electronics

MDPI

Article

Representative Real-Time Dataset Generation Based on Automated Fault Injection and HIL Simulation for ML-Assisted Validation of Automotive Software Systems

Mohammad Abboush *, Christoph Knieke and Andreas Rausch

Institute for Software and Systems Engineering, Technische Universität Clausthal, 38678 Clausthal-Zellerfeld, Germany; christoph.knieke@tu-clausthal.de (C.K.); andreas.rausch@tu-clausthal.de (A.R.)

* Correspondence: mohammad.abboush@tu-clausthal.de

Abstract: Recently, a data-driven approach has been widely used at various stages of the system development lifecycle thanks to its ability to extract knowledge from historical data. However, despite its superiority over other conventional approaches, e.g., approaches that are model-based and signal-based, the availability of representative datasets poses a major challenge. Therefore, for various engineering applications, new solutions to generate representative faulty data that reflect the real world operating conditions should be explored. In this study, a novel approach based on a hardware-in-the-loop (HIL) simulation and automated real-time fault injection (FI) method is proposed to generate, analyse and collect data samples in the presence of single and concurrent faults. The generated dataset is employed for the development of machine learning (ML)-assisted test strategies during the system verification and validation phases of the V-cycle development model. The developed framework can generate not only time series data but also a textual data including fault logs in an automated manner. As a case study, a high-fidelity simulation model of a gasoline engine system with a dynamic entire vehicle model is utilised to demonstrate the capabilities and benefits of the proposed framework. The results reveal the applicability of the proposed framework in simulating and capturing the system behaviour in the presence of faults occurring within the system's components. Furthermore, the effectiveness of the proposed framework in analysing system behaviour and acquiring data during the validation phase of real-time systems under realistic operating conditions has been demonstrated.

Keywords: HIL testing; real-time validation; fault injection; automotive software systems; dataset generation; machine learning

Citation: 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**, *13*, 437. https://doi.org/10.3390/electronics13020437

Academic Editor: Dimitra I. Kaklamani

Received: 13 November 2023
Revised: 14 January 2024
Accepted: 17 January 2024
Published: 20 January 2024

1. Introduction

This article is an extension of our published paper, in which a framework for representative dataset generation of automotive software systems based on a manual fault injection process was proposed [1]. Recently, the advanced functionalities of modern automotive software systems have played a crucial role in reshaping our future mobility, e.g., ADAS [2]. However, along with the rapid development of software-driven systems, major challenges in testing and validation have also emerged [3,4]. This is because of the ever-expanding system architecture, with several hundred ECUs connected to the system via multiple system buses [5,6]. Consequently, with such a complex system containing many millions of lines of code, the probability of a fault is very high [7]. To ensure an acceptable level of safety and reliability, the functional safety standard ISO 26262 [8] has been introduced, which defines the requirements and recommendations for the development process.

In accordance with model-based design and the V-model development approach, the System Under Test (SUT) is tested at each level of the development process. In other words, the testing process is performed for different states of the SUT, i.e., the executable

model, the model code and the implemented code on the target machine [9]. Several platforms have been leveraged to carry out the testing and validation activities known as X-in-the-loop [10]. Among them, Hardware-in-the-Loop (HIL) real-time simulation is recommended by ISO 26262 as a safe, flexible, reliable, and effective platform [11]. Recently, HIL has played a vital role in the verification and validation of automotive-embedded control systems under real-time conditions. Replacing the real hardware elements with a realistically simulated system not only eliminates potential risks, but also reduces test costs for various applications, e.g., electric drives, power electronics, power grids, railways and automotives [12]. The HIL-based system validation is currently a hot topic in academia and industry due to its ability to provide efficient, fast, and realistic real-time simulations with high accuracy. On top of that, by serving as a digital test drive platform, HIL has contributed to overcoming the limitations of real test drives in terms of time, cost, effort and risk to the tester [13]. However, at the system integration testing level, an enormous amount of datasets from heterogeneous components and subsystems are generated as a result of tests' execution [14]. Moreover, despite the test automation of the SUT, the execution results of the Test Cases (TCs) are documented as pass/fail in the test report [15]. Therefore, analysing the vast amount of test records based on traditional approaches is costly, difficult, and time consuming [16]. This is why an intelligent system capable of analyzing the test results of HIL in an efficient way without domain knowledge is necessary.

Over the last decade, advances in computing resources have paved the way for the widespread application of a data-driven approach to tackle the problem of Fault Detection and Diagnosis (FDD). Central to this approach is the idea of extracting knowledge from historical data and constructing non-linear relationships between input and output classes to discover hidden patterns. Unlike other FDD approaches, neither expert knowledge nor a precise mathematical model is required. Hence, the data-driven approach has attracted the attention of researchers, and the development of FDD methods has increased rapidly in various technical fields. Deep Learning (DL) and Machine Learning (ML)-based methods, as subsets of the data-driven approach, are widely used in various phases of the software development life cycle, i.e., software requirements, software architecture and design, software implementation, software quality and analysis, and software maintenance [17]. For example, in the testing and analysis phase, DL and ML are concerned not only with the generation and selection of TCs, but also with the detection and analysis of defects and anomalies. However, despite the remarkable accomplishments in different domains, one of the major challenges impeding DL and ML methods is the lack of representative datasets [18,19]. Besides a sufficient amount of high-quality data, acquiring datasets with diverse scenarios that reflect real-world operating conditions can also be a challenge. Furthermore, the dynamic environment with changing conditions and satisfying the real-time requirements is another complicating factor when applying the data-driven approach with real-time system validation. Above all, in the automotive industry, especially for safety-relevant real-time systems, it is important to ensure that data-driven approaches comply with the applied development standards, e.g., ISO 26262. As a result, the availability of datasets is a constraint for the development process of DL-based FDD in real-world applications.

As mentioned above, one of the core elements of FDD model development is the dataset. In principle, obtaining fault-free data can be realised by recording the system behaviour in a fault-free mode [20]. However, obtaining representative data that captures the system's responses under fault conditions is complicated in terms of difficulty and cost [21]. In the automotive domain, due to the confidentiality of test data, it is rare to have publicly available real-world data that includes faulty behaviour [22]. Moreover, even in the case of limited data availability, the ratio of faulty to healthy data is small and unstable, which in turn leads to the problem of unbalanced data. One reason for this is the difficulty in simulating the open set of failure modes under complex working conditions [2]. Moreover, the implementation of hardware faults in real-world applications is not feasible [23].

In order to train a robust intelligent FDD model, the following dataset requirements should be ensured: 1. A high-quality dataset with a sufficient number of samples, including healthy and faulty operating conditions; 2. A high degree of fault classes coverage; 3. Consideration of the occurrence of the single and simultaneous faults; 4. Consideration of real-time constraints on system behaviour under normal and faulty conditions; 5. Offering different types of datasets, i.e., time series and textual data.

Therefore, overcoming the problem of the unavailability of representative dataset is still an open issue, and collecting faulty data that meets the above requirements should be further explored. In this article, we attempt to bridge this gap by proposing a novel framework for generating and collecting a representative real-time dataset, including healthy and faulty samples, during the development of Automotive Software Systems (ASSs). To demonstrate the benefits and applicability of the proposed framework, a high-fidelity simulation of a gasoline engine system is used as a case study. Vehicle dynamics, environment, a driver, and powertrain models were also considered to capture the detailed characteristics of the vehicle during data acquisition. The main contributions of the proposed framework can be summarised as follows:

- We propose a novel framework capable of generating real-time faulty datasets, taking into account the time constraints of the system behaviour.
- The framework enables the system behaviour to be automatically analyzed under realistic operating conditions in order to determine the critical faults that lead to a violation of the functional safety requirements.
- Not only single but also simultaneous fault occurrence can be simulated, covering the major types of random hardware sensor and actuator faults in ASSs.
- The faults are automatically injected into the targeted components in real time, without modifying the original system model and without having to use physical hardware.
- Real-time HIL simulation and high-fidelity simulation models are used to provide high-quality real-time coverage of the data collected.
- Finally, our proposed framework provides the ability to collect two types of representative datasets, i.e., time series and text log data.

The rest of this paper is structured as follows: Related work and other contributions are presented in Section 2. Section 3 introduces the proposed approach, highlighting the key stages of dataset collection and preparation. The implementation steps and the case study are described in Section 4. Section 5 summarises the results and findings. Finally, Section 6 presents the conclusion and future work.

2. Related Work

To tackle the problem of the unavailability of faulty data, several solutions have been proposed in the literature. Some of them have used online public datasets, while others have had to generate the data depending on the real prototype or simulated system. For example, Rengasamy et al. [24] used a standard gas turbine engine dataset provided by NASA. This dataset is employed to develop a DL-based model for predictive and diagnostic tasks. Similarly, based on Audi's industrial dataset, an intelligent model relying on DL methods for fault detection, isolation, identification and prediction has been developed in [25]. The target system of the work is autonomous automotive driving. Despite the advantage of using published standard data, the fault classes depend on the source and cannot be flexibly extended under identical operating circumstances. As an alternative solution, a static injection of the faulty sample into the data, based on normal distributions with one standard deviation, is used. For other researchers, the employment of a real prototype was an available alternative solution. For example, aiming at developing a robust FDD for vehicle engines, a real engine has been employed in [26] to capture the system behaviour under normal and abnormal conditions. As the data collected closely matched the real industrial application data, the developed model showed a high degree of applicability and robustness to environmental conditions such as noise. However, the major limitation of the applied technique is the potential risk and damage to the target

physical system in case of fault injection. Similarly, as a self-powered sensor to generate electrical signals, Yang et al. [27] proposed an EG-DEG device to record the flow rate of liquids. Based on the generated data, an intelligent system was developed to detect the particles and analyse the fluid. In the same respect, but for IoT applications, a new representative data-driven IoT/IIoT dataset has been proposed in [28] to be used for the training and evaluation of intrusion detection systems. From the proposed medium-scale testbed of datasets, new data characteristics, i.e., telemetry data, operating system data and network data of IoT/IIoT, can be generated and collected.

Depending on the hydraulic machinery test bench, sensor-related datasets have been collected and used to develop DL-based FDD of hydraulic systems in [29]. In order to provide representative faulty data, three types of sensor faults were injected into the collected dataset, i.e., constant, gain, and bais. However, the generated dataset does not reflect the working environment or provide any contextual information under faulty conditions in the real time. What is more, it does not offer a scenario that includes simultaneous failures. In the same regard, Bafroui et al. [30] proposed an FDD model for an automotive gearboxes system based on data collected from an experimental test bench in the laboratory. Some other studies have been carried out, based on data collected from real vehicle prototypes, to develop an intelligent solution for detecting unknown faults in test drive recordings [31]. However, in addition to the high cost, this type of experimentation could pose a risk not only to the vehicle but also to the tester in the event of a critical situation. Moreover, some types of faults cannot be investigated where the entire physical system could be damaged.

To overcome the limitations of using real physical systems, a simulation platform has been used to generate representative datasets. Based on modelling and simulation techniques, many researchers have proposed solutions to address the problem of the unavailability of data. For example, Tagawa et al. [32] used the FI method with a simulated system in the MATLAB/Simulink environment to generate the faulty dataset. In the aforementioned work, four driving scenarios were carried out to generate erroneous data alongside normal data. Similarly, Biddle et al. [33] approached the problem of faulty data unavailability by injecting five different types of faults into the simulated system, namely erratic, hard-over, drift, spike and stuck-at fault. To this end, the failure modes were modelled in the simulation environment, i.e., MATLAB/IPG CarMaker co-simulation, and then the faults were artificially injected according to the fault parameters. Consequently, an ML-based architecture for multiple faults in the multi-sensor FDD was created based on the collected data. However, despite the ability to reproduce the test under critical conditions with a high degree of confidence, the real-time conditions remain unconsidered. The effect of this is that the application of the target model in the real world becomes increasingly restricted. Not only that, the currently developed FDD models are validated based on simulation data generated by a simulation platform. By doing so, the real industrial conditions, such as influences, noise and uncertainties, are ignored [34]. In the context of autonomous driving, the Carla simulator has been used to generate object detection datasets in [35], including realistic driving images. The key feature of the generated dataset is the consideration of abnormal weather and lighting conditions, which improves the performance of the developed intelligent object detection system. However, the generated datasets focus on the images and contain neither faulty sensors' reading data nor system behaviour under the actuator faults. In the railway domain, a visual simulation framework called TrainSim has been proposed in [36] to generate synthetic datasets for the development of DL-based models. The generated datasets contain a wide range of realistic railway scenarios with labelled data using a set of simulated sensors. Cameras, LiDARs, and inertial measurement units have been used in the work to produce a variety of labelled images, but without considering time series data.

Recently, with the aim of overcoming the limitations of pure simulation, real-time HIL simulation has been introduced as a solution for generating a dataset covering various critical conditions [37]. For example, in [38], real-time simulation data collected from a HIL platform was used to develop an intelligent fault detection model for vehicle air brake

systems. By performing a virtual test drive under normal conditions, the healthy dataset, i.e., the wheel speed data of the four wheels, is collected. Despite the consideration of real-time constraints with realistic vehicle operating conditions, the work carried out is limited to two single-fault scenarios. In [39], to conduct the FDD strategy of traction systems in high-speed trains, two FI simulation platforms have been used, i.e., software-based FI and HIL-based FI. Through the proposed platform, six types of faults were considered so that two types were injected at three locations, i.e., sensor, traction motor, and traction converter. Despite the significant results achieved by the proposed framework compared to the state of the art, the dataset of the work is limited to specific type of faults and focuses on the system operation under the single faults. Similarly, a sensor-related fault dataset generated from HIL simulation is used in [40] to develop an integral diagnostic strategy for electric traction drives in railway applications. However, in order to capture the system behaviour under faulty conditions, the fault mode was simulated by extending the original system model with additional simulation blocks. This, in turn, could lead to the real-time constraints of the SUT being violated. Furthermore, the manual preparation of test cases and scenarios to identify the system state under an abnormal operation is time consuming. The more complex the system architecture, the more time and effort it takes to model faults [12].

Employing a representative real-time dataset generated from HIL, and based on hybrid DL techniques, an intelligent fault classification model has been proposed in [41,42] to be used during the development phases of ASSs, i.e., system integration testing. The basis of the developed model is the faulty data collected by programmatically injecting different sensor faults into the target system in real time without changing the model. As an extension of the aforementioned work, in this study, the simultaneous occurrence of the faults is considered, including transient and permanent faults. Moreover, an automation tool is employed to perform the fault injection process with high efficiency in terms of time and effort. Thus, the proposed work enables not only the fault injection but also the analysis and evaluation of the faults' effects to be performed automatically. As a result, besides the time series data, the textual logs of the test execution results are collected. Regardless of the currently available datasets for ML-based FDD, which neither represent real environments nor fulfil the time constraints, representative real-time datasets of ASSs based on an automated FI framework and a HIL simulator were proposed in this work. To reflect the real-world operating conditions, the drawbacks of the dataset with balance classes were overcome by considering the transient faults that lead to imbalanced data. An overview of the related work is provided in Table 1, highlighting the key features of the proposed work in comparison with other related work.

Table 1. Overview of the related work.

Related Work	Dataset	Application Domain	Target System	Data Generation	Faults	Remarks
[24]	Time series data from NASA	Prognostics and health management	Gas turbine engine	NA	Sensor-related faults	Coverage: low Concurrent faults: not considered. Real-time constraints: not considered.
[25]	Industrial dataset from Audi	FDD and health forecasting	Autonomous vehicles	NA	Sensor-related faults	Coverage: low Concurrent faults: considered. Real-time constraints: considered.
[26]	Real engine dataset	Fault detection and classification	Combustion engine	Using engine test rig	Sensor and actuator-related faults	Coverage: high Concurrent faults: not considered. Real-time constraints: considered.
[30]	Vibration signal dataset	Fault diagnosis	Automotive gearbox	Using experimental test bench	Sensor-related faults	Coverage: low Concurrent faults: not considered. Real-time constraints: considered.
[31]	Real test drive dataset	Anomaly detection	Test recordings of automotive systems	Using real vehicle	Sensor-related faults	Coverage: low Concurrent faults: not considered. Real-time constraints: considered.
[32]	Non-real-time Simulation dataset	Fault detection and analysis	Automotive systems	Using MATLAB simulation platform	Sensor-related faults	Coverage: low Concurrent faults: not considered. Real-time constraints: not considered.
[33]	Non-real-time Simulation dataset	Fault detection, identification and prediction	Control system of autonomous vehicle	Using MATLAB/IPG CarMaker co-simulation platform	Sensor-related faults	Coverage: low Concurrent faults: not considered. Real-time constraints: not considered.
[35]	Driving images Dataset	Object detection system	Autonomous driving	Using Carla simulator	Abnormal weather and lighting conditions	Coverage: low Concurrent faults: not considered. Real-time constraints: not considered.
[38]	Time series real-time simulation dataset	Fault detection and identification	Air brake automotive systems	Using HIL simulation	Sensor-related faults	Coverage: low Concurrent faults: not considered. Real-time constraints: considered.
Proposed work	Textual and time series real-time simulation dataset	Single and simultaneous FDC for HIL testing of ASSs	Validation of real-time ASSs	Using HIL simulation	Sensor-related faults	Coverage: high Concurrent faults: considered. Real-time constraints: considered.

3. Methodology

In this section, the proposed framework is presented, including real-time HIL simulation, FI framework, data analysis, and management, as shown in Figure 1.

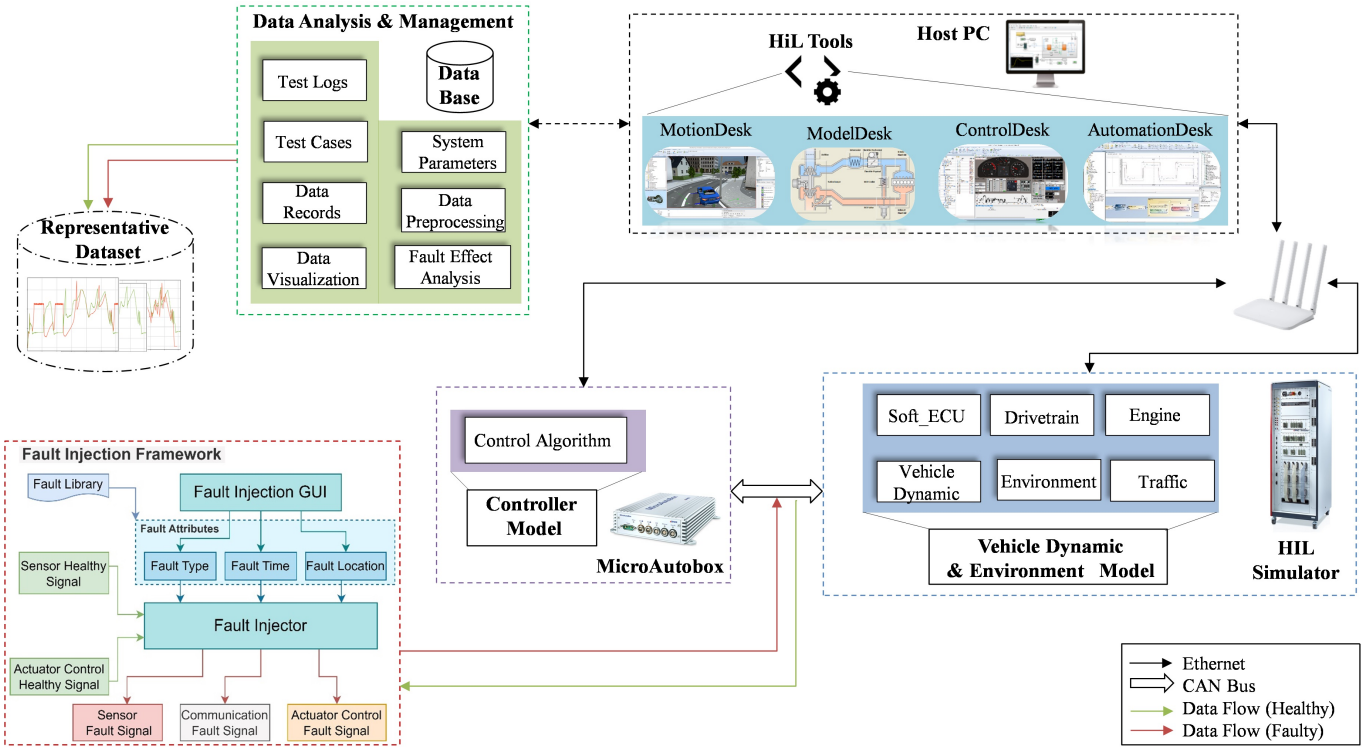


Figure 1. Proposed framework for representative real-time dataset generation.

3.1. Real-Time HIL Simulation System

The HIL system is the core of the framework, in which the complex target system is simulated and executed in real time. It consists of two main parts, namely the HIL simulator and the target prototype (controller). The HIL simulator is responsible for the real-time execution of the controlled system (plant). In our case, the controlled system is the entire vehicle model. The engine model, the vehicle dynamics model, the environment model, the traffic model, and the powertrain model are the main subsystems of the selected system. It is worth noting that the model of the SoftECU, i.e., the virtual ECU, is also included in the model of the controlled system. This allows the entire vehicle model, including the ECU model, to be executed in the HIL simulator as a Rapid Control Prototype (RCP), known as the offline mode. MicroAutoBox II, on the other hand, is the target machine of the SUT on which the ECU model is deployed and executed. MicroAutoBox II is regarded as the RCP and acts as the real ECU. Both parts of the HIL system are connected via the CAN bus. To establish the connection, the signal interface is modelled on both sides in the simulation environment, i.e., in the ECU model and in the plant model. The generated code of these models is injected into the ECU and HIL simulator via an Ethernet from the host PC.

On the host PC, the configurations and experiments are carried out. For this purpose, four software tools from dSPACE are used, namely MotionDesk, ConfigurationDesk, ModelDesk, AutomationDesk, and ControlDesk [43]. In addition to parameterising the system model, the ModelDesk is used to design the test drive scenarios and the TCs. Thanks to the model-based design approach, once the model was configured, the code for both models, i.e., SUT and plant, is automatically generated and deployed using ConfigurationDesk. MotionDesk is used to visualise the driving environment and dynamic traffic in 3D. Finally, ControlDesk is used for instrumentation, measurements, data acquisition, and real-time experiment control. This tool also allows switching between offline (SoftECU) and online (Real ECU) execution modes.

3.2. FI Framework with HIL

The FI process takes place at the signal interface between the ECU and the HIL simulator during a real-time execution. This ensures the real-time execution of the ECU and plant system model as a black box without modification. Furthermore, the injection process is executed programmatically, without extending the target system model with additional components. The input to the FI framework is healthy data representing the system behaviour under normal conditions. The target signals are then manipulated according to user-configured fault attributes. Three attributes should be configured in the fault injector prior to injection, i.e., fault type, fault location, and FI time. The fault type covers most sensor and actuator faults such as gain, offset, hard-over, stuck-at, delay, noise, packet loss, drift, and spike faults. The target sensor and actuator signal is identified as the location for faults to be injected. The duration of the FI and the time at which the fault is injected are specified according to the drive cycle or standard system behaviour. During the real-time execution of the HIL, once the above FI attributes have been specified, the fault injector manipulates the accessed healthy signals on the CAN bus accordingly. The manipulated signals interact with the system variables in a closed loop between the simulator and the ECU. As a result, the system response to the abnormal state of the faulty components, such as sensor or actuator, is captured as a deviation in the system behaviour, which is recorded as multivariate time series data. For each fault test case, the fault injection process is repeated during the system execution to achieve high coverage and diverse conditions in the collected datasets. In the case of simultaneous faults, the same process is performed, taking into account that two types of faults are injected simultaneously at two different locations.

To overcome the drawbacks of the manual fault injection process, in this study, the automatic execution of real-time fault injection was enabled in the proposed framework. To this end, an automation software tool, i.e., AutomationDesk, was used. AutomationDesk allows not only the automatic execution of systematic test cases, but also the automatic evaluation and reporting of test results. These features were the reason for selecting the tool as the environment for the automatic execution of the fault injection and evaluation process. Test scenarios, including the environment, roads and dynamic objects, are designed in ModelDesk and initiated for execution in AutomationDesk's test routines. Fault injection scenarios, on the other hand, are designed in AutomationDesk as a sequence of block hierarchies, as shown in Figure 2. In these routines, the fault injection process takes place in three phases: reading, manipulating signals, and writing. In the read block, the locations of the faults to be injected, i.e., the system components, are accessed via the signals of the system variables. The signals of the selected variables are then passed to the manipulation function, where the signals are manipulated according to the specified fault injection configurations. It is noteworthy that the injection process is performed by the source code of the manipulation function representing the fault modes, without extending the original architecture of the system model. Finally, the faults are activated by the write block at run-time without violating the timing behaviour of the SUT.

3.3. Dataset Collection and Preparation

As a result of executing the test cases, including fault injection, the system behaviour is recorded as time series data and evaluated against the expected behaviour. In this way, the critical faults that lead to a violation of the functional safety requirements can be automatically identified and documented in the generated test reports. The specifications of the recording process are set by the logging system in the HIL platform. At the system level, engine speed, engine torque, vehicle speed, throttle position, engine temperature, intake manifold pressure and rail pressure are considered as system variables in the data acquisition process. In this way, the fault-free data samples are collected by running the system under fault-free conditions. On the other hand, the faulty samples are collected as a result of system execution under faulty conditions, i.e., random sensor/actuator fault. Notably, the dataset containing unbalanced data is collected by injecting transient faults

temporarily for a certain period of time into the target sensor/actuator signals. As a result, the ratio of faulty samples to healthy samples is different in the generated dataset. For the development of an intelligent model using ML/DL methods, the collected dataset from the injection of critical faults as well as the healthy data are then considered as a representative real-time dataset.

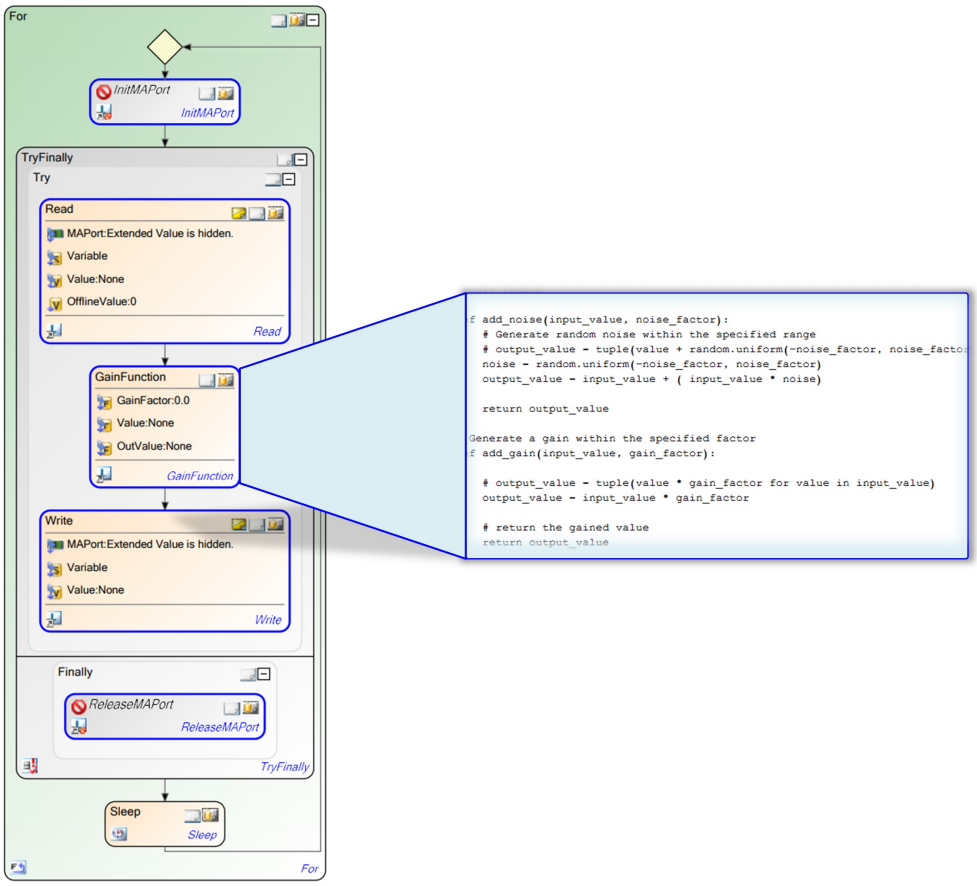


Figure 2. Automated real-time fault injection process.

Beside the data generation, data are pre-processed to remove outliers and redundant patterns. The main steps in data preparation are variable selection, data cleaning, data labelling, scaling and normalisation, balancing, and data division. In addition, data are converted into a suitable format, e.g., CSV, Mat or XML, to be ready for the ML/DL model development process. This phase also involves the correction of missing samples and the reduction in irrelevant data. In this way, not only can the computational cost be reduced, but the problem of over-fitting can also be avoided. To perform the labelling process, a multi-class, multi-label approach has been used in this work, so that all possible classes can be considered and labelled. To ensure that attributes with large values do not outweigh those with smaller values, the normalisation process is performed using the Z-score function by scaling the variable values in a range between [0, 1]. Finally, the pre-processed data are visualised on the host PC to obtain a deeper understanding of the patterns and features before training the DL/ML model. It is noteworthy that the data management is conducted through a host PC. This is where the system requirements, test case specifications, scenario descriptions, and test methods are documented.

4. Case Study and Experimental Implementation

To validate the applicability of the proposed framework, this section presents the selected case study, highlighting the system architecture and implementation steps.

4.1. Case Study

As a case study from the automotive domain, the gasoline engine system from dSPACE [43] was used to validate the proposed approach. The architecture of the system is shown in Figure 3. It can be observed that the various systems and subsystems have been modelled to accurately reflect the actual physical characteristics. MATLAB/Simulink was selected as the graphical simulation and modelling environment to model the dynamic system. The Simulink tool has important features for designing, analysing, and testing the dynamic system using the model-based approach. It also provides the ability to test the SUT in a simulated environment and automatically generate code before deployment, supporting real-time simulation. However, despite the realistic simulation of the chosen system, it does not allow lateral control driving. To overcome this limitation, in our case, a complex gasoline engine model was integrated into the vehicle dynamic system model. Thereby, lateral and longitudinal driving with manual and automatic transmission can be simulated. The major subsystem models that structure the gasoline engine are the exhaust, fuel, air path, cooler, and the piston engine system. Various components have been modelled and interconnected in a block diagram environment so that the comprehensive functions of the gasoline engine are simulated. Additionally, the powertrain, driver, and environmental models are included to capture the overall dynamic characteristics of the vehicle. The gasoline engine is controlled by the ECU in two modes, i.e., offline and online. In the offline mode, the soft ECU is internally connected to the engine, whereas a separate ECU model in the RCP is used as the real ECU to enable the online mode. Finally, the interface between the real ECU model and the gasoline engine model is realised using a Real Time Interface CAN multi-message blockset (RTICANMM), where the fault injections are configured and modelled. Thanks to the RTICANMM feature, the system signals can be accessed and manipulated programmatically without modifying the original system model.

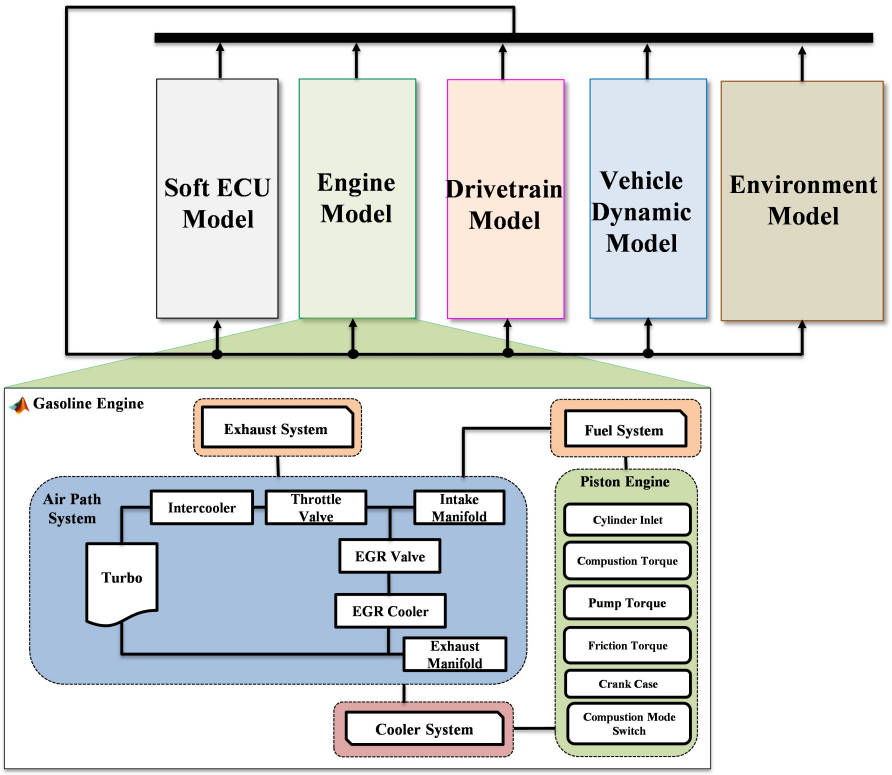
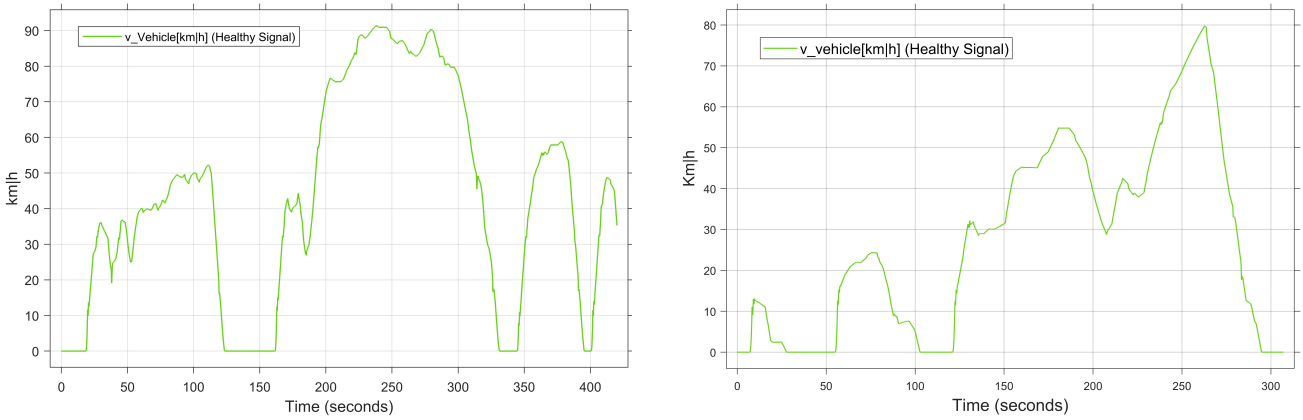


Figure 3. System architecture of the case study.

4.2. Experimental Setup

There are three levels of configuration in our case study, namely the system model configurations, experiment configurations, and FI configurations. At the first level, dSPACE's

software tools, ModelDesk and ConfigurationDesk, are used to specify the parameters and variables of the models based on the GUI. In addition, driving scenarios and TCs are designed based on the specifications. In our case, two driving scenarios are selected, namely "Highway" and "Ftp 75", as shown in Figure 4. This allows us to cover highway, non-urban/open road, and urban driving scenarios. Once the system and test configurations are specified, the code can be automatically generated and deployed to the target machines using a ConfigurationDesk. Two target machines are employed in our study, MicroAutoBox II for the ECU system and the HIL SCALEXIO simulator for the entire controlled vehicle system. As shown in Figure 5, data are transferred between the ECU and the HIL simulator via the CAN bus. Thanks to the ability of the model-based development approach to generate code from the model, the model code is automatically generated and deployed on the target machine. For this purpose, the ConfigurationDesk tool is used to build the generated code and transfer it to the corresponding hardware, i.e., MicroAutoBox II and HIL simulator. The RTICANMM interface plays an important role in providing access to all system variables at runtime. Therefore, implementing the FI framework with RTICANMM provides the ability to inject faults into the target components in real time without changing the system architecture.



(**a**) (**b**)

Figure 4. Driving scenario of the selected case study. (**a**) Highway driving scenario. (**b**) Ftp 75 driving scenario.

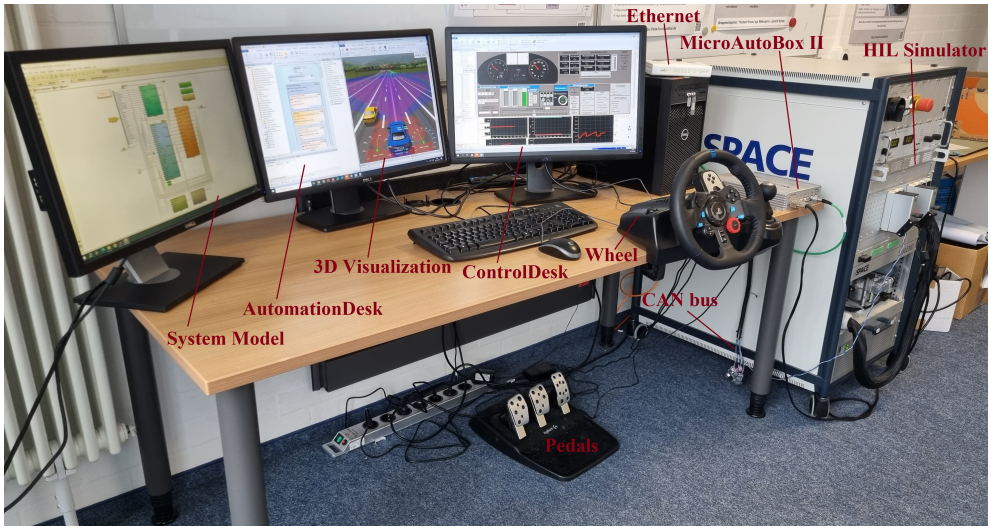


Figure 5. Real-time hardware-in-the-loop simulation.

Finally, the 3D visualisation of the driving manoeuvres, i.e., the defined scenarios, is modelled with MotionDesk. In other words, MotionDesk receives the data from the HIL

simulator and provides a visualisation of the moving objects in realistic 3D scenes. In this way, the normal and abnormal conditions of the driving environment, e.g., roads, weather, dynamic objects, can be considered, which in turn provides a good understanding of the simulated system.

On the second level, the configurations of the fault injector are set. To be specific, the FI attributes, i.e., fault location, fault type and FI time, are configured. As already mentioned, all system signals, i.e., sensors and actuators, can be accessed via the CAN bus interface. Therefore, the degree of coverage of our developed framework is high compared to the traditional approach. Some examples of sensor signals accessed are crank angle, battery voltage, engine speed, accelerator pedal position (APP), ignition and starter request, EGR mass flow, intake and exhaust manifold pressure, fuel pressure, coolant temperature, and railbar sensor. The actuators involved are the throttle valve, the fuel metering unit, the pressure control valve, and the ignition angle adjuster. Due to the significant effect of a fault in the APP and RPM sensor on the vehicle behaviour, we selected these locations for the FI in our study. On the other hand, nine different types of single faults and 15 combinations have been identified to demonstrate the coverage of the fault data generation. In this way, not only the single fault-based dataset but also the concurrent faults-based dataset can be collected. Finally, the timing and duration of the fault injection is determined based on the selected drive cycle. FI can be permanent until the faulty component is addressed, or it can be transient, occurring frequently for a specified duration. By injecting permanent and transient faults, both balanced and imbalanced datasets can be acquired for developing a reliable FDD model for real-world applications. The detailed configuration of the FI framework for single and simultaneous faults is shown in [44].

The experimental setup takes place on the last level, where the measurements, recording and instrumentation are carried out. The sampling time of the signal measurements in this study is set to 0.001 s. At the system level, six variables of the target system are selected to represent the recorded system behaviour under normal and fault conditions. These system variables are the throttle position, engine temperature, mean effective engine torque, engine speed, intake manifold pressure, rail pressure, and vehicle speed, as illustrated in Figure 6.

	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. Generated time series dataset.

With the aid of ControlDesk, which can be used to configure and control the runtime experiments, the system is executed under various conditions. Once the real-time execution is complete, the system response to the faults is captured as time series data and stored in a CSV file containing system variables and data samples. On the other hand, TCs designed in ModelDesk are executed on the target SUT to generate the documented error logs as the text data. The TCs should contain the data inputs and the expected outputs. According to the actual system behaviour, the defined expected output is then compared with the measured output. Depending on the test execution, the results can be either a pass or fail. Thanks to ModelDesk's ASM test functionality, the TCs are automatically executed and evaluated. Note that the generated report contains key information about the specifications of the TCs as well as the execution results (passed or failed). The report also contains

a failure log describing the failure that occurred. Using the logs of the failed TCs, an intelligent model for RCA can be developed based on natural language processing and machine learning methods.

5. Results and Discussion

In this section, the results of the data collection for the different FI configurations are presented. Specifically, the generated data are described, highlighting the effects of single and concurrent faults on system behaviour. Moreover, the log data of failed TCs generated in the test report is presented.

5.1. Representative Time Series Dataset

In real time, both the SUT and the controlled system are executed under normal and abnormal conditions according to the aforementioned system and FI configurations. A total of 40 experiments were conducted with single and simultaneous faults in two driving scenarios. In particular, two files as healthy data from "highway" and "ftp 75" scenarios, 18 files representing the single fault types, and 30 files for simultaneous faults with 14 combinations were collected. As a result, 50 CSV files have been obtained as HIL test records. Considering six types of single fault and sampling time 01001, the total number of faulty data samples from the data collection phase is 947,000, with 420,000 samples for each experiment, as shown in Table 2. On the other hand, the total number of faulty data samples from the concurrent FI process is 2,267,000, with 420,000 samples for each experiment, as can be observed in Table 3.

Table 2. 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 3. 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

Focusing on packet loss and stuck-at as examples of fault types, the system behaviour under the injected fault can be captured at the system level on vehicle and engine speed signals. In Figure 7a, the stuck-at fault was injected into the RPM sensor after 10 s of the driving cycle. The engine's behaviour deviates significantly from 750 rpm in the normal

range to 2300 rpm as soon as the fault is activated, which in turn leads to an increase in energy consumption. Until 180 s, the control unit is unable to cope with the occurred fault. However, after 180 s, the system tries to overcome the fault and behave according to the desired behaviour. Then, the deviation and fluctuations occur again from 210 s and 290 s, respectively. The reason behind this change is the driving variations and conditions between the acceleration mode, deceleration mode and the steady state mode. In other words, the control strategy cannot properly mitigate the disturbance when the vehicle is accelerating and decelerating rapidly, as shown in Figure 7b.

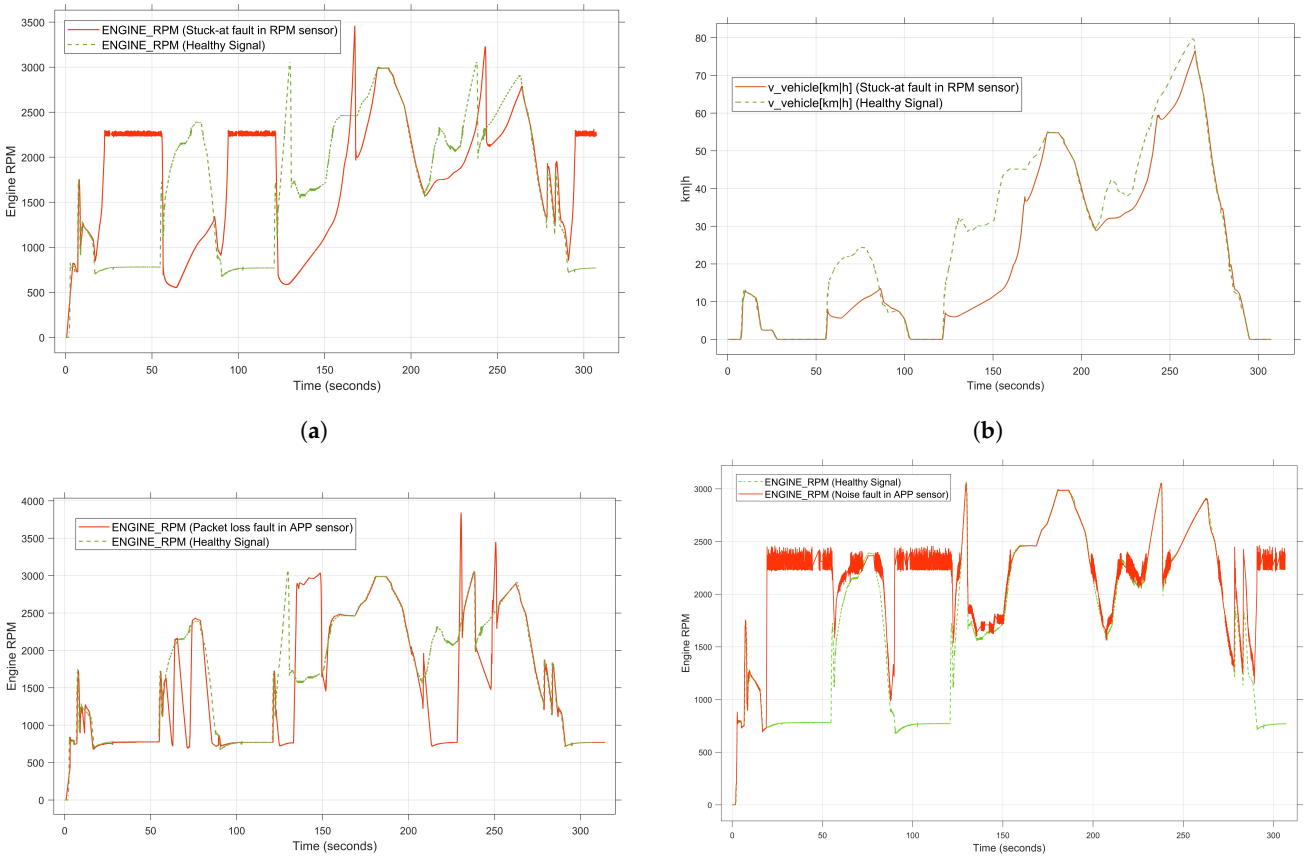


Figure 7. Effect of the single faults on the system behaviour. (**a**) Engine system behaviour under stuck-at fault. (**b**) Vehicle system behaviour under stuck-at fault. (**c**) Engine system behaviour under packet loss fault. (**d**) Engine system behaviour under noise fault.

On the other hand, by injecting the packet loss into the APP sensor during the driving cycle, the effect of the fault on the vehicle speed and the engine speed cannot be directly observed, as shown in Figure 7c. However, from sec 75 onwards, significant fluctuations can be observed. At the system level, the behaviour of the vehicle and engine deviates from the standard in the form of severe jerking. At this moment, the ECU tries to overcome the signal losses for a certain period. This is due to the fact that the behaviour remains in a state of fluctuation, while the loss time is less than 3 s.

Similarly, once a noise is injected into the APP sensor at 25 s, the effect can be observed in the form of a strong amplitude fluctuation in the RPM signals (see Figure 7d). The effect is particularly noticeable during the stationary period of the vehicle speed; i.e., between 20–55, 80–125 and 280–330 s, the fault effect is manifest. On the other hand, during the driving mode, the SUT seeks to exhibit the desired driving behaviour despite the occurrence of the fault. Therefore, the system will follow the defined scenario between 125 and 280 s with an acceptable range in the RPM signal.

According to the scenario "ftp 75", what can be observed in Figure 8a,b is the effect of injecting a single fault into the APP and RPM sensors, respectively. Specifically, 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. On the other hand, the effect of the delay in the RPM sensor signal causes the engine speed to fluctuate between 265–307 s, as shown in Figure 8b. During this period, the ECU cannot perfectly perform the desired behaviour with an increasing delay of the received signal, especially when the driving mode is changed.

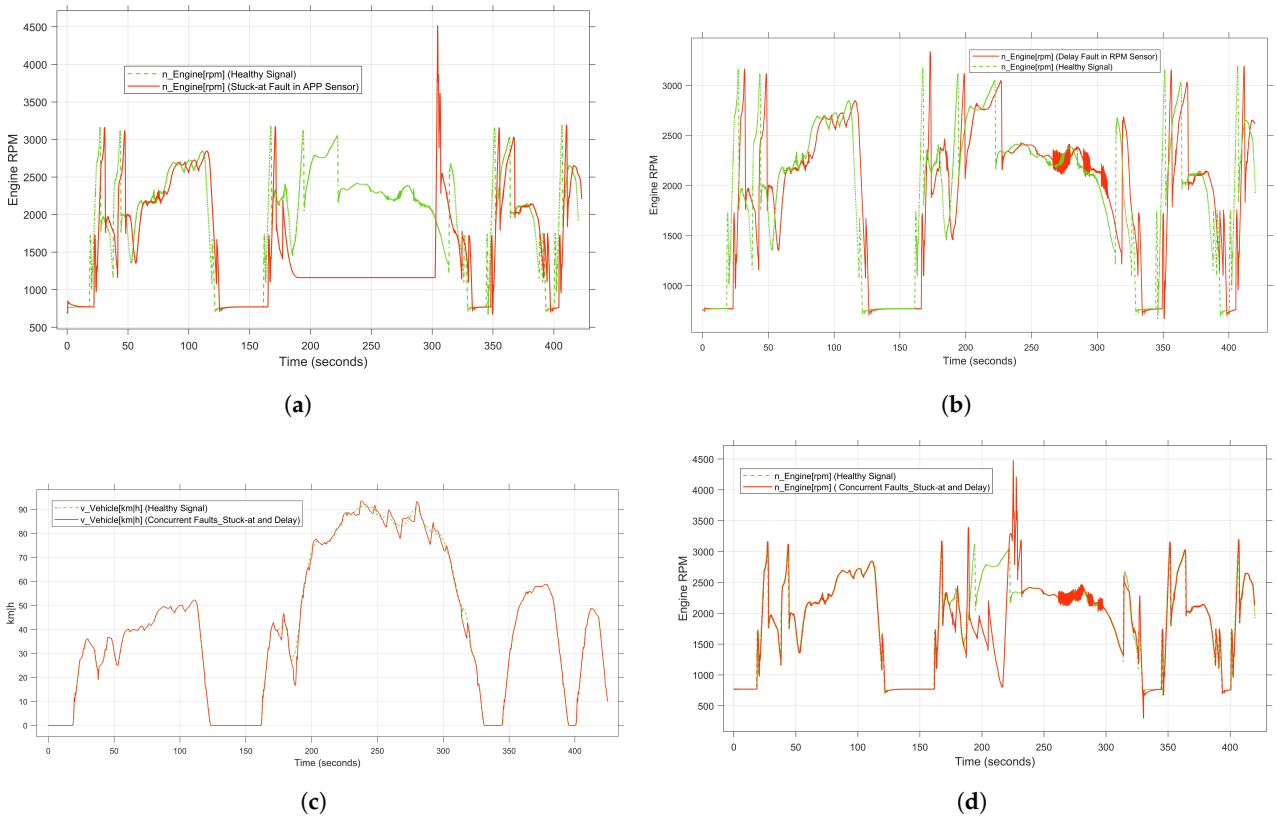


Figure 8. Effect of the simultaneous faults on the system behaviour. (**a**) Engine system behaviour under stuck-at fault. (**b**) Engine system behaviour under delay fault. (**c**) Vehicle system behaviour under stuck-at and delay faults simultaneously. (**d**) Engine system behaviour under stuck-at and delay faults simultaneously.

In the case of simultaneous faults, two different fault types are injected simultaneously into two different locations. For example, while the stuck-at fault is injected into the APP sensor, the delay fault is simultaneously injected into the engine speed sensor. To put it in another way, two or more factors have contributed to creating a novel pattern in the signals as a result of the simultaneous FI. Detailed information regarding the setup of the simultaneous FI can be found at [45]. Figure 8c illustrates the effect of the simultaneous faults on the vehicle system behaviour, i.e., vehicle speed, between 170–330 s. The vehicle speed closely follows the desired behaviour up to 170 s, the time at which the faults are activated. As a result of the large fluctuations shown in Figure 8d, the vehicle is unable to accelerate at 190 s, the time at which the speed drops to 18 km/h, causing a risk of failure. However, the system returns to an acceptable state with small deviations.

Although the collected data represents comprehensive characteristics of the system behaviour, the generated CSV files contain redundant or useless information. For example, each file contains information about the recording process as well as the data samples. Hence, the collected data should be pre-processed to remove outliers and redundant information. Three steps are applied to the data in the pre-processing phase, i.e., removal of

useless information and outliers, data scaling and normalisation, and data splitting. After converting the csv files into Excel format, the training variables are selected and the non-useful data are removed. By doing so, useful data samples are ensured for the development of the ML model. The next step is to apply a normalisation function to all variables, as each variable has a different range of values. Thus, all variable values are scaled in the range between [0–1]. The last step is to divide the data into three parts, i.e., training data, validation data and test data. It should be noted that when developing ML models based on supervised learning, the process of labelling the data should take place before splitting the data. During this process, the defect class is assigned to the appropriate data for the classification task. In general, the data in each part are divided into 80% for training, 10% for validation, and 10% for the test part.

In order to benchmark the proposed framework, the characteristics of the real-time dataset generated in this study have been compared with the currently available datasets from the existing systems, as presented in Table 4. It can be observed that most of the existing simulation systems provide time series data collected from the system components. However, the occurrence of the concurrent faults has not been considered in the provided datasets, e.g., CMAPSS, CarMaker, and IIoT datasets. Instead, in our proposed work, data samples have been collected under both single and concurrent faults in real time. Notably, the A2D2 dataset only contains samples of single and concurrent faults. However, it is limited to four fault types under specific driving scenarios, which do not reflect real-world operating conditions. In our proposed work, besides considering the effect of adverse weather conditions and lighting, various fault scenarios occurring in the sensors and actuators were simulated in real time.

Table 4. A comparison of available datasets from the existing systems with the proposed dataset.

Dataset	Year	Type	Size	Label	Faults	Scenarios
CMAPSS [24]	2008	Time series data	76 k	Yes	Single	Limited
A2D2 [25]	2020	Time series data	657 k	Yes	Single, concurrent	Limited
CarMaker [33]	2021	Time series data	3 m	Yes	Single	Limited
IIoT [28]	2019	Time series, logs	-	Yes	Single	Diverse
CarFree [35]	2022	Images	5 k	Yes	-	Diverse
TrainSim [36]	2023	Images, point clouds	8k	Yes	-	Diverse
Proposed work	2024	Time series and logs	8.8 m	Yes	Single, concurrent	Diverse

5.2. Automated Test Evaluation

In the proposed framework, besides the automated fault injection, the test case evaluation is also performed automatically with the aid of the AutomationDesk tool. To this end, the expected output of the SUT should be defined in advance. The evaluation function, in turn, compares the actual and the expected behaviour. In the case of a discrepancy between both aforementioned outputs, the executed test cases are evaluated as failed. As can be observed in Figure 9, the measured output of the SUT is outside the defined tolerance range, i.e., the vehicle position, which means that the SUT was not able to mitigate the fault, resulting in a failure. In other words, as a result of the fault propagating from the component through the subsystems to other components, a failure occurs at the system-level, as the required functions can no longer be performed. Thanks to the aforementioned tool, hundreds of test cases can thus be automatically executed and evaluated without the need for human intervention. Furthermore, the results are documented in a generated report, which plays an important role in identifying critical faults. It is worth noting that not all introduced faults result in a system-level failure; rather, they are mitigated by the fault tolerance mechanism. On the other hand, faults that lead to failures, i.e., termination of system functionality, are considered critical faults. The system behaviour resulting from the injection of critical faults is captured as a representative faulty dataset and then used to develop the FDD model.

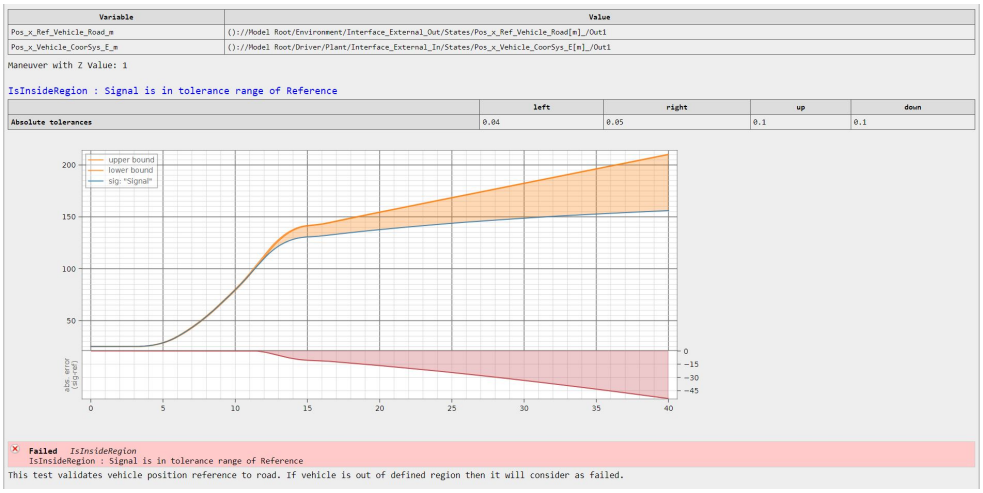


Figure 9. Evaluation of the test cases' execution.

5.3. Representative Textual Dataset

As a result of the TCs execution, comparing the desired system signals with the observed outputs, a test report was generated. What is of interest in the report is the logs, written in natural language, describing the failures that occurred. An example of a generated test report is shown in Figure 10. As can be observed, a short text is written indicating the failed TCs as information about the causes of the failure. However, without the expert knowledge of the tester, these transcripts are not able to correctly identify the failure causes. Therefore, based on the collected logs, an intelligent model can be trained to perform the RCA process efficiently without human intervention. Furthermore, the generated logs can be used to provide additional information to support the decision of the signal-based FDD model. Similar to the time series data, a number of pre-processing steps are performed on the collected logs prior to model training. The main pre-processing steps for text data are tokenisation, normalisation, stop word removal and stemming. After applying Natural Language Processing (NLP) techniques to the collected data, the training is divided into three parts: training with 80%, validation with 10%, and testing with 10%.

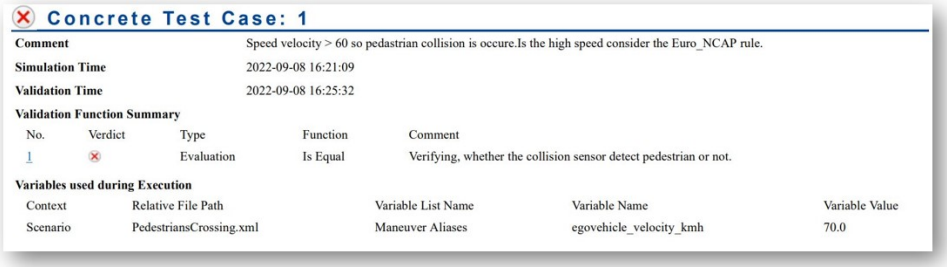


Figure 10. Generated report of test case execution.

5.4. Setup Cost Discussion

Despite the reliability of the proposed framework in simulating realistic system behaviour in the presence of faults, several complicating factors concerning the setup and implementation should be considered. Since three different fault attributes have to be specified in the setup phase, an unlimited number of possible configurations could be created, i.e., an unbounded fault space. For complex software systems, the trade-off between achieving a high test coverage and exploring the most effective critical fault is still a challenge. Notably, in this study, the fault test cases were identified based on three factors, i.e., representative and realistic faults to be injected, the impact and speed of the injected fault causing failures, and the variety of fault scenarios. Besides the setup effort, a real-time system should be provided to enable a real-time execution of the system model along

with the framework. For this purpose, in our study, a MicroAutoBox II embedded PC (DS1401 Base Board) with 900 MHz processor, 6th Gen.Intel® CoreTM i7-6822EQ, 16 MB memory and 340 ms boot time for 3 MB application is utilized. Based on the debugging run-time information for 10-s log and the intended sampling time of 1 ms, the maximum task execution time is 0.738 ms. Each test case has a test execution time of 40,242.6164 ms, including the evaluation period.

Finally, depending on the specification of the log data system, a huge amount of data can be collected from heterogeneous components, which poses another challenge in terms of the computational cost of ML/DL model development. Specifically, considering 0.001 s as the sampling period, several million data samples are generated, including outliers, noise and missing/redundant samples. Therefore, training the ML/DL model not only requires significant effort and time in the pre-processing and training phases but also demands powerful hardware with high computational capability, such as GPUs or TPUs, to process such a large amount of data.

6. Conclusions

A novel framework for generating and collecting representative real-time dataset for ML/DL applications during the system validation process of ASSs is proposed in this article. To this end, HIL simulation and an automated fault injection method are used considering real-time constraints. The objective of the proposed framework is to provide the required dataset for the training, testing, and validation of the DL-based intelligent failure analysis of HIL test records. Innovatively, it requires neither fault modelling within the system architecture nor the physical hardware device to perform the FI for data acquisition. Instead, the target faults are programmatically injected in real time without any modelling effort and time. Furthermore, the faults are injected and evaluated in an automated manner, without modifying the SUT model, by using an automation tool, i.e., AutomationDesk. The proposed framework is able to simulate and collect faulty data in different formats, i.e., time series and textual. As time-series data, the effect of random hardware faults is simulated at the system level in the form of transient and permanent faults. In this manner, the system behaviour under various critical fault conditions can be accurately and reliably captured in real time. The higher the quality of the captured data, the higher the performance of the developed DL model. In this study, a complex automotive gasoline engine has been selected to validate and demonstrate the proposed approach. The outcomes show the applicability of the approach in representing the effects of single and simultaneous faults at the system level. Nine different types of sensor and actuator faults were simulated in this study. Moreover, the combination of two faults at different locations simultaneously was simulated. On the other hand, the textual descriptions of the failed TCs were generated as a result of injecting the faults in real time during the system executions. To conclude, a high quality dataset with a high fault class coverage can be collected with low effort during real time simulation. Such datasets can be used for developing innovative algorithms and DL models for various tasks and applications, e.g., the fault detection, diagnosis, prognosis and root cause analysis of failed TCs.

As future work, the proposed framework can be extended to cover the faults that may occur in other system components, e.g., ECU and the network. Furthermore, new features can be added to the proposed framework by considering the user's behaviour during data acquisition, i.e., the HIL-based digital test drive. Finally, the proposed framework will be further improved using the ML approach so that the fault test case generation process can be performed automatically, effectively and systematically, according to the requirements.

Author Contributions: Conceptualization, M.A.; methodology, M.A.; software, M.A.; validation, M.A.; formal analysis, M.A.; investigation, M.A.; resources, M.A., C.K. and A.R.; data curation, M.A.; writing—original draft preparation, M.A.; writing—review and editing, M.A., C.K. and A.R.; visualization, M.A.; supervision, M.A., C.K. and A.R.; project administration, M.A., C.K. and A.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data available on request due to restrictions.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. 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.
2. Kukkala, V.K.; Tunnell, J.; Pasricha, S.; Bradley, T. Advanced driver-assistance systems: A path toward autonomous vehicles. *IEEE Consum. Electron. Mag.* **2018**, *7*, 18–25. [CrossRef]
3. Koopman, P.; Wagner, M. Challenges in autonomous vehicle testing and validation. *SAE Int. J. Transp. Saf.* **2016**, *4*, 15–24. [CrossRef]
4. D'Ambrosio, J.; Soremekun, G. Systems engineering challenges and MBSE opportunities for automotive system design. In Proceedings of the 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Banff, AB, Canada, 5–8 October 2017; pp. 2075–2080.
5. Pretschner, A.; Broy, M.; Kruger, I.H.; Stauner, T. Software engineering for automotive systems: A roadmap. In Proceedings of the Future of Software Engineering (FOSE'07), Minneapolis, MN, USA, 23–25 May 2007; pp. 55–71.
6. Ebert, C.; Favaro, J. Automotive software. *IEEE Softw.* **2017**, *34*, 33–39. [CrossRef]
7. Bello, L.L.; Mariani, R.; Mubeen, S.; Saponara, S. Recent advances and trends in on-board embedded and networked automotive systems. *IEEE Trans. Ind. Inform.* **2018**, *15*, 1038–1051. [CrossRef]
8. *ISO 26262-10:2018*; Road Vehicles Functional Safety. Available online: https://www.iso.org/standard/68392.html (accessed on 6 January 2024).
9. Garousi, V.; Felderer, M.; Karapıçak, Ç.M.; Yılmaz, U. Testing embedded software: A survey of the literature. *Inf. Softw. Technol.* **2018**, *104*, 14–45. [CrossRef]
10. Shokry, H.; Hinchey, M. Model-Based Verification of Embedded Software. *Computer* **2009**, *42*, 53–59. [CrossRef]
11. Himmler, A.; Lamberg, K.; Beine, M. *Hardware-in-the-Loop Testing in the Context of ISO 26262*; Technical Report, SAE Technical Paper; SAE: Warrendale, PA, USA, 2012.
12. Mihalič, F.; Truntič, M.; Hren, A. Hardware-in-the-loop simulations: A historical overview of engineering challenges. *Electronics* **2022**, *11*, 2462. [CrossRef]
13. Chen, Y.; Chen, S.; Zhang, T.; Zhang, S.; Zheng, N. Autonomous vehicle testing and validation platform: Integrated simulation system with hardware in the loop. In Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, China, 26–30 June 2018; pp. 949–956.
14. Jordan, C.V.; Hauer, F.; Foth, P.; Pretschner, A. Time-series-based clustering for failure analysis in hardware-in-the-loop setups: An automotive case study. In Proceedings of the 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Coimbra, Portugal, 12–15 October 2020; pp. 67–72.
15. Vermeulen, B. Functional debug techniques for embedded systems. *IEEE Des. Test Comput.* **2008**, *25*, 208–215. [CrossRef]
16. Nair, V.V.; Koustubh, B.P. Data analysis techniques for fault detection in hybrid/electric vehicles. In Proceedings of the 2017 IEEE Transportation Electrification Conference (ITEC-India), Pune, India, 13–15 December 2017; pp. 1–5.
17. Shafiq, S.; Mashkoor, A.; Mayr-Dorn, C.; Egyed, A. A literature review of using machine learning in software development life cycle stages. *IEEE Access* **2021**, *9*, 140896–140920. [CrossRef]
18. Badihi, H.; Zhang, Y.; Jiang, B.; Pillay, P.; Rakheja, S. A comprehensive review on signal-based and model-based condition monitoring of wind turbines: Fault diagnosis and lifetime prognosis. *Proc. IEEE* **2022**, *110*, 754–806. [CrossRef]
19. Neupane, D.; Seok, J. Bearing fault detection and diagnosis using case western reserve university dataset with deep learning approaches: A review. *IEEE Access* **2020**, *8*, 93155–93178. [CrossRef]
20. Zhang, S.; Zhang, S.; Wang, B.; Habetler, T.G. Deep learning algorithms for bearing fault diagnostics—A comprehensive review. *IEEE Access* **2020**, *8*, 29857–29881. [CrossRef]
21. Zhang, T.; Chen, J.; Li, F.; Zhang, K.; Lv, H.; He, S.; Xu, E. Intelligent fault diagnosis of machines with small & imbalanced data: A state-of-the-art review and possible extensions. *ISA Trans.* **2022**, *119*, 152–171. [PubMed]
22. Theissler, A.; Pérez-Velázquez, J.; Kettelgerdes, M.; Elger, G. Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry. *Reliab. Eng. Syst. Saf.* **2021**, *215*, 107864. [CrossRef]
23. Fernandes, M.; Corchado, J.M.; Marreiros, G. Machine learning techniques applied to mechanical fault diagnosis and fault prognosis in the context of real industrial manufacturing use-cases: A systematic literature review. *Appl. Intell.* **2022**, *52*, 14246–14280. [CrossRef] [PubMed]
24. Rengasamy, D.; Jafari, M.; Rothwell, B.; Chen, X.; Figueredo, G.P. Deep learning with dynamically weighted loss function for sensor-based prognostics and health management. *Sensors* **2020**, *20*, 723. [CrossRef] [PubMed]

25. Safavi, S.; Safavi, M.A.; Hamid, H.; Fallah, S. Multi-sensor fault detection, identification, isolation and health forecasting for autonomous vehicles. *Sensors* **2021**, *21*, 2547. [CrossRef]
26. Jung, D. Data-driven open-set fault classification of residual data using Bayesian filtering. *IEEE Trans. Control Syst. Technol.* **2020**, *28*, 2045–2052. [CrossRef]
27. Yang, L.; Yu, J.; Guo, Y.; Chen, S.; Tan, K.; Li, S. An Electrode-Grounded Droplet-Based Electricity Generator (EG-DEG) for Liquid Motion Monitoring. *Adv. Funct. Mater.* **2023**, *33*, 2302147. [CrossRef]
28. Alsaedi, A.; Moustafa, N.; Tari, Z.; Mahmood, A.; Anwar, A. TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems. *IEEE Access* **2020**, *8*, 165130–165150. [CrossRef]
29. Mallak, A.; Fathi, M. Sensor and component fault detection and diagnosis for hydraulic machinery integrating LSTM autoencoder detector and diagnostic classifiers. *Sensors* **2021**, *21*, 433. [CrossRef] [PubMed]
30. Bafroui, H.H.; Ohadi, A. Application of wavelet energy and Shannon entropy for feature extraction in gearbox fault detection under varying speed conditions. *Neurocomputing* **2014**, *133*, 437–445. [CrossRef]
31. Theissler, A. Detecting known and unknown faults in automotive systems using ensemble-based anomaly detection. *Knowl.-Based Syst.* **2017**, *123*, 163–173. [CrossRef]
32. Tagawa, T.; Tadokoro, Y.; Yairi, T. Structured denoising autoencoder for fault detection and analysis. In Proceedings of the Asian Conference on Machine Learning, PMLR, Hong Kong, China, 20–22 November 2015; pp. 96–111.
33. Biddle, L.; Fallah, S. A novel fault detection, identification and prediction approach for autonomous vehicle controllers using svm. *Automot. Innov.* **2021**, *4*, 301–314. [CrossRef]
34. Yin, Z.; Hu, N.; Chen, J.; Yang, Y.; Shen, G. A review of fault diagnosis, prognosis and health management for aircraft electromechanical actuators. *IET Electr. Power Appl.* **2022**, *16*, 1249–1272. [CrossRef]
35. Jang, J.; Lee, H.; Kim, J.C. Carfree: Hassle-free object detection dataset generation using carla autonomous driving simulator. *Appl. Sci.* **2021**, *12*, 281. [CrossRef]
36. D'Amico, G.; Marinoni, M.; Nesti, F.; Rossolini, G.; Buttazzo, G.; Sabina, S.; Lauro, G. TrainSim: A railway simulation framework for LiDAR and camera dataset generation. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 15006–15017. [CrossRef]
37. Gonzalez-Jimenez, D.; Del-Olmo, J.; Poza, J.; Garramiola, F.; Madina, P. Data-driven fault diagnosis for electric drives: A review. *Sensors* **2021**, *21*, 4024. [CrossRef]
38. Raveendran, R.; Devika, K.; Subramanian, S.C. Brake fault identification and fault-tolerant directional stability control of heavy road vehicles. *IEEE Access* **2020**, *8*, 169229–169246. [CrossRef]
39. Guo, L.; Li, R.; Jiang, B. Fault detection and diagnosis using statistic feature and improved broad learning for traction systems in high-speed trains. *IEEE Trans. Artif. Intell.* **2022**, *4*, 679–688. [CrossRef]
40. Garramiola, F.; Del Olmo, J.; Poza, J.; Madina, P.; Almandoz, G. Integral sensor fault detection and isolation for railway traction drive. *Sensors* **2018**, *18*, 1543. [CrossRef] [PubMed]
41. 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**, *22*, 4066. [CrossRef] [PubMed]
42. 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**, *23*, 6606. [CrossRef] [PubMed]
43. Automotive Simulation Models. Available online: https://www.dspace.com/en/pub/home/products/sw/automotive_simulation_models.cfm#175_26315 (accessed on 11 April 2023).
44. 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**, *22*, 1360. [CrossRef]
45. 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**, *11*, 140022–140040. [CrossRef]