micouin, *Model Based Systems Engineering: Fundamentals and Methods*. John Wiley & Sons, 2014.

[192] R. Svenningsson, J. Vinter, H. Eriksson, and M. Törngren, "Modifi: a model-implemented fault injection tool," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2010, pp. 210–222.

[193] J. Vinter, L. Bromander, P. Raistrick, and H. Edler, "Fiscade-a fault injection tool for scade models," in *2007 3rd Institution of Engineering and Technology Conference on Automotive Electronics*. IET, 2007, pp. 1–9.

[194] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Increasing efficiency of iso 26262 verification and validation by combining fault injection and mutation testing with model based development," in *International Conference on Software Engineering and Applications*, vol. 2. SCITEPRESS, 2013, pp. 251–257.

[195] P. Folkesson, F. Ayatolahi, B. Sangchoolie, J. Vinter, M. Islam, and J. Karlsson, "Back-to-back fault injection testing in model-based development," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2014, pp. 135–148.

[196] M. Saraoglu, A. Morozov, and K. Janschek, "Mobatsim: Model-based autonomous traffic simulation framework for fault-error-failure chain analysis," *IFAC-PapersOnLine*, vol. 52, no. 8, pp. 239–244, 2019.

[197] G. Juez, E. Amparan, R. Lattarulo, A. Ruíz, J. Pérez, and H. Espinoza, "Early safety assessment of automotive systems using sabotage simulation-based fault injection framework," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2017, pp. 255–269.

[198] S. Jha, T. Tsai, S. Hari, M. Sullivan, Z. Kalbarczyk, S. W. Keckler, and R. K. Iyer, "Kayotee: A fault injection-based system to assess the safety and reliability of autonomous vehicles to faults and errors," *arXiv preprint arXiv:1907.01024*, 2019.

[199] M. Saraoğlu, A. Morozov, M. T. Söylemez, and K. Janschek, "Errorsim: A tool for error propagation analysis of simulink models," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2017, pp. 245–254.

[200] S. Jha, S. S. Banerjee, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "Avfi: Fault injection for autonomous vehicles," in *2018 48th annual ieee/ifip international conference on dependable systems and networks workshops (dsn-w)*. IEEE, 2018, pp. 55–56.

[201] A. M. Silveira, R. E. Araújo, and R. de Castro, "Fieev: a co-simulation framework for fault injection in electrical vehicles," in *2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2012)*. IEEE, 2012, pp. 357–362.

[202] I. Pill, I. Rubil, F. Wotawa, and M. Nica, "Simultate: A toolset for fault injection and mutation testing of simulink models," in *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2016, pp. 168–173.

[203] A. Palladino, G. Fiengo, and D. Lanzo, "A portable hardware-in-the-loop (hil) device for automotive diagnostic control systems," *ISA transactions*, vol. 51, no. 1, pp. 229–236, 2012.

[204] O. Gietelink, J. Ploeg, B. De Schutter, and M. Verhaegen, "Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations," *Vehicle System Dynamics*, vol. 44, no. 7, pp. 569–590, 2006.

[205] R. Conti, E. Meli, A. Ridolfi, and A. Rindi, "An innovative hardware in the loop architecture for the analysis of railway braking under degraded adhesion conditions through roller-rigs," *Mechatronics*, vol. 24, no. 2, pp. 139–150, 2014.

[206] J. J. Poon, M. A. Kinsy, N. A. Pallo, S. Devadas, and I. L. Celanovic, "Hardware-in-the-loop testing for electric vehicle drive applications," in *2012 Twenty-Seventh Annual IEEE Applied Power Electronics Conference and Exposition (APEC)*. IEEE, 2012, pp. 2576–2582.

[207] X. Yang, C. Yang, T. Peng, Z. Chen, B. Liu, and W. Gui, "Hardware-in-the-loop fault injection for traction control system," *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 6, no. 2, pp. 696–706, 2018.

[208] F. Garramiola, J. Poza, J. del Olmo, and T. Nieva, "Hardware-in-the-loop performance analysis of a railway traction system under sensor faults," *The Journal of Engineering*, vol. 2019, no. 17, pp. 3797–3801, 2019.

[209] M. Elgharbawy, A. Schwarzhaupt, M. Frey, and F. Gauterin, "A real-time multisensor fusion verification framework for advanced driver assistance systems," *Transportation research part F: traffic psychology and behaviour*, vol. 61, pp. 259–267, 2019.

[210] X. Zhang, K. Han, H. Cao, Z. Wang, and K. Huo, "Fault injection model of induction motor for stator interturn fault diagnosis research based on hils," *World Electric Vehicle Journal*, vol. 12, no. 4, p. 170, 2021.

[211] F. Garramiola, J. Poza, P. Madina, J. Del Olmo, and G. Ugalde, "A hybrid sensor fault diagnosis for maintenance in railway traction drives," *Sensors*, vol. 20, no. 4, p. 962, 2020.

[212] Y. Fu, A. Terechko, T. Bijlsma, P. J. Cuijpers, J. Redegeld, and A. O. Örs, "A retargetable fault injection framework for safety validation of autonomous vehicles," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2019, pp. 69–76.

[213] J. Park and B. Choi, "Asfit: Autosar-based software fault injection test for vehicles," *Electronics*, vol. 9, no. 5, p. 850, 2020.

[214] S. Safavi, M. A. Safavi, H. Hamid, and S. Fallah, "Multi-sensor fault detection, identification, isolation and health forecasting for autonomous vehicles," *Sensors*, vol. 21, no. 7, p. 2547, 2021.

[215] D. Jung, "Data-driven open-set fault classification of residual data using bayesian filtering," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 5, pp. 2045–2052, 2020.

[216] L. Yang, J. Yu, Y. Guo, S. Chen, K. Tan, and S. Li, "An electrode-grounded droplet-based electricity generator (eg-deg) for liquid motion monitoring," *Advanced Functional Materials*, p. 2302147, 2023.

[217] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, and A. Anwar, "Ton_iot telemetry dataset: A new generation dataset of iot and iiot for data-driven intrusion detection systems," *Ieee Access*, vol. 8, pp. 165 130–165 150, 2020.

[218] A. Mallak and M. Fathi, "Sensor and component fault detection and diagnosis for hydraulic machinery integrating lstm autoencoder detector and diagnostic classifiers," *Sensors*, vol. 21, no. 2, p. 433, 2021.

[219] H. H. Bafroui and A. Ohadi, "Application of wavelet energy and shannon entropy for feature extraction in gearbox fault detection under varying speed conditions," *Neurocomputing*, vol. 133, pp. 437–445, 2014.

[220] T. Tagawa, Y. Tadokoro, and T. Yairi, "Structured denoising autoencoder for fault detection and analysis," in *Asian conference on machine learning*. PMLR, 2015, pp. 96–111.

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

[222] J. Jang, H. Lee, and J.-C. Kim, "Carfree: Hassle-free object detection dataset generation using carla autonomous driving simulator," *Applied Sciences*, vol. 12, no. 1, p. 281, 2021.

[223] G. D'Amico, M. Marinoni, F. Nesti, G. Rossolini, G. Buttazzo, S. Sabina, and G. Lauro, "Trainsim: A railway simulation framework for lidar and camera dataset generation," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[224] D. Gonzalez-Jimenez, J. Del-Olmo, J. Poza, F. Garramiola, and P. Madina, "Data-driven fault diagnosis for electric drives: A review," *Sensors*, vol. 21, no. 12, p. 4024, 2021.

[225] R. Raveendran, K. Devika, and S. C. Subramanian, "Brake fault identification and fault-tolerant directional stability control of heavy road vehicles," *IEEE Access*, vol. 8, pp. 169 229–169 246, 2020.

[226] L. Guo, R. Li, and B. Jiang, "Fault detection and diagnosis using statistic feature and improved broad learning for traction systems in high-speed trains," *IEEE Transactions on Artificial Intelligence*, 2022.

[227] F. Garramiola, J. Del Olmo, J. Poza, P. Madina, and G. Almandoz, "Integral sensor fault detection and isolation for railway traction drive," *Sensors*, vol. 18, no. 5, p. 1543, 2018.

[228] F. Mihalič, M. Truntič, and A. Hren, "Hardware-in-the-loop simulations: A historical overview of engineering challenges," *Electronics*, vol. 11, no. 15, p. 2462, 2022.

[229] S. Safavi, M. A. Safavi, H. Hamid, and S. Fallah, "Multi-sensor fault detection, identification, isolation and health forecasting for autonomous vehicles," *Sensors*, vol. 21, no. 7, p. 2547, 2021.

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

[231] D. Jung, "Data-driven open-set fault classification of residual data using bayesian filtering," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 5, pp. 2045–2052, 2020.

[232] H. Kaplan, K. Tehrani, and M. Jamshidi, "A fault diagnosis design based on deep learning approach for electric vehicle applications," *Energies*, vol. 14, no. 20, p. 6599, 2021.

[233] J.-H. Zhong, P. K. Wong, and Z.-X. Yang, "Fault diagnosis of rotating machinery based on multiple probabilistic classifiers," *Mechanical Systems and Signal Processing*, vol. 108, pp. 99–114, 2018.

[234] L. Biddle and S. Fallah, "A novel fault detection, identification and prediction approach for autonomous vehicle controllers using svm," *Automotive Innovation*, vol. 4, no. 3, pp. 301–314, 2021.

[235] P. Sarhadi and S. Yousefpour, "State of the art: hardware in the loop modeling and simulation with its applications in design, development and implementation of system and control software," *International Journal of Dynamics and Control*, vol. 3, no. 4, pp. 470–479, 2015.

[236] F. Garramiola, J. Del Olmo, J. Poza, P. Madina, and G. Almandoz, "Integral sensor fault detection and isolation for railway traction drive," *Sensors*, vol. 18, no. 5, p. 1543, 2018.

[237] R. Raveendran, K. Devika, and S. C. Subramanian, "Brake fault identification and fault-tolerant directional stability control of heavy road vehicles," *IEEE Access*, vol. 8, pp. 169 229–169 246, 2020.

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

[239] H. Shao, H. Jiang, H. Zhao, and F. Wang, "A novel deep autoencoder feature learning method for rotating machinery fault diagnosis," *Mechanical Systems and Signal Processing*, vol. 95, pp. 187–204, 2017.

[240] J. Qian, Z. Song, Y. Yao, Z. Zhu, and X. Zhang, "A review on autoencoder based representation learning for fault detection and diagnosis in industrial processes," *Chemometrics and Intelligent Laboratory Systems*, p. 104711, 2022.

[241] Q. Wang, Y. Yu, H. O. Ahmed, M. Darwish, and A. K. Nandi, "Fault detection and classification in mmc-hvdc systems using learning methods," *Sensors*, vol. 20, no. 16, p. 4438, 2020.

[242] T. Han and Y.-F. Li, "Out-of-distribution detection-assisted trustworthy machinery fault diagnosis approach with uncertainty-aware deep ensembles," *Reliability Engineering & System Safety*, vol. 226, p. 108648, 2022.

[243] C. H. Park, H. Kim, C. Suh, M. Chae, H. Yoon, and B. D. Youn, "A health image for deep learning-based fault diagnosis of a permanent magnet synchronous motor under variable operating conditions: Instantaneous current residual map," *Reliability Engineering & System Safety*, vol. 226, p. 108715, 2022.

[244] P. K. Wong, J. Zhong, Z. Yang, and C. M. Vong, "Sparse bayesian extreme learning committee machine for engine simultaneous fault diagnosis," *Neurocomputing*, vol. 174, pp. 331–343, 2016.

[245] L. Chang, X. Xu, Z.-g. Liu, B. Qian, X. Xu, and Y.-W. Chen, "Brb prediction with customized attributes weights and tradeoff analysis for concurrent fault diagnosis," *IEEE Systems Journal*, vol. 15, no. 1, pp. 1179–1190, 2020.

[246] B. Wu, W. Cai, H. Chen, and X. Zhang, "A hybrid data-driven simultaneous fault diagnosis model for air handling units," *Energy and Buildings*, vol. 245, p. 111069, 2021.

[247] Z. Zhang, S. Li, Y. Xiao, and Y. Yang, "Intelligent simultaneous fault diagnosis for solid oxide fuel cell system based on deep learning," *Applied Energy*, vol. 233, pp. 930–942, 2019.

[248] P. Liang, C. Deng, J. Wu, Z. Yang, J. Zhu, and Z. Zhang, "Single and simultaneous fault diagnosis of gearbox via a semi-supervised and high-accuracy adversarial learning framework," *Knowledge-Based Systems*, vol. 198, p. 105895, 2020.

[249] A. Namigtle-Jiménez, R. Escobar-Jiménez, J. Gómez-Aguilar, C. García-Beltrán, and A. Téllez-Anguiano, "Online ann-based fault diagnosis implementation using an fpga: Application in the efi system of a vehicle," *ISA transactions*, vol. 100, pp. 358–372, 2020.

[250] H. Kaplan, K. Tehrani, and M. Jamshidi, "A fault diagnosis design based on deep learning approach for electric vehicle applications," *Energies*, vol. 14, no. 20, p. 6599, 2021.

[251] Y. Chen, M. Rao, K. Feng, and G. Niu, "Modified varying index coefficient autoregression model for representation of the nonstationary vibration from a planetary gearbox," *IEEE Transactions on Instrumentation and Measurement*, vol. 72, pp. 1–12, 2023.

[252] Y. Chen, G. Niu, Y. Li, and Y. Li, "A modified bidirectional long short-term memory neural network for rail vehicle suspension fault detection," *Vehicle System Dynamics*, pp. 1–25, 2022.

[253] S. Scharoba, K.-U. Basener, J. Bielefeldt, H.-W. Wiesbrock, and M. Hübner, "Towards machine learning support for embedded system tests," in *2021 24th Euromicro Conference on Digital System Design (DSD)*. IEEE, 2021, pp. 166–173.

[254] S. Asgari, R. Gupta, I. K. Puri, and R. Zheng, "A data-driven approach to simultaneous fault detection and diagnosis in data centers," *Applied Soft Computing*, vol. 110, p. 107638, 2021.

[255] S. Li, H. Cao, and Y. Yang, "Data-driven simultaneous fault diagnosis for solid oxide fuel cell system using multi-label pattern identification," *Journal of Power Sources*, vol. 378, pp. 646–659, 2018.

[256] A. Rodríguez Ramos, C. Domínguez Acosta, P. J. Rivera Torres, E. I. Serrano Mercado, G. Beauchamp Baez, L. A. Rifón, and O. Llanes-Santiago, "An approach to multiple fault diagnosis using fuzzy logic," *Journal of Intelligent Manufacturing*, vol. 30, pp. 429–439, 2019.

[257] A. Othman, N. Iqbal, S. M. Hanafy, and U. B. Waheed, "Automated event detection and denoising method for passive seismic data using residual deep convolutional neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–11, 2021.

[258] R. A. Wynne, V. Beanland, and P. M. Salmon, "Systematic review of driving simulator validation studies," *Safety science*, vol. 117, pp. 138–151, 2019.

[259] L. Bruck, B. Haycock, and A. Emadi, "A review of driving simulation technology and applications," *IEEE Open Journal of Vehicular Technology*, vol. 2, pp. 1–16, 2020.

[260] G. Xu, P. Diao, X. He, J. Wu, G. Wang, and C. Wang, "Research on vehicle active steering control based on linear matrix inequality and hardware in the loop test scheme design and implement for active steering," *Advances in Mechanical Engineering*, vol. 11, no. 11, p. 1687814019892108, 2019.

[261] D. H. Weir, "Application of a driving simulator to the development of in-vehicle human–machine-interfaces," *IATSS research*, vol. 34, no. 1, pp. 16–21, 2010.

[262] C. Galko, R. Rossi, and X. Savatier, "Vehicle-hardware-in-the-loop system for adas prototyping and validation," in *2014 International conference on embedded computer systems: Architectures, Modeling, and Simulation (SAMOS XIV)*. IEEE, 2014, pp. 329–334.

[263] F. A. Schiegg, J. Krost, S. Jesenski, and J. Frye, "A novel simulation framework for the design and testing of advanced driver assistance systems," in *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*. IEEE, 2019, pp. 1–6.

[264] M. Saraoğlu, A. Morozov, and K. Janschek, "Safety assessment of autonomous and connected vehicles by a model-based traffic simulation framework," in *19. Internationales Stuttgarter Symposium: Automobil-und Motorentechnik*. Springer, 2019, pp. 988–1002.

[265] R. Lattarulo, J. Pérez, and M. Dendaluce, "A complete framework for developing and testing automated driving controllers," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 258–263, 2017.

[266] G. Sievers, C. Seiger, M. Peperhowe, H. Krumm, S. Graf, and H. Hanselmann, "Driving simulation technologies for sensor simulation in sil and hil environments," in *Proceedings of the DSC*, 2018, pp. 127–130.

[267] I. Karl, G. Berg, F. Ruger, and B. Farber, "Driving behavior and simulator sickness while driving the vehicle in the loop: Validation of longitudinal driving behavior," *IEEE intelligent transportation systems magazine*, vol. 5, no. 1, pp. 42–57, 2013.

[268] C. Park, S. Chung, and H. Lee, "Vehicle-in-the-loop in global coordinates for advanced driver assistance system," *Applied Sciences*, vol. 10, no. 8, p. 2645, 2020.

[269] S. Solmaz, M. Rudigier, and M. Mischinger, "A vehicle-in-the-loop methodology for evaluating automated driving functions in virtual traffic," in *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020, pp. 1465–1471.

[270] A. Soltani and F. Assadian, "A hardware-in-the-loop facility for integrated vehicle dynamics control system design and validation," *Ifac-Papersonline*, vol. 49, no. 21, pp. 32–38, 2016.

[271] M. Lee, H. Lee, K. S. Lee, S. Ha, J. Bae, J. Park, H. Park, H. Choi, and H. Chun, "Development of a hardware in the loop simulation system for electric power steering in vehicles," *International journal of Automotive technology*, vol. 12, pp. 733–744, 2011.

[272] H. Eom and S. H. Lee, "Human-automation interaction design for adaptive cruise control systems of ground vehicles," *Sensors*, vol. 15, no. 6, pp. 13 916–13 944, 2015.

[273] T. Vo-Duy, M. C. Ta, B.-H. Nguyn, and J. P. F. Trovão, "Experimental platform for evaluation of on-board real-time motion controllers for electric vehicles," *Energies*, vol. 13, no. 23, p. 6448, 2020.

[274] A. Tumasov, A. Vashurin, Y. P. Trusov, E. Toropov, P. Moshkov, V. Kryaskov, and A. Vasilyev, "The application of hardware-in-the-loop (hil) simulation for evaluation of active safety of vehicles equipped with electronic stability control (esc) systems," *Procedia computer science*, vol. 150, pp. 309–315, 2019.

[275] G. Lee, S. Ha, and J.-i. Jung, "Integrating driving hardware-in-the-loop simulator with large-scale vanet simulator for evaluation of cooperative eco-driving system," *Electronics*, vol. 9, no. 10, p. 1645, 2020.

[276] L. Pintard, J.-C. Fabre, K. Kanoun, M. Leeman, and M. Roy, "Fault injection in the automotive standard iso 26262: an initial approach," in *Dependable Computing: 14th European Workshop, EWDC 2013, Coimbra, Portugal, May 15-16, 2013. Proceedings*. Springer, 2013, pp. 126–133.

[277] M. Abboush, D. Bamal, C. Knieke, and A. Rausch, "Hardware-in-the-loop-based real-time fault injection framework for dynamic behavior analysis of automotive software systems," *Sensors*, vol. 22, no. 4, p. 1360, 2022.

[278] M. Abboush, C. Knieke, and A. Rausch, "Representative dataset generation framework for ai-based failure analysis during real-time validation of automotive software systems," pp. 7312–7322, 2024.

[279] M. Abboush, D. Bamal, C. Knieke, and A. Rausch, "Intelligent fault detection and classification based on hybrid deep learning methods for hardware-in-the-loop test of automotive software systems," *Sensors*, vol. 22, no. 11, p. 4066, 2022.

[280] B. Xue, M. Zhang, W. N. Browne, and X. Yao, "A survey on evolutionary computation approaches to feature selection," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 4, pp. 606–626, 2015.

[281] U. R. Acharya, H. Fujita, O. S. Lih, Y. Hagiwara, J. H. Tan, and M. Adam, "Automated detection of arrhythmias using different intervals of tachycardia ecg segments with convolutional neural network," *Information sciences*, vol. 405, pp. 81–90, 2017.

[282] J. Jiao, M. Zhao, J. Lin, and K. Liang, "A comprehensive review on convolutional neural network in machine fault diagnosis," *Neurocomputing*, vol. 417, pp. 36–63, 2020.

[283] A. Shenfield and M. Howarth, "A novel deep learning model for the detection and identification of rolling element-bearing faults," *Sensors*, vol. 20, no. 18, p. 5112, 2020.

[284] P. T. Yamak, L. Yujian, and P. K. Gadosey, "A comparison between arima, lstm, and gru for time series forecasting," in *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence*, 2019, pp. 49–55.

[285] H. Zhao, S. Sun, and B. Jin, "Sequential fault diagnosis based on lstm neural network," *Ieee Access*, vol. 6, pp. 12 929–12 939, 2018.

[286] P. Park, P. D. Marco, H. Shin, and J. Bang, "Fault detection and diagnosis using combined autoencoder and long short-term memory network," *Sensors*, vol. 19, no. 21, p. 4612, 2019.

[287] J. Chen, W. Hu, D. Cao, B. Zhang, Q. Huang, Z. Chen, and F. Blaabjerg, "An imbalance fault detection algorithm for variable-speed wind turbines: A deep learning approach," *Energies*, vol. 12, no. 14, p. 2764, 2019.

[288] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[289] W. You, C. Shen, D. Wang, L. Chen, X. Jiang, and Z. Zhu, "An intelligent deep feature learning method with improved activation functions for machine fault diagnosis," *IEEE Access*, vol. 8, pp. 1975–1985, 2019.

[290] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for lvcsr using rectified linear units and dropout," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 8609–8613.

[291] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," *arXiv preprint arXiv:2010.16061*, 2020.

[292] P. Koopman, "A case study of toyota unintended acceleration and software safety," *Presentation. Sept*, 2014.

[293] "tf.keras.utils.normalize | tensorflow core v2.7.0," https://www.tensorflow.org/api_docs/python/tf/keras/utils/normalize, (Accessed on 12/12/2021).

[294] "Tensorflow," https://www.tensorflow.org/, (Accessed on 12/12/2021).

[295] "Keras: the python deep learning api," https://keras.io/, (Accessed on 12/12/2021).

[296] "sklearn.metrics.accuracy_score — scikit-learn 1.0.1 documentation," https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html, (Accessed on 12/12/2021).

[297] M. Abboush, C. Knieke, and A. Rausch, "Gru-based denoising autoencoder for detection and clustering of unknown single and concurrent faults during system integration testing of automotive software systems," *Sensors*, vol. 23, no. 14, p. 6606, 2023.

[298] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[299] N. R. Goodman, "Statistical analysis based on a certain multivariate complex gaussian distribution (an introduction)," *The Annals of mathematical statistics*, vol. 34, no. 1, pp. 152–177, 1963.

[300] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[301] G. Dong, G. Liao, H. Liu, and G. Kuang, "A review of the autoencoder and its variants: A comparative perspective from target recognition in synthetic-aperture radar images," *IEEE Geoscience and Remote Sensing Magazine*, vol. 6, no. 3, pp. 44–68, 2018.

[302] J. Cowton, I. Kyriazakis, T. Plötz, and J. Bacardit, "A combined deep learning gru-autoencoder for the early detection of respiratory disease in pigs using multiple environmental sensors," *Sensors*, vol. 18, no. 8, p. 2521, 2018.

[303] Q. Liu, T. Liang, and V. Dinavahi, "Real-time hierarchical neural network based fault detection and isolation for high-speed railway system under hybrid ac/dc grid," *IEEE Transactions on Power Delivery*, vol. 35, no. 6, pp. 2853–2864, 2020.

[304] M. Ahmed, R. Seraj, and S. M. S. Islam, "The k-means algorithm: A comprehensive survey and performance evaluation," *Electronics*, vol. 9, no. 8, p. 1295, 2020.

[305] S. U. Jan, Y.-D. Lee, J. Shin, and I. Koo, "Sensor fault classification based on support vector machine and statistical time-domain features," *IEEE Access*, vol. 5, pp. 8682–8690, 2017.

[306] J. Zabalza, J. Ren, J. Zheng, H. Zhao, C. Qing, Z. Yang, P. Du, and S. Marshall, "Novel segmented stacked autoencoder for effective dimensionality reduction and feature extraction in hyperspectral imaging," *Neurocomputing*, vol. 185, pp. 1–10, 2016.

[307] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE transactions on pattern analysis and machine intelligence*, no. 2, pp. 224–227, 1979.

[308] M. Abboush, C. Knieke, and A. Rausch, "Intelligent identification of simultaneous faults of automotive software systems under noisy and imbalanced data based on ensemble lstm and random forest," *IEEE Access*, 2023.

[309] T. Muhammed and R. A. Shaikh, "An analysis of fault detection strategies in wireless sensor networks," *Journal of Network and Computer Applications*, vol. 78, pp. 267–287, 2017.

[310] B. Xue, M. Zhang, W. N. Browne, and X. Yao, "A survey on evolutionary computation approaches to feature selection," *IEEE Transactions on evolutionary computation*, vol. 20, no. 4, pp. 606–626, 2015.

[311] C.-M. Vong, P.-K. Wong, and W.-F. Ip, "A new framework of simultaneous-fault diagnosis using pairwise probabilistic multi-label classification for time-dependent patterns," *IEEE transactions on industrial electronics*, vol. 60, no. 8, pp. 3372–3385, 2012.

[312] S. Patro and K. K. Sahu, "Normalization: A preprocessing stage," *arXiv preprint arXiv:1503.06462*, 2015.

[313] P. Bedi, N. Gupta, and V. Jindal, "I-siamids: an improved siam-ids for handling class imbalance in network-based intrusion detection systems," *Applied Intelligence*, vol. 51, pp. 1133–1151, 2021.

[314] T. Hasanin and T. Khoshgoftaar, "The effects of random undersampling with simulated class imbalance for big data," in *2018 IEEE international conference on information reuse and integration (IRI)*. IEEE, 2018, pp. 70–79.

[315] Z. Ren, T. Lin, K. Feng, Y. Zhu, Z. Liu, and K. Yan, "A systematic review on imbalanced learning methods in intelligent fault diagnosis," *IEEE Transactions on Instrumentation and Measurement*, 2023.

[316] X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma, "A survey on ensemble learning," *Frontiers of Computer Science*, vol. 14, pp. 241–258, 2020.

[317] S. Ma, M. Chen, J. Wu, Y. Wang, B. Jia, and Y. Jiang, "Intelligent fault diagnosis of hvcb with feature space optimization-based random forest," *Sensors*, vol. 18, no. 4, p. 1221, 2018.

[318] M. A. Marins, B. D. Barros, I. H. Santos, D. C. Barrionuevo, R. E. Vargas, T. d. M. Prego, A. A. de Lima, M. L. de Campos, E. A. da Silva, and S. L. Netto, "Fault detection and classification in oil wells and production/service lines using random forest," *Journal of Petroleum Science and Engineering*, vol. 197, p. 107879, 2021.

[319] D. Ruta and B. Gabrys, "Classifier selection for majority voting," *Information fusion*, vol. 6, no. 1, pp. 63–81, 2005.

[320] P. Koopman, "A case study of toyota unintended acceleration and software safety," *Pittsburgh, PA, USA*, p. 3–6, September 2014.

[321] Y. Wu, W. Jin, Y. Li, and D. Wang, "A novel method for simultaneous-fault diagnosis based on between-class learning," *Measurement*, vol. 172, p. 108839, 2021.

[322] X. Ma, Y. Hu, M. Wang, F. Li, and Y. Wang, "Degradation state partition and compound fault diagnosis of rolling bearing based on personalized multilabel learning," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–11, 2021.

[323] M. Abboush, C. Knieke, and A. Rausch, "A virtual testing framework for real-time validation of automotive software systems based on hardware in the loop and fault injection," *Sensors*, vol. 24, no. 12, p. 3733, 2024.

[324] J. Zhou, R. Schmied, A. Sandalek, H. Kokal, and L. del Re, "A framework for virtual testing of adas," *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, vol. 9, no. 2016-01-0049, pp. 66–73, 2016.

[325] M. H. Shojaeefard, M. Mollajafari, S. Ebrahimi-Nejad, and S. Tayebi, "Weather-aware fuzzy adaptive cruise control: Dynamic reference signal design," *Computers and Electrical Engineering*, vol. 110, p. 108903, 2023.

[326] A. Amyan, M. Abboush, C. Knieke, and A. Rausch, "Automating fault test cases generation and execution for automotive safety validation via nlp and hil simulation," *Sensors*, vol. 24, no. 10, p. 3145, 2024.