TU Clausthal

MASTER THESIS

A machine learning-based framework for effectively analyzing execution results of the back-to-back test method during real-time validation of automotive software systems

Shaoqing Guo
Matriculation Number: 529572

MSc Informatik

1st Examiner:	Prof. Dr. Andreas Rausch
2nd Examiner:	apl. Prof. Christoph Knieke
Supervisor:	Dr. Mohammad Abboush

Institute for Software and System Engineering, Clausthal University of Technology, Germany

March 27, 2025

Declaration of Authorship

I have read and understood the guidelines of the Technical University of Clausthal and those published in the ITE bulletin, including those regarding the use of literature and other sources. I confirm that I have prepared this thesis independently by myself. Any information taken from other sources and being reproduced in this thesis is clearly referenced.

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

I hereby agree that my master thesis may be exhibited in the institute's library and kept for inspection. Note: is not applicable if the thesis is rated as confidential in the companies agreement!

Clausthal-Zellerfeld, 27th March 2025	
Location, Date	First name and family name

Abstract

With the rapid development of the automotive industry towards electrification, intelligence, and connectivity, automotive software systems have shown unprecedented growth in complexity. Modern high-end vehicles contain up to 100 million lines of code, with complexity comparable to advanced aviation systems. This complexity introduces significant verification challenges for back-to-back (B2B) testing in line with the ISO 26262 standard, including ensuring real-time responsiveness, accurately handling multi-variable interactions, and dynamically adapting fault detection thresholds under varying operational scenarios. This research proposes an intelligent analysis framework based on deep learning and clustering analysis to improve the efficiency and accuracy of back-to-back testing for automotive software systems. The framework integrates a CNN-LSTM deep autoencoder for feature extraction and anomaly detection based on the fault-free behavior, along with K-means clustering algorithm for fault pattern classification. Our goal is to address the limitations of traditional fixed-threshold methods by providing a more adaptive, accurate, and interpretable approach to detecting and classifying faults in complex automotive systems. Through systematic investigation and extensive experimentation, we designed and validated a two-stage processing approach where potential anomalies are first identified through reconstruction error analysis before applying clustering for fault pattern differentiation. This approach not only improved computational efficiency but also enhanced fault identification clarity. The framework was evaluated using unseen testing data from both Simulink Model-in-the-Loop environment (MIL) and SCALEXIO real-time Hardware-in-the-Loop platform (HIL) across various driving scenarios, demonstrating its robustness and generalizability. The main contributions of this research include: an innovative CNN-LSTM hybrid architecture that demonstrated superior performance in automotive signal fault detection with 90.54% accuracy and a 0.926 F1 score; optimized K-means clustering with fault-based K value selection that improved the Davies-Bouldin Index by 38.6% in multi-fault scenarios, providing more interpretable fault pattern differentiation while reducing processing time by 36.4%; strong cross-scenario generalization capability, allowing a single model trained on mixed scenario data to be deployed across various operational conditions; and a comprehensive intelligent B2B testing framework that provides engineers with automated feature extraction, statistically optimized threshold selection, and clear visualization and interpretation of

fault patterns. Experimental results demonstrate that the proposed CNN-LSTM-DAE model with optimized K-means clustering framework exhibits strong robustness in high-noise environments, successfully identifies and clusters both single and multiple fault scenarios, maintains highly stable performance across different driving scenarios, and keeps all processing times within the computational costs requirements. This research provides a novel intelligent approach to automotive software back-to-back testing, overcoming the limitations of traditional methods and offering a more efficient and accurate solution for ensuring system safety and reliability.

Zusammenfassung

Mit der rasanten Entwicklung der Automobilindustrie in Richtung Elektrifizierung, Intelligenz und Konnektivität haben Automobilsoftwaresysteme ein beispielloses Wachstum an Komplexität gezeigt. Moderne Fahrzeuge der Oberklasse enthalten inzwischen bis zu 100 Millionen Codezeilen, deren Komplexität mit fortschrittlichen Luftfahrtsystemen vergleichbar ist. Diese Komplexität führt zu erheblichen Herausforderungen bei der Verifizierung von Back-to-Back (B2B) Tests gemäß der ISO-26262-Norm, insbesondere hinsichtlich der Sicherstellung von Echtzeitreaktionen, der präzisen Handhabung mehrerer gleichzeitig wechselwirkender Variablen sowie der dynamischen Anpassung von Fehlererkennungsschwellen unter variierenden Betriebsbedingungen.

In dieser Arbeit wird ein intelligentes Analyse-Framework basierend auf Deep Learning und Clusteranalyse vorgeschlagen, um die Effizienz und Genauigkeit von Back-to-Back Tests für Automobilsoftwaresysteme zu verbessern. Das Framework integriert einen CNN-LSTM Deep Autoencoder zur Merkmalsextraktion und Anomalieerkennung basierend auf dem fehlerfreien Verhalten sowie einen K-Means-Clustering-Algorithmus zur Klassifizierung von Fehlermustern. Ziel ist es, die Grenzen traditioneller Methoden mit festgelegten Schwellenwerten zu überwinden und eine adaptive, genauere und interpretierbarere Methode zur Erkennung und Klassifizierung von Fehlern in komplexen Automobilsystemen bereitzustellen.

Durch systematische Untersuchung und umfangreiche Experimente wurde ein zweistufiger Verarbeitungsansatz entwickelt und validiert. Dabei werden potenzielle Anomalien zunächst mittels Rekonstruktionsfehleranalyse identifiziert, bevor zur Differenzierung von Fehlermustern das Clustering-Verfahren angewendet wird. Dieser Ansatz verbessert nicht nur die Recheneffizienz, sondern erhöht auch die Klarheit bei der Fehlererkennung. Das Framework wurde mit ungesehenen Testdaten sowohl aus der Simulink Model-in-the-Loop-Umgebung (MIL) als auch von der SCALEXIO Hardware-in-the-Loop-Plattform (HIL) über verschiedene Fahrszenarien hinweg evaluiert, wobei es seine Robustheit und Generalisierbarkeit unter Beweis stellte.

Die wichtigsten Beiträge dieser Forschung umfassen: eine innovative CNN-LSTM Hybridarchitektur, welche bei der Erkennung von Automobilsignalfehlern eine überlegene Leistung mit einer Genauigkeit von 90,54% und einem F1-Score von 0,926 erzielte; ein optimiertes K-Means-Clustering mit fehlerbasierter Auswahl des K-Wertes, das den Davies-

Bouldin-Index bei Mehrfehlerszenarien um 38,6% verbesserte, eine interpretierbarere Differenzierung der Fehlermuster ermöglichte und die Verarbeitungszeit um 36,4% reduzierte; eine starke szenarienübergreifende Generalisierungsfähigkeit, die es erlaubt, ein einziges, auf gemischten Szenariendaten trainiertes Modell unter verschiedenen Betriebsbedingungen einzusetzen; sowie ein umfassendes, intelligentes B2B-Testframework, das Ingenieuren automatisierte Merkmalsextraktion, statistisch optimierte Schwellenwertauswahl sowie eine klare Visualisierung und Interpretation von Fehlermustern bietet.

Experimentelle Ergebnisse zeigen, dass das vorgeschlagene CNN-LSTM-DAE-Modell mit optimiertem K-Means-Clustering-Framework eine ausgeprägte Robustheit in stark verrauschten Umgebungen aufweist, sowohl einzelne als auch multiple Fehlerszenarien zuverlässig gruppiert, eine stabile Leistung über verschiedene Fahrszenarien hinweg gewährleistet und dabei sämtliche Verarbeitungszeiten innerhalb der Anforderungen an die Rechenkosten hält. Diese Forschung stellt somit einen neuartigen intelligenten Ansatz für den Back-to-Back-Test von Automobilsoftware dar, welcher die Grenzen herkömmlicher Methoden überwindet und eine effizientere und genauere Lösung zur Sicherstellung der Systemsicherheit und -zuverlässigkeit bietet.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Research Background and Significance

The rapid development of the automotive industry toward electrification, intelligence, and connectivity presents unprecedented challenges in software system complexity. Modern automotive systems heavily rely on software to implement various functions, from basic control to advanced driver assistance features. Broy et al.[1] point out that today's high-end vehicles contain up to 100 million lines of code, a complexity comparable to that of advanced aviation systems.

The growing complexity of automotive software systems has led to significant verification challenges. According to research by McKinsey & Company,[2] software-related recall incidents have increased by approximately 30% over the past five years, causing estimated annual losses exceeding $600 million to the automotive industry. Furthermore, as Charette[3] demonstrates, modern premium vehicles typically integrate more than 100 electronic control units (ECUs), whose complex interactions and interdependencies substantially increase the difficulty of verification. Pretschner et al.[4] emphasize how these intricate interactions and real-time requirements increase testing challenges exponentially. Through systematic analysis, Staron[5] identifies software not only as the core driver of innovation in modern vehicles but also as a primary challenge for ensuring functional safety.

Against this backdrop, ISO 26262[6] imposes stringent requirements for software system verification. Research by Altinger et al.[7] shows that with the rapid development of Advanced Driver-Assistance Systems (ADAS) and autonomous driving technologies, the complexity and criticality of software verification have significantly escalated. Among various verification methods, back-to-back (B2B) testing has received widespread attention due to its capability to ensure consistency between different implementation stages, such as model-in-the-loop (MIL) and hardware-in-the-loop (HIL). ISO 26262 explicitly recommends B2B testing due to its structured approach in validating functional consistency across multiple testing environments.[6]

However, traditional B2B testing primarily relies on fixed thresholds to determine test result consistency, an approach facing severe limitations when dealing with complex, dynamically changing systems. Additionally, the manual analysis typically involved in

comparing the outputs of simulation models with those from real target hardware is time-consuming, error-prone, and lacks scalability, further complicating effective fault diagnosis and verification in automotive systems. Zhang et al.[8] demonstrate that in complex automotive scenarios, this fixed-threshold method often results in a high false positive rate, sometimes as high as 30%, which significantly reduces the efficiency and reliability of testing processes.

Additionally, traditional approaches struggle to effectively handle multi-variable interactions, especially prevalent in real-time automotive software. Kim and Lee[9] highlight that these methods inadequately capture dynamic interdependencies among system variables, leading to either missed detections or excessive false alarms. Furthermore, Shin et al.[10] indicate that although dynamic thresholding approaches have been introduced, their adaptability and real-time responsiveness remain insufficient for modern automotive systems, which have stringent real-time constraints defined by ISO 26262.

Furthermore, rule-based verification methods, such as those proposed by Khan et al.,[11] encounter severe scalability and maintainability issues, as each new vehicle variant requires substantial modifications to the existing ruleset. Baraldi et al.[12] emphasize the scalability problems inherent in rule-based verification systems, highlighting the extensive adjustments needed for each new system iteration.

Recent works involving machine learning and deep learning, such as Liu et al.[13] and Wang et al.,[14] demonstrate potential in learning complex system behaviors, yet these approaches still face limitations regarding real-time applicability and robustness. Liu et al.[13] demonstrate deep learning's promising capability in automotive systems but rely heavily on extensive manual feature engineering. Wang et al.[14] stress the need for automated and robust feature extraction methods to cope effectively with dynamic and complex system characteristics encountered during real-time validations.

Considering these critical issues, it becomes evident that a new, intelligent approach is necessary to overcome these inherent limitations and ensure robust validation of automotive software systems.

1.2 Related Work

A comprehensive review of existing research in automotive software testing and fault detection reveals several areas for potential improvement, particularly in automated feature extraction and real-time analysis capabilities. This section explores in depth back-to-back testing, applications of deep learning in automotive testing, and the role of

cluster analysis in fault diagnosis, establishing the theoretical foundation for the methods proposed in this research.

1.2.1 Back-to-Back Testing and Automotive Functional Safety

As automotive electronic systems become increasingly complex, functional safety verification becomes increasingly important. Through an in-depth study of the ISO 26262 standard, Bringmann and Kramer[15] demonstrate that the development of automotive electronic systems is driving continuous evolution in software verification requirements. Research by Conrad et al.[16] shows that a comprehensive software verification system should cover three dimensions: functional requirement verification, performance requirement verification, and safety mechanism verification, accounting for approximately 40%, 35%, and 25% of total verification work, respectively. Furthermore, analysis by Schauffele and Zurawka[17] reveals the stringent timing requirements of modern automotive electronic systems: response times for critical functions must be controlled within 10ms, fault detection time must not exceed 100ms, and safety mechanism activation time must remain below 50ms.

Back-to-back testing is an important verification method used to ensure functional consistency between different implementation versions of a system. In the automotive industry, this method has been widely applied in multiple environments, including model implementation verification, cross-platform consistency, version upgrade verification, and consistency verification between different development stages (such as model-in-the-loop, software-in-the-loop, and hardware-in-the-loop). Traditional back-to-back testing methods primarily rely on setting fixed thresholds to compare reference results with test results. However, this approach faces significant challenges in sensitivity balance, multi-variable interaction, dynamic behavior adaptation, and real-time constraints.

The implementation of back-to-back testing faces several key challenges. First, traditional statistical analysis methods show significant limitations when dealing with complex automotive systems. Although Matinnejad et al.[18] successfully applied dynamic threshold methods to simple automotive controller testing, these methods are insufficient for modern vehicle systems with complex variable interactions. Wang et al.[19] further demonstrate that traditional threshold-based methods cannot effectively handle coupling effects between multiple variables, leading to unreliable results in practical applications.

Rule-based verification methods attempt to address these limitations by formalizing expert knowledge. However, as Li et al.[20] discovered in their comprehensive study

of engine control system verification, maintaining and updating rule sets becomes increasingly challenging as system complexity grows. Work by Zander et al.[21] highlights a fundamental issue: rule-based systems require extensive modifications for each new vehicle variant, making them unsuitable for large-scale deployment.

1.2.2 Applications of Deep Learning in Automotive System Analysis

In recent years, deep learning has shown enormous potential in the field of automotive system analysis. O'Kelly et al.[22] explored applications of deep neural networks in automatic feature extraction from vehicle data, demonstrating significant improvements compared to traditional manual feature engineering. The automatic comparison framework proposed by Liu and Yang[23] shows promise in handling complex system behaviors, providing new perspectives for automotive testing and validation.

Deep autoencoders show particular advantages in analyzing automotive system behaviors. Wen et al.[24] demonstrated their effectiveness in learning compact representations of complex vehicle signals, achieving more robust performance compared to traditional feature extraction methods. Work by Zheng and Li[25] further validated the advantages of deep learning methods in capturing time dependencies and system dynamics, especially when processing back-to-back test data.

However, current applications of deep learning in automotive testing still face several limitations. As shown by Choi and Noh,[26] many existing approaches lack efficiency when processing large volumes of test data. Weber and Jouffe[27] highlight the challenge of maintaining model robustness across different operating conditions. Additionally, Sun et al.[28] point out the difficulties in interpreting and validating deep learning models in safety-critical applications, which poses barriers to widespread adoption in the automotive industry.

Deep learning models have improved traditional back-to-back testing methods in multiple aspects, including automatic feature extraction, dynamic threshold adaptation, anomaly detection, and temporal correlation analysis. However, applying deep learning in back-to-back testing still faces challenges, particularly in model transparency and interpretability, as well as handling complex and limited training data. This research aims to address these challenges by developing a framework that combines deep learning and clustering analysis for back-to-back test result analysis, enhancing the efficiency and accuracy of automotive software testing.

1.2.3 Applications of Cluster Analysis in Fault Diagnosis

Several researchers have made significant contributions to applying clustering techniques in fault diagnosis. Zhao et al.[29] developed a two-stage clustering approach for power system fault diagnosis, combining density-based clustering with supervised classification to achieve 95% accuracy in identifying fault types. Similarly, Khan et al.[30] integrated K-means clustering with deep neural networks for engine fault diagnosis, demonstrating a 30% improvement in fault detection rate compared to conventional methods.

In automotive systems, MacGregor and Kourti[31] pioneered the application of multivariate statistical techniques for process fault detection, while Chandola et al.[31] systematically reviewed anomaly detection techniques including clustering-based approaches. Chen and Liu[20] implemented a hierarchical clustering method for electric vehicle battery fault diagnosis, achieving 92% classification accuracy across multiple fault scenarios.

For multi-fault scenarios, which are common in automotive systems, Liu et al.[23] proposed a subspace clustering approach that outperformed traditional methods by 25% in identification accuracy. This approach specifically addressed feature interactions and fault masking effects. Yan and Yu[32] developed an enhanced DBSCAN algorithm for automotive sensor networks, capable of effectively identifying multiple concurrent faults, improving both analysis efficiency and accuracy.

The combination of deep learning feature extraction with clustering analysis represents a promising direction in automotive fault diagnosis. Zhang et al.[8] demonstrated that features extracted by deep autoencoders significantly improved the performance of subsequent K-means clustering for engine fault classification. Similarly, Peng et al.[13] showed that combined CNN-extracted features with Gaussian Mixture Models could effectively identify subtle fault patterns in transmission systems that traditional methods missed.

In automotive systems, multiple faults occurring simultaneously are not uncommon, which presents special challenges for cluster analysis, including feature interactions, cluster shape complexity, fault masking effects, and determining the number of clusters. To address these challenges, researchers have proposed various improved methods, such as hierarchical clustering, subspace clustering, and ensemble-based clustering methods. This research will particularly focus on clustering strategies applicable to multiple fault scenarios to improve diagnostic accuracy in complex fault situations.

An overview of the related work is presented in Table 1.1.

Table 1.1: Overview of the related works

Reference	Techniques	Application Domain	Dataset	Feature Learning	Remarks
[15]	Statistical threshold	Engine control B2B	Simulation data	Manual	Simple thresholds, limited features
[18]	Dynamic threshold	ECU B2B testing	HIL test data	Manual	Manual threshold setting
[20]	Rule-based	Powertrain B2B	HIL data	Expert knowledge	Limited adaptability
[22]	Deep learning	ADAS B2B testing	Real test data	Automated	Poor performance efficiency
[24]	Autoencoder	Vehicle control B2B	MIL + HIL	Automated	High computational cost
[26]	Traditional ML	Sensor B2B test	Real-time data	Semi-automated	Limited feature extraction
Proposed work	**Denoising Deep Autoencoder**	**Automotive B2B**	**MIL + HIL**	**Fully automated**	**Efficient processing**

1.3 Research Objectives

The main objective of this research is to develop a machine learning-based intelligent framework for automated analysis of back-to-back test results in automotive software systems. Specific objectives include:

1. Designing an efficient deep learning architecture for analysis of B2B test data, enhancing the automation and accuracy of testing processes.

2. Developing an automated feature extraction method based on deep autoencoders, reducing dependence on manual feature engineering, and enhancing robustness to complex multi-variable interactions.

3. Constructing a comprehensive framework capable of identifying and classifying both single and multiple faults, significantly improving fault detection accuracy, interpretability, and adaptability across diverse testing scenarios.

4. Validating the proposed framework's effectiveness and robustness under different driving scenarios and vehicle models, particularly its performance in high-noise environments and dynamically complex driving conditions.

5. Developing and evaluating different clustering methods to classify fault patterns accurately, providing automotive engineers with actionable insights and tools for timely fault diagnosis and resolution.

1.4 Thesis Structure

The remainder of this thesis is structured as follows: Chapter 2 provides a detailed theoretical foundation, focusing on the theoretical background of all techniques, methods, tools, systems and concepts used in the work. Chapter 3 introduces the proposed intelligent framework in-depth, describing the employed methodologies, algorithms, and architectural choices. Chapter 4 describes the experimental setup, including hardware and software configurations, test scenarios, and model training processes. Chapter 5 analyzes experimental results and discusses the findings in the context of performance, reliability, and real-time capabilities. Finally, Chapter 6 concludes the thesis by summarizing key contributions and outlining directions for future research.

2 Theoretical Background

This chapter details the theoretical foundations of all techniques, methods, tools, systems, and concepts used in this research, including ISO 26262[6] functional safety standard, back-to-back testing methodology, deep learning techniques, clustering analysis algorithms, hyperparameter optimization methods, and automotive system development and testing environments. A comprehensive understanding of these fundamentals provides the theoretical basis for the machine learning-based back-to-back testing framework proposed in subsequent chapters.

2.1 ISO 26262 Functional Safety Standard

2.1.1 Standard Overview

ISO 26262 is an international functional safety standard specifically designed for road vehicle electrical/electronic systems, first published by the International Organization for Standardization (ISO) in 2011 and updated to the second version in 2018.[6] This standard is derived from the generic IEC 61508 standard but tailored to address the specific needs of the automotive industry.[33] As automotive electronic systems have grown increasingly complex, with software playing an ever more critical role in vehicle functionality, ISO 26262 has emerged as a comprehensive framework and methodology for ensuring the functional safety of automotive electronic systems.

ISO 26262 adopts a risk-based approach to assess safety requirements and introduces the concept of Automotive Safety Integrity Levels (ASIL).[34] ASIL is categorized into four levels: A, B, C, and D, with D representing the highest safety requirements. Each ASIL level corresponds to different safety measures and verification requirements that must be met throughout system development.[3]

2.1.2 V-Model Development Process

ISO 26262 recommends the V-model as the standard process for automotive system development,[1] as illustrated in Figure 2.1. The V-model divides the system development process into a development phase on the left side and a verification and validation phase on the right side, forming a ”V” shape.

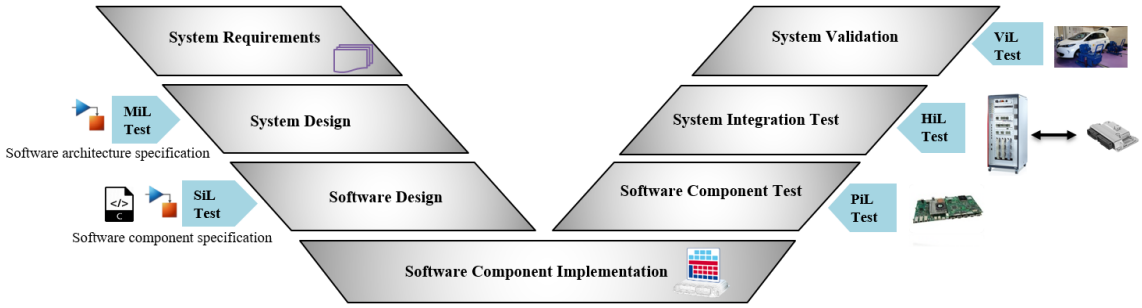


Figure 2.1: V-model development process[35]

The left side of the V-model starts with system requirements, progressing through system design, software design, to software component implementation, gradually refining and decomposing requirements; the right side consists of verification activities corresponding to each development stage on the left, including software component testing, software integration testing, system integration testing, and system verification.[17] In this process, each verification stage is associated with its corresponding development stage, ensuring that the final system meets the initial requirements.

2.1.3 Software Testing Requirements

ISO 26262 imposes stringent requirements on software testing, particularly for the verification of safety-critical functions.[4] According to Part 6 of the standard, software verification should include multiple stages, such as software unit testing, software integration testing, and hardware-software integration testing.[6] For different ASIL levels, the standard requires different testing methods and test coverage standards.

Notably, ISO 26262 emphasizes the importance of verifying software in a real-time environment.[15] For automotive electronic systems, real-time performance is a key requirement. The standard stipulates that for systems at ASIL C and D levels, the response time of critical functions must be strictly controlled within a specified range, typically 10–100 milliseconds.[36]

2.2 Back-to-Back Testing (B2B)

2.2.1 Definition and Principles

Back-to-Back Testing (B2B) is a comparative testing method used to verify consistency between different implementation versions.[37] In this approach, the same test inputs are applied simultaneously to a reference model and the system under test, and the outputs from both are compared.[38] If the outputs are similar within an acceptable margin of error, the test is considered passed; otherwise, the system under test may have defects.

The basic principle of B2B testing is to use a verified reference model as a "Golden Standard" to generate test oracles based on the responses of the reference model.[39] This approach is particularly suitable for verifying complex systems, especially when system behavior is difficult to fully describe through traditional specifications.

Wang[40] notes that B2B testing is also referred to as comparative testing, equivalence checking, or conformance testing in different literature. Despite the variation in terminology, the core idea remains similar: verifying system correctness by comparing outputs from different implementations.

2.2.2 Testing Methods during Model-Driven Development

B2B testing is a crucial verification technique in Model-Driven Development (MDD).[5] In the MDD paradigm, systems begin as abstract models and are progressively transformed into concrete implementations. Throughout this process, B2B testing is used to ensure functional consistency in each transformation.[41]

As shown in Figure 2.1, B2B testing can be applied across multiple stages of the development process:

- Model-in-the-Loop (MiL): At this stage, both the control algorithm and plant model are implemented in a simulation environment.[42] MiL testing verifies the correctness of the control algorithm design at the conceptual level.
- Software-in-the-Loop (SiL): The control algorithm is transformed into software code but still executes in a host environment.[42] SiL testing verifies the correctness of the code generation process.
- Processor-in-the-Loop (PiL): The software executes on the target processor, but the plant remains a model.[43] PiL testing focuses on code performance on the specific processor.

- Hardware-in-the-Loop (HIL): The software runs on actual ECU hardware, connected to a real-time simulation system.[44] HIL testing is the closest to the actual operating environment.
- Vehicle-in-the-Loop (ViL): Final verification on an actual vehicle.[43]

At each stage, B2B testing can be used to compare behavioral consistency between the current implementation and previous stages. For example, conducting B2B testing between SiL and MiL can verify whether the generated code behaves consistently with the original model.

2.2.3 Status in ISO 26262

The ISO 26262 standard explicitly recommends B2B testing as one of the verification methods.[6] Particularly in Part 6 of the standard, B2B testing is recommended for verifying correctness at the stages of software units, software integration, and hardware-software integration.

For different ASIL levels, the recommendation level for B2B testing varies. For ASIL A and B levels, B2B testing is "recommended" (+); while for ASIL C and D levels, B2B testing is "highly recommended" (++).[21] This indicates that B2B testing is a crucial verification method for systems with high safety integrity requirements.

Notably, ISO 26262 particularly emphasizes the role of B2B testing in verifying the temporal precision and accuracy of safety mechanisms.[7] This is because in real-time systems, not only is functional correctness important, but the correctness of temporal behavior is equally critical.

2.2.4 Traditional B2B Test Result Analysis Methods and Their Limitations

Traditional B2B test result analysis primarily relies on simple threshold comparisons.[45] In this approach, a fixed threshold is set, and if the difference between the output of the system under test and the reference model exceeds this threshold, the test is considered failed.[46]

Wang[40] identifies several key challenges faced by traditional threshold-based B2B test result analysis:

- **Sensitivity Balance**: Fixed thresholds struggle to balance between detecting genuine faults and tolerating normal variations.[32] Setting the threshold too low leads to false positives, while setting it too high might result in missed detections.
- **Multi-variable Interaction**: Traditional methods struggle to handle coupling effects between multiple variables in complex systems.[30] When multiple signals change simultaneously, methods based on single-variable thresholds may not capture their combined impact.
- **Dynamic Behavior Adaptation**: Automotive systems exhibit different dynamic behaviors under different operating conditions.[47] Fixed thresholds struggle to adapt to these dynamic changes.
- **Efficiency Constraints**: Traditional methods typically require offline analysis, making them inefficient when processing large volumes of data.[28] ISO 26262 requires fault detection time not to exceed 100 milliseconds for critical safety functions.

These limitations make traditional B2B test result analysis methods challenging for verifying modern complex automotive software systems, necessitating more advanced solutions.[8]

2.2.5 Threshold Selection Theory for Anomaly Detection

Threshold selection plays a crucial role in anomaly detection systems, directly impacting detection performance. Statistical approaches to threshold selection are particularly relevant in the context of B2B testing for automotive systems. Chandola et al.[31] identify several key approaches to threshold selection in anomaly detection:

- **Probability Distribution-Based Approach**: Assuming the error follows a known distribution (often approximated as Gaussian for many engineering applications), thresholds can be set based on statistical confidence intervals. Typically, 95th or 99th percentiles (corresponding to approximately 2 or 3 standard deviations in a normal distribution) are chosen to balance detection sensitivity with false alarm rates.
- **Extreme Value Theory**: For systems where anomalies are rare events in the extreme tails of distributions, extreme value theory provides a mathematical framework for modeling and setting thresholds.

- **Empirical Quantile-Based Approach**: This non-parametric approach makes no assumptions about underlying distributions and directly uses quantiles from training data. The 99th percentile is theoretically optimal for anomaly detection when anomalies constitute approximately 1% of the overall distribution.

For autoencoder-based anomaly detection specifically, An and Cho[48] show that reconstruction error distributions tend to exhibit right-skewed characteristics, making high percentile thresholds (e.g., 99th percentile) particularly appropriate. This right-skewed property arises from the autoencoder's tendency to reconstruct normal data well while producing significantly higher errors for anomalous inputs.

The theoretical justification for using the 99th percentile specifically comes from achieving an optimal balance between precision and recall in anomaly detection. In domains where false alarms are costly but missed detections have higher consequences (which is typical in automotive safety applications), thresholds at the 99th percentile tend to optimize the F1-score, providing the best trade-off between detection capability and false alarm rate.

2.3 Deep Learning Techniques

2.3.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs), initially proposed by LeCun et al.,[49] are a special class of deep neural networks designed for processing data with spatial structure. Although CNNs were originally applied in image recognition, their powerful local feature extraction capability has also shown excellence in time series analysis.[50]

Basic Principles and Architecture The core components of CNNs include convolutional layers, pooling layers, and fully connected layers.[51] Convolutional layers extract local features from input data through sliding convolution kernels, pooling layers perform downsampling to reduce data dimensionality and extract prominent features, and fully connected layers integrate the extracted features for final prediction.

The mathematical representation of a CNN can be summarized as:

$$h^{l+1} = \sigma(W^l * h^l + b^l),$$

where h^l represents the feature map at layer l, W^l is the convolution kernel, b^l is the bias term, $*$ denotes the convolution operation, and σ is a non-linear activation function.[51]

Applications in Time Series Analysis Although CNNs were initially designed for two-dimensional image processing, one-dimensional CNNs also have wide applications in time series analysis.[52] In time series analysis, CNNs use one-dimensional convolution filters sliding along the time dimension to effectively capture local temporal patterns and features.

The hierarchical structure of one-dimensional CNNs enables them to automatically learn feature abstractions at different levels, eliminating the need for manual feature engineering.[53] This is particularly valuable for analyzing complex automotive system data, as manually designing features is often time-consuming and may not cover all possible patterns.

2.3.2 Long Short-Term Memory Networks (LSTM)

Recurrent Neural Networks (RNNs) are neural network architectures designed for processing sequence data, but traditional RNNs face the long-term dependency problem.[54] Long Short-Term Memory (LSTM) networks, proposed by Hochreiter and Schmidhuber,[55] aim to solve the vanishing gradient problem in traditional RNNs, enabling more effective capture of long-term dependencies.

Basic Structure and Working Principles The core of LSTM is a memory cell with gating mechanisms, including a forget gate, an input gate, and an output gate.[56] These gates control information flow, allowing the network to selectively remember or forget information.

The mathematical representation of an LSTM cell is:

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

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

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C),$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t,$$

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

$$h_t = o_t * \tanh(C_t),$$

where f_t, i_t, and o_t are the outputs of the forget gate, input gate, and output gate respectively, C_t is the cell state, h_t is the hidden state, σ is the sigmoid function, and

tanh is the hyperbolic tangent function.[56]

Advantages in Sequence Learning LSTM excels in capturing long-term dependencies in time series data.[57] Compared to traditional RNNs, LSTM better handles the vanishing gradient problem, enabling it to learn patterns over longer time spans.

In automotive system analysis, LSTM has been widely applied to sensor data prediction, fault diagnosis, and anomaly detection.[58] Malhotra et al.[59] demonstrated the application of LSTM in mechanical system failure prediction, proving its ability to accurately capture equipment degradation patterns.

Another advantage of LSTM is its ability to handle variable-length sequences. This is particularly important for automotive system analysis, as vehicle operational data typically contains driving scenarios of different lengths.

2.3.3 Deep Autoencoders

Autoencoders are a type of unsupervised learning neural network designed to learn effective representations of input data.[60] Deep autoencoders implement more powerful feature extraction capabilities through multi-layer neural networks, particularly suitable for dimensionality reduction and feature learning of high-dimensional data.[61]

Encoder-Decoder Architecture Autoencoders consist of two parts: an encoder and a decoder.[62] The encoder maps input data to a low-dimensional latent space, while the decoder attempts to reconstruct the original input from this latent representation.

The mathematical representation of an autoencoder is:

$$z = f_\theta(x), \quad \hat{x} = g_\phi(z),$$

where x is the input data, z is the latent representation, $\hat{x}$ is the reconstructed output, f_θ is the encoder function, and g_ϕ is the decoder function.[62]

The training objective is to minimize the reconstruction error:

$$L(x, \hat{x}) = \|x - \hat{x}\|^2.$$

By minimizing reconstruction error, the autoencoder is forced to learn the most salient features of the input data.[51]

Denoising Autoencoders Denoising Autoencoders (DAE) are a variant of autoencoders specifically designed to enhance robustness to noise.[63] During training, noise is artificially added to the input data, but the decoder still attempts to reconstruct the original noise-free data.

The training process of a DAE can be represented as:

$$\tilde{x} = x + \epsilon, \quad z = f_\theta(\tilde{x}), \quad \hat{x} = g_\phi(z), \quad L(x, \hat{x}) = \|x - \hat{x}\|^2,$$

where $\tilde{x}$ is the noise-corrupted input, and ϵ is the noise.[63]

This training approach makes the autoencoder focus more on the structural features of the data rather than simply memorizing the input.[64] This is particularly valuable for automotive system analysis, as sensor data typically contains various types of noise.

Applications in Anomaly Detection Autoencoders have widespread applications in anomaly detection.[65] The basic idea is to train an autoencoder on normal data and then calculate reconstruction errors. When encountering anomalous data, since the autoencoder has not seen such patterns before, the reconstruction error is typically larger.[48]

In automotive system analysis, autoencoders are used to detect sensor faults, system anomalies, and abnormal driving behavior.[66] Malhotra et al.[67] demonstrated a time series anomaly detection method based on LSTM autoencoders, capable of effectively identifying anomalous patterns in mechanical systems.

2.3.4 Hybrid Deep Learning Architectures

Hybrid deep learning architectures combine the advantages of different types of neural networks to create more powerful models.[68] This approach is particularly suitable for data with complex spatiotemporal characteristics, such as automotive system data.

CNN-LSTM Hybrid Models CNN-LSTM hybrid models combine the local feature extraction capability of CNNs with the long-term dependency modeling capability of LSTMs.[69] In this architecture, CNN layers first extract local features from the input data, then LSTM layers process these feature sequences, capturing temporal dependencies.

The mathematical representation of a CNN-LSTM hybrid model is:

$$H = \text{CNN}(X), \quad O = \text{LSTM}(H),$$

where X is the input data, H is the features extracted by CNN, and O is the output of LSTM.

This architecture has shown excellence in various applications, including video analysis, sensor data processing, and time series prediction.[70] In automotive system analysis, CNN-LSTM hybrid models are used for driving behavior prediction, fault diagnosis, and system monitoring.

CNN-LSTM Autoencoders CNN-LSTM autoencoders combine CNN-LSTM hybrid models with the autoencoder architecture.[71] In this architecture, a CNN-LSTM is used as the encoder, and another symmetric LSTM-CNN structure serves as the decoder.

This architecture is particularly suitable for time series anomaly detection, as it can simultaneously capture both local patterns and long-term dependencies in data.[72] In automotive system analysis, CNN-LSTM autoencoders can effectively detect complex anomalous patterns, such as multi-sensor collaborative faults and system performance degradation.

2.3.5 Other Machine Learning Models

Besides the deep learning models mentioned above, several traditional machine learning methods are also applied to automotive system analysis and fault detection. These methods have their own advantages in specific scenarios, especially in situations with limited computational resources or small data volumes.

Principal Component Analysis (PCA) Principal Component Analysis is a commonly used dimensionality reduction technique that projects high-dimensional data into a lower-dimensional space through linear transformation, while preserving the main variations in the data.[73] PCA can be used for feature extraction, dimensionality reduction, and anomaly detection.

In anomaly detection, PCA identifies anomalies by calculating the reconstruction error of data points to the principal component space.[31] If the reconstruction error exceeds a certain threshold, the data point is considered anomalous.

Support Vector Machines (SVM) Support Vector Machine is a powerful classification and anomaly detection algorithm that seeks the maximum margin hyperplane to separate different classes of data.[74] In anomaly detection, One-Class SVM learns the distribution boundary of the training data, identifying data points outside this boundary as anomalies.[75]

K-Nearest Neighbors (KNN) K-Nearest Neighbors algorithm is an instance-based learning method that classifies or performs regression based on the similarity of a test instance to the K nearest instances in the training set.[76] In anomaly detection, if a data point's average distance to its K nearest neighbors exceeds a certain threshold, it is considered anomalous.[77]

Decision Trees (DT) Decision Tree is a non-parametric supervised learning method used for classification and regression.[78] Decision trees create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. In a decision tree, each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label or a regression value.

The main advantages of decision trees include their simplicity and interpretability, requiring little data preparation, and ability to handle both numerical and categorical data.[79] In anomaly detection, decision trees can be trained to classify normal versus anomalous patterns based on various feature combinations, providing transparent decision paths that are easily understandable by engineers.

However, decision trees also have limitations such as their tendency to overfit complex datasets and potential instability where small variations in the data may result in a completely different tree. In automotive system analysis, decision trees are often combined with other algorithms in ensemble methods (such as Random Forests) to improve robustness and detection accuracy.

2.4 Clustering Analysis Algorithms

2.4.1 K-means Clustering

K-means is one of the most basic and widely used clustering algorithms.[80] It groups data points into K clusters through iterative optimization, such that each data point belongs to the cluster with the nearest center.

Basic Principles The basic steps of the K-means algorithm are:

1. Randomly initialize K cluster centers
2. Assign each data point to the nearest cluster center
3. Recalculate the center of each cluster
4. Repeat steps 2–3 until the cluster centers stabilize or the maximum number of iterations is reached

The objective function of K-means is to minimize the sum of squared distances from all data points to their respective cluster centers:

$$J = \sum_{i=1}^{n} \sum_{j=1}^{k} w_{ij} \|x_i - \mu_j\|^2,$$

where x_i is a data point, μ_j is a cluster center, and w_{ij} is an indicator variable, which is 1 if point i belongs to cluster j and 0 otherwise.

Advantages and Limitations The main advantages of K-means include its simple implementation, high computational efficiency, and easily interpretable results.[81] These make it a preferred clustering algorithm in many application domains.

However, K-means also has several limitations:

- Requires pre-specifying the number of clusters K
- Sensitive to initial cluster centers
- Only suitable for convex-shaped clusters
- Sensitive to outliers

Despite these limitations, K-means is still widely applied in automotive system analysis, particularly in fault pattern recognition and system behavior classification.

2.4.2 Density-Based Clustering Algorithms (DBSCAN)

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm originally proposed by Ester et al.[82] Unlike K-means, DBSCAN does not require pre-specifying the number of clusters and can discover clusters of arbitrary shapes.

Basic Principles DBSCAN is based on two key parameters: ϵ (neighborhood radius) and MinPts (minimum number of points). Its basic concepts include:

- Core Point: A point that has at least MinPts points within its ϵ-neighborhood
- Border Point: A point that is within the ϵ-neighborhood of a core point but is not itself a core point
- Noise Point: A point that is neither a core point nor a border point

The basic step of DBSCAN is to start from an unvisited core point and continuously expand the cluster until all reachable points are included in the cluster.[82]

Applications in Anomaly Detection DBSCAN is particularly valuable in anomaly detection because it can naturally identify noise points.[83] In automotive system analysis, DBSCAN is used to detect abnormal driving behavior, sensor faults, and system anomalies.

Compared to K-means, DBSCAN performs better in handling irregularly shaped clusters and identifying outliers.[84] However, DBSCAN is sensitive to parameter selection and may face the "curse of dimensionality" in high-dimensional spaces.

2.4.3 Gaussian Mixture Models (GMM)

Gaussian Mixture Model (GMM) is a probability-based clustering method that assumes data is composed of multiple Gaussian distributions.[85] Unlike hard clustering methods, GMM assigns each data point a probability of belonging to each cluster.

Mathematical Foundations The mathematical representation of GMM is:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k),$$

where π_k is the weight of the kth Gaussian distribution, and $\mathcal{N}(x \mid \mu_k, \Sigma_k)$ is a Gaussian distribution with mean μ_k and covariance matrix Σ_k.

GMM is typically trained using the Expectation-Maximization (EM) algorithm,[86] which iteratively executes two steps:

1. E-step: Calculate the posterior probability of each data point belonging to each Gaussian distribution

2. M-step: Update distribution parameters to maximize likelihood

Applications in Fault Diagnosis GMM has various applications in fault diagnosis, particularly in dealing with fault modes with probabilistic characteristics.[31] In automotive system analysis, GMM is used for engine fault diagnosis, sensor fault classification, and system state monitoring.

The main advantages of GMM include providing probabilistic outputs and adapting to different data distribution shapes. However, GMM also faces challenges such as parameter estimation difficulties and high computational complexity.

2.4.4 Fuzzy C-means Clustering (FCM)

Fuzzy C-means Clustering (FCM) is an extension of the K-means algorithm, allowing data points to belong to multiple clusters with different membership degrees.[87] This soft clustering method is particularly suitable for datasets with fuzzy boundaries.

The objective function of FCM is:

$$J_m = \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^m \|x_i - c_j\|^2,$$

where u_{ij} is the membership degree of point x_i belonging to cluster j, $m > 1$ is the fuzziness coefficient, and c_j is the cluster center.

FCM excels in handling fuzzy boundaries and gradual fault modes, making it particularly suitable for analyzing progressive faults in automotive systems. Compared to traditional K-means, FCM can more accurately capture the transition process from normal state to anomalous state.[87]

2.4.5 Cluster Validity Assessment

Selecting the optimal number of clusters and evaluating the quality of clustering results are critical steps in cluster analysis. **Internal evaluation metrics** measure how well the clusters fit the dataset without requiring external labels. Three widely used internal metrics in fault diagnosis and general cluster analysis are:

Davies-Bouldin Index (DBI)[88] The Davies-Bouldin Index evaluates intra-cluster similarity and inter-cluster separation. A smaller DBI indicates better clustering (i.e.,

lower intra-cluster dispersion and greater between-cluster separation). Formally, for k clusters:

$$\text{DBI} = \frac{1}{k}\sum_{i=1}^{k}\max_{j\neq i}\left(\frac{\sigma_i + \sigma_j}{d(\mu_i, \mu_j)}\right),$$

where:

- σ_i is the average distance of all points in cluster i to the cluster center μ_i.
- μ_i and μ_j are the centers (centroids) of clusters i and j, respectively.
- $d(\mu_i, \mu_j)$ is the distance between the two cluster centers (often Euclidean distance).

Intuitively, DBI penalizes clusters that are both large (i.e., have large intra-cluster scatter σ_i) and too close to other clusters.

Silhouette Coefficient[89] The Silhouette Coefficient measures how similar a data point is to the points in its own cluster compared to points in other clusters. It ranges from -1 to $+1$, where a higher value signifies better-defined clusters. For each point i:

$$S(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}},$$

where:

- $a(i)$ is the average distance from point i to other points in the same cluster.
- $b(i)$ is the smallest average distance from point i to points in any other cluster.

The overall Silhouette Coefficient for the clustering result is typically the mean $S(i)$ across all points. Values near $+1$ mean data points are significantly closer to their own cluster than to neighboring clusters, while negative values indicate points likely assigned to the wrong cluster.

Calinski-Harabasz Index (CHI)[90] Also called the Variance Ratio Criterion, the Calinski-Harabasz Index quantifies how distinct clusters are relative to the variance within each cluster. A larger CH value indicates better-defined clusters:

$$\text{CH} = \frac{\frac{B_k}{k-1}}{\frac{W_k}{N-k}},$$

where:

- B_k is the between-cluster dispersion (sum of squared distances between each cluster center and the global mean, weighted by cluster size).
- W_k is the within-cluster dispersion (sum of squared distances of each point to its own cluster center).
- k is the number of clusters, and N is the total number of data points.

In automotive fault-diagnosis applications—especially when distinguishing among multiple fault modes—these metrics help confirm whether the identified clusters represent genuinely different fault patterns (low DBI, high Silhouette, and high CH) and provide engineers with greater confidence in the clustering results.

2.5 Hyperparameter Optimization Methods

2.5.1 Hyperparameter Definition and Importance

Hyperparameters are parameters that need to be set prior to the training process of a model, distinct from model parameters learned through training.[91] Typical hyperparameters include learning rate, batch size, number of layers, number of neurons, activation function type, and so on.

The choice of hyperparameters has a significant impact on model performance.[91] Inappropriate hyperparameter settings may lead to underfitting or overfitting, slow convergence, or even complete failure to converge. In complex deep learning models, the hyperparameter space is typically high-dimensional, making manual tuning of hyperparameters both time-consuming and inefficient. Therefore, automated hyperparameter optimization methods have become increasingly important.

2.5.2 Optuna Framework

Optuna is an automatic hyperparameter optimization framework developed by Akiba et al.,[92] specifically designed for machine learning and deep learning models. Optuna provides flexible APIs and advanced optimization algorithms, particularly suitable for large-scale hyperparameter optimization tasks.

The main features of Optuna include:[92]

- Defining studies consisting of trials: Each trial evaluates a specific set of hyperparameters.

- Supporting various samplers: Including random sampling, Bayesian optimization (TPE), Covariance Matrix Adaptation Evolution Strategy (CMA-ES), etc.
- Pruning mechanism: Automatically stopping poorly performing trials to save computational resources.
- Parallelization support: Executing trials in parallel on multi-core or distributed systems.
- Visualization tools: Providing visualization of hyperparameter importance, relationship between hyperparameters and performance, etc.

Compared to other hyperparameter optimization frameworks, Optuna has a more modern and user-friendly design, particularly suitable for optimizing deep learning models. In practical applications, Optuna has been proven to effectively optimize various complex models, including CNNs, LSTMs, and autoencoders.[92]

2.6 Automotive System Development and Testing Environments

2.6.1 MATLAB/Simulink Development Environment

MATLAB/Simulink is one of the most widely used tools in automotive software development, providing a complete Model-Based Design (MBD) solution.[93]

Model-Based Development Environment Simulink is a graphical programming environment that allows engineers to create system models using blocks and connection lines.[93] This intuitive approach makes the design and verification of complex systems more efficient. In Simulink, systems are represented as interconnections of blocks, with each block representing a specific function or component.

The Model-Based Development (MBD) approach has several advantages in the automotive industry:[94]

- Improved development efficiency: Reducing manual coding work through automatic code generation
- Early error detection: Finding and fixing issues at the design stage

- Promoting team collaboration: Models provide clear system representations, facilitating communication
- Supporting reuse: Model components can be reused across multiple projects
- Simplified verification: Supporting automated testing and verification

Bringmann and Kramer[15] point out that MBD has become the standard method for automotive electronic system development, particularly in terms of compliance with safety standards such as ISO 26262.

ASM Automotive Simulation Models Automotive Simulation Models (ASM) are a set of high-fidelity automotive system simulation environments provided by dSPACE.[95] ASM contains detailed models of multiple subsystems, including engines, transmissions, vehicle dynamics, etc., allowing engineers to develop and test control strategies without requiring an actual vehicle.

The main subsystems provided by ASM include:[95]

- Engine models: Including gasoline and diesel engines, simulating intake systems, combustion processes, exhaust systems, etc.
- Transmission models: Simulating automatic transmissions, manual transmissions, and dual-clutch transmissions
- Vehicle dynamics models: Simulating vehicle motion, suspension systems, tire characteristics, etc.
- Battery and electric motor models: For hybrid and electric vehicle simulation
- Environment models: Simulating road conditions, wind resistance, slopes, etc.

ASM models can be used in MIL, SIL, and HIL environments, supporting the entire development process from concept verification to system integration.

ModelDesk and ControlDesk ModelDesk is a tool for parameterizing and configuring simulation models, particularly suitable for ASM models.[96] It provides an intuitive interface allowing engineers to adjust model parameters, define test scenarios, and analyze simulation results. The main functions of ModelDesk include parameter setting, test automation, and result visualization.

ControlDesk is an experiment execution and data acquisition platform, mainly used for HIL testing and prototype verification.[96] It supports real-time monitoring and control, allowing engineers to modify parameters and observe system behavior during test execution. The main functions of ControlDesk include real-time data visualization, test automation, and fault injection.

dSPACE SCALEXIO System SCALEXIO is a high-performance real-time simulation system provided by dSPACE, specifically designed for HIL testing.[97] The SCALEXIO system adopts a modular architecture that can be expanded according to testing needs, supporting everything from simple single-ECU testing to complex networked system testing.

The main components of SCALEXIO include:[97]

- Processing units: Executing real-time simulation calculations, supporting multi-core processing
- I/O interfaces: Connecting to various sensors, actuators, and communication buses
- Signal conditioning: Providing signal amplification, filtering, and protection functions
- Fault injection units: Simulating various hardware fault scenarios
- Real-time operating system: Ensuring deterministic computation and low latency

ConfigurationDesk and ControlDesk ConfigurationDesk is an engineering tool designed specifically for SCALEXIO systems, used to configure hardware settings, map model signals to I/O channels, and generate the necessary .sdf files for real-time execution.[96] ConfigurationDesk provides a graphical interface for system configuration, significantly simplifying the process of connecting simulation models to hardware components.

ControlDesk is an experiment execution and data acquisition platform, mainly used for HIL testing and prototype verification.[96] It supports real-time monitoring and control, allowing engineers to modify parameters and observe system behavior during test execution. The main functions of ControlDesk include real-time data visualization, test automation, and fault injection. The SCALEXIO system connects to the host PC via Ethernet, using ConfigurationDesk for configuration and ControlDesk for experiment execution.

3 Proposed Method

3.1 Research Framework Overview

This research proposes an intelligent framework for back-to-back testing based on deep learning and cluster analysis to address the limitations of traditional fixed-threshold methods. The framework integrates automated feature extraction, anomaly detection, and fault pattern classification to provide comprehensive diagnostic information while maintaining high efficiency. Building on foundational work in deep learning-based time series analysis by Längkvist et al.,[98] our approach achieves both improved detection accuracy and interpretability.

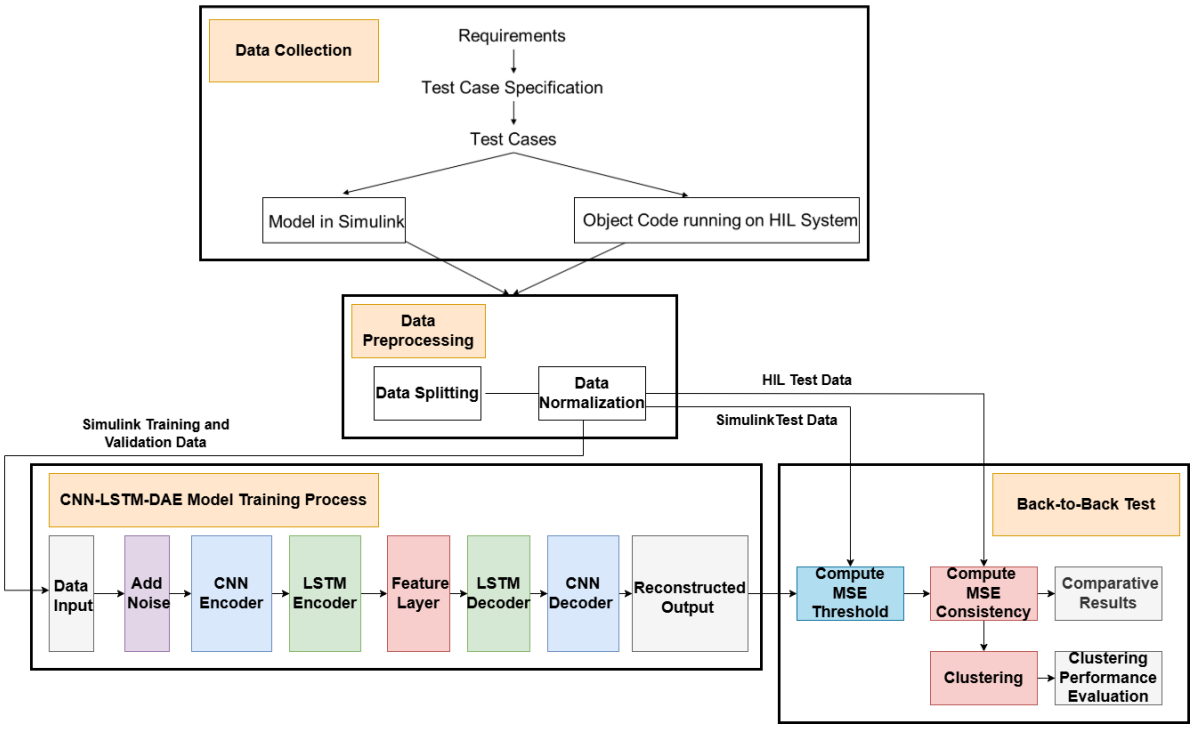


Figure 3.1: Framework Overview

Figure 3.1 illustrates the complete process flow of the CNN-LSTM-DAE deep autoencoder research framework. The core CNN-LSTM-DAE model training process includes data input, noise addition, CNN encoder, LSTM encoder, feature layer, LSTM decoder, and CNN decoder, ultimately producing reconstructed output. After model training is

completed, the system conducts back-to-back testing, calculating MSE threshold and MSE consistency, generating comparative results, and evaluating system performance through cluster analysis. This process fully demonstrates the systematic and comprehensive nature of the research methodology, forming a closed-loop intelligent analysis system from data collection to anomaly detection and finally to cluster analysis.

The core components of the framework include data collection strategy, data preprocessing, CNN-LSTM-DAE model, anomaly detection system, and cluster analysis module. The data collection strategy gathers data from different environments (MIL and HIL) and different driving scenarios to build comprehensive training and testing datasets. Data preprocessing and feature extraction use deep learning models (especially CNN-LSTM deep autoencoders) to automatically extract key features from raw data. The anomaly detection system, based on a reconstruction error threshold mechanism, automatically identifies data patterns that deviate from normal behavior. The cluster analysis module performs clustering analysis on detected anomalous data, identifying and classifying different fault patterns, helping engineers understand and locate faults. The performance evaluation system comprehensively evaluates the framework's performance in terms of detection accuracy, feature extraction quality, clustering effect, and real-time performance.

The entire framework design follows modular principles, with components connected through standard interfaces, facilitating individual optimization and replacement. Simultaneously, the framework pays special attention to processing efficiency, ensuring rapid analysis and processing of automotive electronic system data.

3.2 Data Collection Strategy

High-quality data is the foundation for successful machine learning models. This research adopts a comprehensive data collection strategy to ensure that the data for model training and testing is comprehensive and representative, following the methodology established by Kang et al.[99] in automotive data collection. Their work on comprehensive test scenario design for autonomous vehicles provides valuable insights into coverage standards and scenario selection.

Data collection covers two main environments: Simulink environment (MIL) and Scalexio[100] environment (HIL). Data collected in the Simulink environment using ASM automotive models without faults is mainly used for model training and validation. This data represents normal system behavior, providing a baseline for autoencoder learning.

Test data collected in the Scalexio environment on real-time hardware-in-the-loop systems includes both normal operation data and various fault scenario data. This data is used to validate the model's detection and classification capabilities.

To ensure the model's generalization ability, data collection covers multiple driving scenarios, including highway scenarios, urban road scenarios, non-urban road scenarios, special test scenarios, and manual driving scenarios. This multi-scenario data collection strategy ensures the model's adaptability and robustness under various driving conditions.

To test the model's fault detection capability, the research generates multiple types of fault data, including gain faults, stuck faults, noise faults, and concurrent faults. Fault injection design considers different fault durations, severities, and locations, ensuring the comprehensiveness of testing.

3.3 Data Preprocessing

Data preprocessing is a key step to ensure model training effectiveness, including data cleaning, normalization, and sequence processing. Data cleaning includes missing value handling, anomaly identification, and redundant column removal. Missing value handling detects and processes missing values in the data through methods such as linear interpolation or forward filling. Anomaly identification uses statistical methods to identify obvious anomalies, preventing them from affecting model training. Redundant column removal reduces feature space dimensionality by removing duplicate or highly correlated features.[101]

In this study, a comprehensive data preprocessing pipeline was developed to ensure consistency and standardization in data handling. The pipeline comprises three essential steps: data loading and partitioning, data merging, and data standardization, concluding with exporting the processed datasets. Specifically, all Excel files from a designated directory were loaded collectively, and each file was segmented individually into training, validation, and test sets based on an 8:1:1 ratio. These subsets were then separately stored in tuples. Subsequently, training, validation, and test subsets from all files were merged respectively, creating unified and continuous datasets.

To mitigate numerical instability and delayed convergence caused by varying feature scales during model training, the StandardScaler method from the sklearn.preprocessing module was applied. Initially, the scaler computed the mean and standard deviation exclusively from the training dataset. Using these parameters, identical transforma-

tions were sequentially applied to the validation and test datasets. Additionally, any timestamp features were deliberately excluded from scaling to avoid inappropriate numerical transformations.

Following standardization, the processed data were converted back into pandas DataFrames retaining the original column names, enhancing clarity and interpretability for subsequent analyses. Ultimately, the standardized training, validation, and test datasets were saved as separate CSV files. Additionally, the fitted StandardScaler object was serialized and stored as a pickle (pkl) file, enabling consistent reuse, especially during model deployment, thereby maintaining uniformity between training and prediction data distributions.

Through the established preprocessing framework, efficient integration and normalization of multi-file, multi-feature datasets were achieved. Moreover, conducting all experimental and deployment phases within the same standardized coordinate framework significantly supports stable model training and reproducibility of results.

3.4 CNN-LSTM Deep Autoencoder Model Design

This research proposes a deep autoencoder architecture combining CNN and LSTM, fully leveraging the advantages of both networks for feature extraction and anomaly detection. The model adopts an encoder-decoder structure, with the encoder consisting of a CNN encoder and LSTM encoder, and the decoder adopting a symmetric structure. As shown in Figure 3.2, the architecture integrates 1D convolutional layers for local pattern extraction with LSTM layers for capturing temporal dependencies, forming a powerful deep autoencoder framework for time series analysis.

CNN-LSTM-DAE Architecture		
Component	**Structure**	**Purpose**
Input Layer	Raw time series data [batch, seq_len, features]	Receive time series signals
CNN Encoder	Conv1D layers with multiple filters Activation, MaxPool, BatchNorm, Dropout	Extract local temporal patterns from input signals
LSTM Encoder	Hidden units with time sequence processing	Capture long-term dependencies in CNN-extracted features
Feature Space	Latent space representation	Compact representation of time-series data
LSTM Decoder	Hidden units (symmetric to encoder)	Decode latent representation
Output Layer	Dense layer with activation Output mapped to original dimension	Reconstruct original signal

Figure 3.2: CNN-LSTM-DAE Architecture Overview

3.4.1 Architecture Components and Mathematical Formulation

The CNN-LSTM-DAE architecture consists of several key components working together to effectively process time series data. The input layer receives raw time series signals in the format [batch, sequence_length, features].

The CNN encoder includes one-dimensional convolutional layers, max pooling layers, batch normalization layers, activation functions, and dropout layers. Convolutional layers capture local patterns in temporal data through sliding filters, max pooling layers reduce feature dimensions, batch normalization layers improve training stability, and dropout layers prevent overfitting.

For an input sequence $X \in \mathbb{R}^{B\times S\times F}$, where B is the batch size, S is the sequence length, and F is the feature dimension, the CNN encoding process can be mathematically described as:

$$X_{reshaped} = \text{reshape}(X) \in \mathbb{R}^{(B\cdot S)\times 1\times F}$$

$$C_1 = \text{ELU}(\text{BN}(\text{MaxPool}(\text{Conv1D}(X_{reshaped}; \theta_{c1}))))$$

$$C_2 = \text{ELU}(\text{BN}(\text{MaxPool}(\text{Conv1D}(C_1; \theta_{c2}))))$$

where θ_{c1} and θ_{c2} represent the parameters of the first and second convolutional layers, with 99 and 187 filters respectively, both using a kernel size of 9.

The output feature sequence from the CNN encoder is then fed into the LSTM encoder, which receives the reshaped CNN output:

$$C_{reshaped} = \text{reshape}(C_2) \in \mathbb{R}^{B\times S\times F'}$$

The LSTM encoder captures temporal dependencies through LSTM layers. The key advantage of LSTM is its ability to learn long-term dependencies, maintaining state information even in sequences with complex temporal dynamics. The LSTM encoding process is represented as:

$$h_t, c_t = \text{LSTM}(C_{reshaped,t}, h_{t-1}, c_{t-1}; \theta_{lstm\_enc})$$

$$Z = h_S$$

where $Z \in \mathbb{R}^{B\times S\times 138}$ is the latent representation, and θ_{lstm_enc} are the LSTM encoder parameters.

The decoder adopts a symmetric structure, including an LSTM decoder and output layer. The LSTM decoder is structurally symmetric to the encoder, mapping encoded features back to sequence form:

$$h'_t, c'_t = \text{LSTM}(Z_t, h'_{t-1}, c'_{t-1}; \theta_{lstm\_dec})$$

$$D = [h'_1, h'_2, ..., h'_S]$$

where $D \in \mathbb{R}^{B\times S\times 138}$ is the decoder output and θ_{lstm_dec} are the LSTM decoder parameters.

The output layer includes fully connected layers, mapping features back to the original input dimension, and using appropriate activation functions to ensure suitable output range:

$$\hat{X} = \text{Linear}(\text{ELU}(\text{Linear}(D; \theta_{d1})); \theta_{d2})$$

where $\hat{X} \in \mathbb{R}^{B\times S\times F}$ is the reconstructed output.

The compact feature space representation created in the model's middle layer (with dimension 138) serves as a powerful compressed encoding of the original time series data, enabling effective anomaly detection. The symmetric decoder structure ensures that this latent representation preserves essential information needed to reconstruct the original signal.

3.4.2 Model Training and Optimization

Model training uses mean squared error (MSE) as the loss function:

$$\mathcal{L}(X, \hat{X}) = \frac{1}{B \cdot S \cdot F} \sum_{i=1}^{B} \sum_{j=1}^{S} \sum_{k=1}^{F} (X_{i,j,k} - \hat{X}_{i,j,k})^2$$

where $\mathcal{L}(X, \hat{X})$ represents the mean squared error loss, measuring the difference between the original input X and the reconstructed output $\hat{X}$. By minimizing this error, the model learns the ability to reconstruct input data, thereby extracting effective feature representations in the encoder part.

Model training strategies include batch size, early stopping mechanism, and noise in-

jection. Batch size is determined through experimentation to find the optimal value, balancing training stability and efficiency. The early stopping mechanism monitors validation set performance to prevent overfitting. Noise injection enhances model robustness and denoising capability by adding different levels of Gaussian noise to the input.

To improve model performance, various optimization techniques are adopted, including hyperparameter optimization, regularization methods, learning rate scheduling, and model ensembling. Hyperparameter optimization uses the Optuna framework for automated optimization to find the best hyperparameter combinations. Regularization methods apply techniques such as Dropout and L2 regularization to prevent overfitting. Learning rate scheduling adopts learning rate decay or cyclic learning rate strategies to improve convergence performance.

Table 6 details the optimal hyperparameters identified for the CNN-LSTM-DAE model. These parameters were determined through extensive experimentation and optimization using the Optuna framework. The careful selection of these hyperparameters significantly contributes to the model's robust performance across different scenarios and fault conditions.

3.5 Reconstruction Error-Based Anomaly Detection

The core idea of our approach is to implement a two-stage processing flow: first using the autoencoder's reconstruction error to identify potential anomalies, and only then applying clustering analysis to characterize and classify the detected anomalies. This approach significantly improves processing efficiency since healthy data can be identified at the first stage, avoiding unnecessary clustering computations.

3.5.1 MSE Threshold Selection Method

Selecting an appropriate threshold for the reconstruction error is crucial for the system's anomaly detection performance. Rather than using a fixed, arbitrary threshold, we adopt a data-driven approach based on the statistical distribution of reconstruction errors from healthy training data.

The threshold selection process follows these steps:

1. Collect reconstruction errors from the autoencoder when processing known healthy data

2. Analyze the statistical distribution of these errors
3. Calculate multiple percentile thresholds (90th, 95th, 96th, 97th, 98th, and 99th)
4. Evaluate each threshold's performance using precision, recall, and F1 score metrics
5. Select the threshold that optimizes the balance between detection sensitivity and false alarm rate

Through extensive experimentation on validation data, we determined that the 99th percentile (corresponding to a threshold value of 0.005804) provides the optimal balance between correctly identifying anomalies and minimizing false alarms. This percentile-based approach has strong theoretical foundations, as discussed in Chapter 2, where extreme value theory suggests that anomalies naturally occur in the extreme tails of error distributions.

The use of the 99th percentile is particularly appropriate for automotive systems where safety is critical, as it ensures that the vast majority of healthy data variations are correctly classified while still maintaining high sensitivity to actual faults.

3.5.2 Anomaly Detection Workflow

The anomaly detection process works as follows:

1. For each input sequence X, the autoencoder generates a reconstruction $\hat{X}$
2. The mean squared error between X and $\hat{X}$ is calculated: $MSE(X, \hat{X}) = \frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{X}_i)^2$
3. This error is compared against the predetermined threshold (0.005804)
4. If the error exceeds the threshold, the sequence is flagged as potentially anomalous and proceeds to clustering analysis
5. If the error is below the threshold, the sequence is classified as healthy with no further processing required

This workflow ensures computational efficiency by only applying the more intensive clustering analysis to sequences that are likely to contain faults, while quickly processing the majority of healthy data. This approach is particularly valuable in real-time automotive applications where processing resources may be limited and response time is critical.

3.5.3 Error Analysis Methods

To provide deeper insights into the nature of detected anomalies, the system calculates and analyzes errors at multiple levels:

- **Sample-level error**: The overall reconstruction error for each time sequence, providing a general measure of anomaly severity
- **Feature-level error**: Reconstruction errors for individual features or signals, helping identify which specific components or sensors might be problematic
- **Time-step level error**: Error distribution across the time dimension, revealing when anomalies occur within a sequence

These multi-level error analyses provide valuable diagnostic information to engineers, helping them understand not just that an anomaly exists, but which aspects of the system are affected and how the anomaly evolves over time.

3.6 Cluster Analysis Methods

After detecting anomalies through the reconstruction error threshold, our framework applies clustering analysis to the detected anomalous data to identify and classify different types of fault patterns. This two-stage approach ensures that clustering is only performed on data that is likely to contain meaningful fault patterns, significantly reducing computational overhead and improving the clarity of clustering results.

3.6.1 Feature Extraction for Clustering

High-quality features are crucial for effective clustering. Our approach extracts features from multiple sources to provide a comprehensive representation of fault characteristics:

- **Latent representations**: Using the output from the encoder part of the CNN-LSTM-DAE autoencoder (with dimension 138), which captures the essential characteristics of the input sequence
- **Reconstruction error patterns**: Derived from the pattern of errors in the reconstruction, which can provide additional insight into fault mechanisms
- **Statistical features**: Such as mean, standard deviation, skewness, and kurtosis of both raw data and reconstruction errors

These features are extracted from data sequences that have already been identified as anomalous based on the reconstruction error threshold. By focusing only on anomalous data, the clustering process can more effectively distinguish between different fault types without being influenced by the predominance of healthy data samples.

3.6.2 Clustering Algorithms - K-means Mathematical Formulation

The K-means algorithm partitions a set of data points into K distinct clusters by minimizing the within-cluster sum of squares (WCSS). For a given dataset with n data points $\{x_1, x_2, ..., x_n\}$, K-means aims to partition these observations into K sets $S = \{S_1, S_2, ..., S_K\}$ by solving the following optimization problem:

$$\arg\min_{S} \sum_{i=1}^{K} \sum_{x \in S_i} \|x - \mu_i\|^2$$

where μ_i is the mean of points in cluster S_i, calculated as:

$$\mu_i = \frac{1}{|S_i|} \sum_{x \in S_i} x$$

The standard K-means algorithm proceeds as follows:

Algorithm 1 K-means Clustering Algorithm

1: **Input:** Dataset $X = \{x_1, x_2, ..., x_n\}$, number of clusters K
2: **Output:** Cluster assignments and centroids
3: Initialize cluster centroids $\mu_1, \mu_2, ..., \mu_K$ randomly from the data points
4: **repeat**
5: **Assignment step:** Assign each data point to the nearest centroid
6: $S_i = \{x_j : \|x_j - \mu_i\|^2 \leq \|x_j - \mu_l\|^2 \text{ for all } l \neq i\}$
7: **Update step:** Recalculate centroids as the mean of all points assigned to that cluster
8: $\mu_i = \frac{1}{|S_i|} \sum_{x \in S_i} x$
9: **until** cluster assignments do not change or maximum iterations reached
10: **return** Cluster assignments $S = \{S_1, S_2, ..., S_K\}$ and centroids $\mu = \{\mu_1, \mu_2, ..., \mu_K\}$

The convergence of K-means is guaranteed, as each iteration decreases the WCSS objective function, which is bounded below by zero. While K-means may converge to a local optimum, multiple runs with different initializations can help find a better solution. The time complexity of the algorithm is O(nKdi), where n is the number of data points,

K is the number of clusters, d is the dimension of the data, and i is the number of iterations until convergence.

3.6.3 Clustering Algorithms Comparison

This research implements and evaluates multiple clustering algorithms to determine the most effective approach for fault classification:

1. **K-means clustering**: A centroid-based algorithm that partitions data into K clusters based on minimizing within-cluster distances. K-means is computationally efficient and produces clusters of roughly similar sizes and shapes.

2. **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**: A density-based algorithm that identifies clusters as dense regions separated by low-density regions. DBSCAN can discover arbitrarily shaped clusters and identify noise points.

3. **Gaussian Mixture Models (GMM)**: A probability-based clustering method that models data as a mixture of multiple Gaussian distributions. GMM provides soft assignments of data points to clusters based on probabilities.

4. **Fuzzy C-means (FCM)**: An extension of K-means that allows data points to belong to multiple clusters with varying degrees of membership. FCM is suitable for data with fuzzy boundaries between clusters.

Each algorithm is systematically evaluated using internal clustering validation metrics such as Silhouette Coefficient, Davies-Bouldin Index, and Calinski-Harabasz Index, as well as computational efficiency and memory usage measurements.

3.6.4 Fault-based K Value Selection Process

For K-means and FCM algorithms where the number of clusters (K) must be specified in advance, we implement a fault-based K value selection process rather than an automatic optimization approach. This method is designed to enhance computational efficiency and improve fault classification clarity:

1. First, assess if the data segment contains potential faults by comparing reconstruction errors against the predetermined threshold (0.005804)

2. If the data is identified as healthy (with reconstruction errors below threshold), no clustering is performed, significantly reducing computational overhead

3. For fault data segments, the K value is determined based on the number of known fault types present in the specific time interval, with consideration for potential healthy data segments

4. Typically, K is set to the number of fault types plus one (K = fault_count + 1) to account for possible healthy data points

This approach ensures that clustering is only performed when necessary and with appropriate complexity, avoiding over-clustering of healthy data while maintaining the ability to discriminate between different fault types. For our analysis, we define specific time intervals and corresponding K values based on the known fault injections in those intervals, which provides more meaningful and interpretable clustering results compared to automatic K selection methods.

3.6.5 Clustering Evaluation and Visualization

To evaluate and interpret clustering results, we employ multiple approaches:

- **Internal validation metrics**: Including Silhouette Coefficient (measuring how similar an object is to its own cluster compared to other clusters), Davies-Bouldin Index (measuring the average similarity between each cluster and its most similar cluster), and Calinski-Harabasz Index (measuring the ratio of between-cluster dispersion to within-cluster dispersion)

- **Stability evaluation**: Assessing the consistency of clustering results under different initializations or parameter settings

- **Visualization**: Using dimensionality reduction techniques (such as PCA or t-SNE) to reduce high-dimensional feature spaces to 2D or 3D for visualization, providing an intuitive evaluation of clustering effectiveness

These evaluation methods help select the most appropriate clustering algorithm and parameters for each specific application scenario, ensuring that the fault classification results are reliable and meaningful.

Through comprehensive analysis of different clustering methods, we have determined that K-means clustering provides the best overall performance when considering both

clustering quality and computational efficiency. The results of this comparative analysis are presented in detail in Chapter 4.

4 Experimental Implementation and Case Studies

4.1 Case Study: Gasoline Engine and Vehicle Dynamics

To validate the proposed method, this research constructed a test environment combining software and hardware. The software simulation layer used the MATLAB/Simulink environment configured with ASM automotive simulation models, covering key subsystems such as engine, powertrain, and vehicle dynamics. The model architecture is illustrated in Figure 4.1. For hardware implementation, a hardware-in-the-loop (HIL) architecture based on dSPACE SCALEXIO achieved real-time system interaction through Ethernet communication.

The hardware platform used in the experiment mainly included the dSPACE SCALEXIO real-time simulator and control input devices, as shown in Figure 4.2. The SCALEXIO system, as the core of the HIL system, provides high-performance real-time computing capabilities, supporting high-fidelity simulation of complex automotive models. The system adopts a multi-core processor architecture with a high number of input/output channels, enabling it to handle a large number of sensor and actuator signals. SCALEXIO connects to the PC via Ethernet, ensuring high-speed stability in data transmission. Control input devices include an LG steering wheel, brake pedal, and accelerator pedal, which are used in manual driving test scenarios to simulate real driving inputs, allowing researchers to collect real driving behavior data through manual operations.

The software environment consisted of the following tools: MATLAB/Simulink 2021 for developing and configuring simulation models, including control algorithm design and signal processing; ModelDesk 5.6 for parameterization configuration and experimental design, supporting the definition and execution of various driving scenarios, and serving as the main interface for opening models and connecting to hardware platforms; ConfigurationDesk for generating .sdf files required for SCALEXIO execution, containing all compiled project content for real-time execution; ControlDesk as the experiment execu-

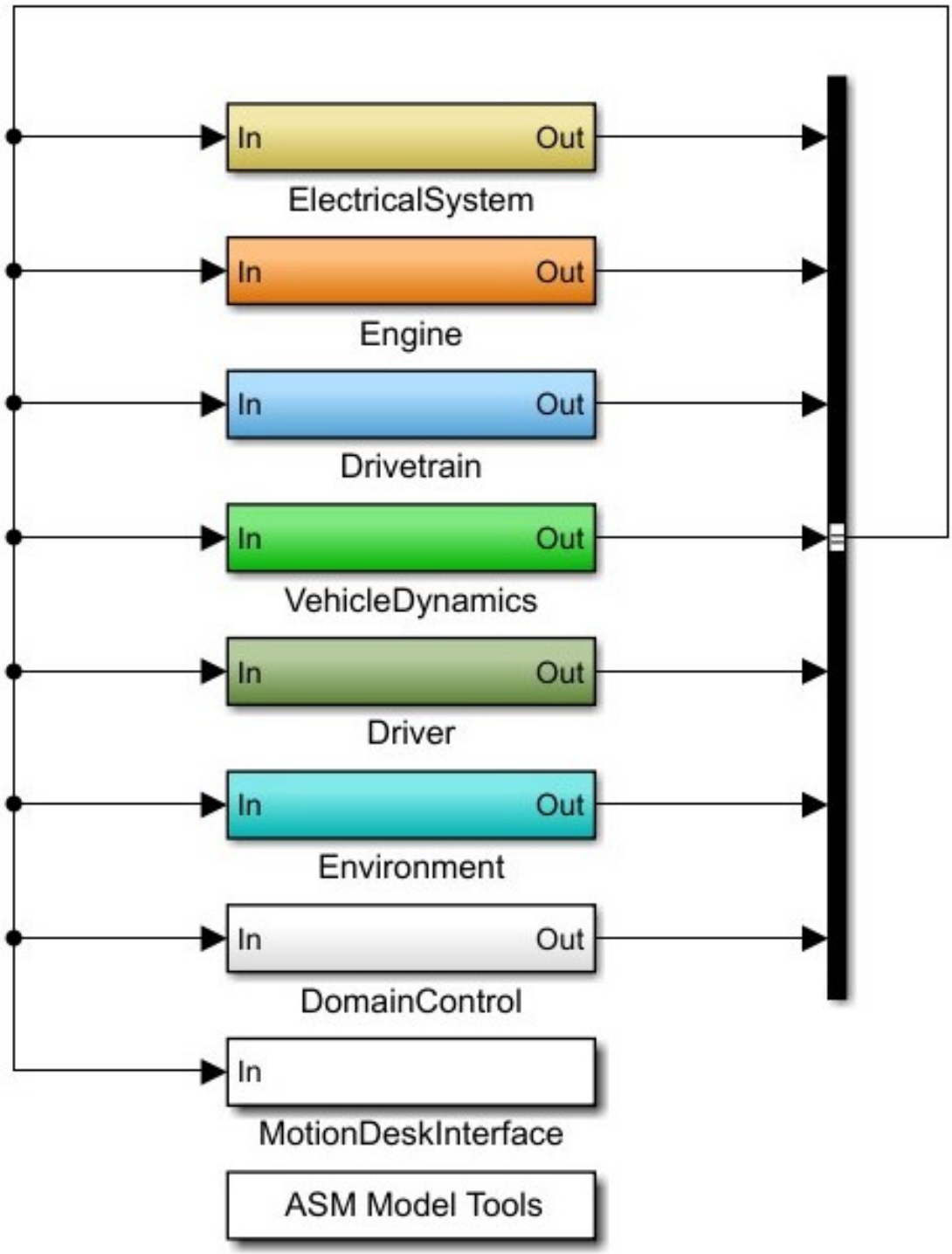


Figure 4.1: Internal structure of the ASM Gasoline Engine Model

tion and data acquisition platform, supporting real-time monitoring and control, used to

Figure 4.2: Real-time virtual test drives environment

record various sensor and actuator data from the simulation environment; MotionDesk as a 3D visualization tool for real-time observation of simulated driving environments and vehicle behavior.

This research adopted three different data collection methods, corresponding to different simulation environments: automatic driving training data collection (Simulink environment), manual driving training data collection (VEOS environment), and Scalexio hardware platform test data collection. Automatic driving training data collection opened the model's .slx file through ModelDesk, automatically started the model in Simulink and ran it, with data recording parameters preset in the Simulink model, using MotionDesk to observe the simulation in real-time, and data exported directly from the Simulink environment. Manual driving training data collection generated .sdf files for the VEOS environment through ASM Model Tools, opened the generated files using ModelDesk, collected data through ControlDesk, operated control input devices (steering wheel, pedals) for manual driving, and observed the simulated driving situation using

MotionDesk. Scalexio hardware platform test data collection generated Scalexio-specific .sdf files through ConfigurationDesk, opened the files using ModelDesk and connected to the Scalexio platform, collected data through ControlDesk, flexibly switched between manual driving (using control devices) and automatic driving modes, and observed the driving environment in real-time using MotionDesk.

This multi-level data collection method ensured that the research could obtain training and test data from different environments, providing a comprehensive learning and validation foundation for the model.

4.2 Data Collection

To comprehensively validate the performance and robustness of the proposed methodology, detailed case studies were conducted focusing on a gasoline engine model. Initially, the ASM Gasoline Engine model parameters required for data collection were configured within Simulink, with collected data items outlined in Table 4.1. Each simulation run initiated automatic recording to facilitate subsequent data exporting. The Simulink model (.slx file) was then loaded into ModelDesk, and different driving scenarios were manipulated through MotionDesk. Specific autonomous driving scenarios utilized for data collection are enumerated in Table 4.2.

Table 4.1: Collected Signals and Their Units

Signal Name	**Description**	**Unit**
Pos_AccPedal	Accelerator pedal position	%
v_Vehicle	Vehicle speed	km/h
n_Engine	Engine speed	rpm
T_Out_Engine	Engine temperature	°C
p_Out_EGR	EGR outlet pressure	Pa
Pos_Throttle	Throttle position	%
Trq_MeanEff_Engine	Mean effective engine torque	Nm
p_Rail	Rail pressure	bar
Ctrl_Throttle	Throttle control signal	[0-1]
CrankAngle_Cyl(1)	Cylinder 1 crankshaft angle	aTDC
CrankAngle_Cyl(2)	Cylinder 2 crankshaft angle	aTDC
CrankAngle_Cyl(3)	Cylinder 3 crankshaft angle	aTDC
CrankAngle_Cyl(4)	Cylinder 4 crankshaft angle	aTDC
CrankAngle_Cyl(5)	Cylinder 5 crankshaft angle	aTDC
CrankAngle_Cyl(6)	Cylinder 6 crankshaft angle	aTDC

Table 4.2: Modeling Data (Simulink) Driving Scenario Statistics

Scenario Type	**File Size**	**Scenario Description**	**Sample Size**
ABSBrake	5.14 KB	Emergency braking scenario	2.83×10^4
CLZ Urban	15.10 KB	Urban road driving	8.30×10^4
FollowRoad	12.07 KB	Follow driving	6.88×10^4
Highway	35.84 KB	Highway driving	2.00×10^5
LaneChange	6.71 KB	Lane change scenario	3.68×10^4
Slalom	5.64 KB	S-shaped maneuvering driving	3.02×10^4
Manual_CLZ1	17.31 KB	Manual urban driving 1	1.16×10^5
Manual_CLZ2	18.88 KB	Manual urban driving 2	1.12×10^5
Manual_Highway1	28.36 KB	Manual highway driving 1	1.65×10^5
Manual_Highway2	26.20 KB	Manual highway driving 2	1.54×10^5

Subsequently, the ASM Tools were employed to generate an SDF file configured for the VEOS environment. Careful attention was given to ensuring that only one type of environment-specific SDF file (either Scalexio or VEOS) was present within a single project to prevent environmental conflicts in ModelDesk, which would otherwise impede manual driving functionality. After correctly loading the VEOS-compatible SDF file into ModelDesk, external equipment facilitated manual driving data collection. Two of the most complex driving scenarios—urban and highway—were selected, each repeated twice. The complete raw dataset collected from the Gasoline Engine model is summarized in Table 4.2. For the VehicleDynamic model, while the driving scenarios remained identical to those employed with the Gasoline Engine model, the specific collected data differed as listed in Table 4.3. Figure 4.3 illustrates comparative vehicle speed profiles across various scenarios.

In addition to Simulink-collected data, further datasets intended for testing purposes

were acquired using the Scalexio platform. The corresponding SDF file utilized here was generated by ConfigurationDesk software. Initially, healthy-state autonomous driving and manual driving data were collected under identical scenarios as previously established in Simulink; these data are documented in Table 4.4. Subsequently, fault-injected autonomous driving data intended solely for evaluating model performance were collected, summarized in Table 4.5. It is essential to highlight that all system sampling intervals were uniformly set to 0.001 seconds. Fault injection was algorithmically implemented upstream of specific model components, ensuring the structural and functional integrity of the models remained unaffected. ControlDesk software enabled precise control over fault injection duration and amplitude, thus ensuring reproducibility and controlled parameter variation during testing.

Table 4.3: Vehicle Dynamics Model Data Collection

Signal	**Unit**
time	s
v_x_Vehicle_CoG	km/h
n_Engine	rpm
YawRate_Vehicle_CoG	deg/s
a_x_Vehicle_CoG	m/s^2
a_y_Vehicle_CoG	m/s^2
TrqSpring_Steering-Column	Nm
Trq_PowerSteering	Nm
Angle_SteeringWheel	deg
omega_Wheel[FL]	rad/s
omega_Wheel[FR]	rad/s
omega_Wheel[RL]	rad/s
omega_Wheel[RR]	rad/s
Pos_BrakePedal	%

4.3 Data Preprocessing

Following data collection, preprocessing was uniformly applied to all Gasoline Engine data obtained from Simulink. Specifically, the scaler object derived from the standardized training dataset was employed consistently across all Scalexio-derived datasets. Time-related features were temporarily removed prior to standardization and reinstated afterward to simplify downstream analysis. This approach resulted in a comprehens-

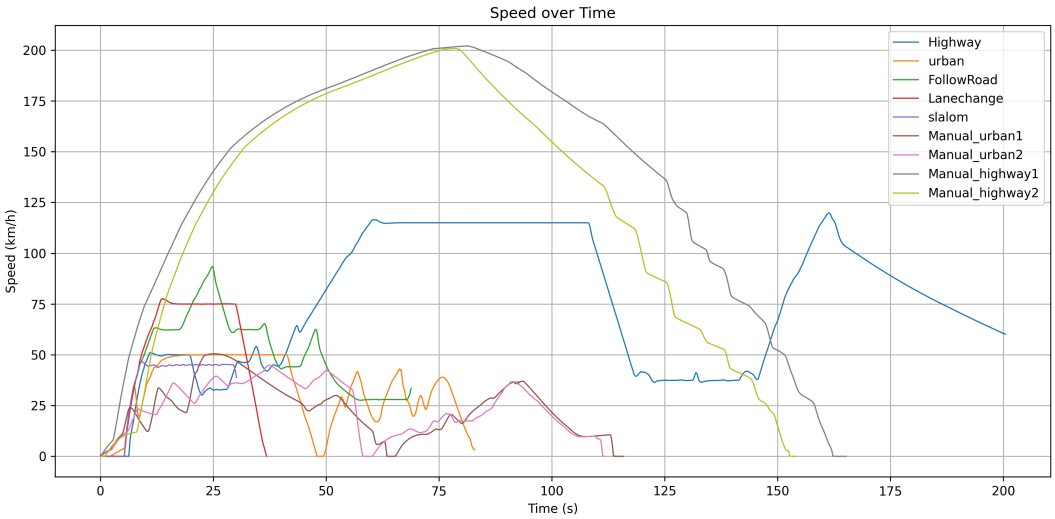


Figure 4.3: Vehicle Speed Comparison Across Different Driving Scenarios

Table 4.4: Healthy Operating Data (Scalexio) Statistics

Scenario Type	**File Size**	**Description**	**Sample Size**
ABSBrake	8.31 KB	Emergency braking scenario	2.98×10^4
Slalom	10.23 KB	S-shaped maneuvering driving	3.50×10^4
LineChange	11.06 KB	Lane change scenario	3.80×10^4
Follow	21.78 KB	Follow driving	7.36×10^4
CLZ	25.43 KB	Urban driving scenario	8.39×10^4
Highway	48.09 KB	Highway driving	1.65×10^5
ManualCLZ	116.45 KB	Manual urban driving	2.28×10^5

Table 4.5: Fault Data (Scalexio) Statistics

Fault Type	**File Size**	**Fault Location**	**Sample Size**
ACC Noise fault	44.12 KB	ACC sensor (25-90s)	1.53×10^5
RPM Gain fault	44.51 KB	RPM sensor (25-94s)	1.51×10^5
ACC5Gain-RPMStuck	44.36 KB	ACC sensor (50-60s), RPM sensor (25-83s)	1.50×10^5
ACC5Gain-RPM2Gain-RPMNoise	44.38 KB	ACC sensor (25-70s), RPM sensors (40-80s, 55-90s)	1.50×10^5
RPM Stuck fault	45.37 KB	RPM sensor (25-94s)	1.54×10^5

ive, standardized dataset across mixed scenarios. Furthermore, to investigate potential scenario-specific performance deviations, additional datasets were individually standard-

ized based on urban-only, highway-only, and non-urban conditions.

Similarly, data collected from the VehicleDynamic model were independently standardized due to intrinsic differences in data attributes compared to the Gasoline Engine datasets. Consequently, separate scaler objects were computed exclusively from training data pertaining to the VehicleDynamic model, resulting in four distinctive scenario datasets (urban, highway, non-urban road, and VehicleDynamic).

In subsequent modeling efforts, hyperparameters optimized using the mixed-scenario Gasoline Engine dataset were directly tested across the aforementioned four specialized datasets.

4.4 Model Selection and Implementation

This research evaluated multiple deep learning and traditional machine learning models to determine the architecture most suitable for back-to-back testing. As shown in Table 4.6, we evaluated seven different model architectures across four key performance metrics. The CNN-LSTM DAE model demonstrates superior performance in most categories, particularly in fault detection (F1 score) and cross-scenario consistency, while maintaining acceptable real-time processing capabilities.

Table 4.6: Model Performance Comparison

Model Type	**Healthy Data Classification Accuracy**	**Fault Detection F1 Score**	**Cross-Scenario Consistency**	**Processing Capability**
CNN-LSTM DAE	90.54%	0.926	High	≈9ms/batch
BI-LSTM DAE	87.21%	0.883	Medium	≈12ms/batch
1DCNN	85.37%	0.875	Medium	≈7ms/batch
GRU DAE	86.92%	0.879	Medium	≈10ms/batch
PCA+SVM	82.14%	0.841	Low	≈5ms/batch
SVM-KNN-DT	80.76%	0.820	Low	≈6ms/batch
WGAN-GP	94.81%	0.762	Low	≈15ms/batch

Based on comprehensive evaluation, the CNN-LSTM deep autoencoder was selected as the best model, primarily based on the following four points:

1. **Best healthy data classification performance**: Compared to other models, the CNN-LSTM model provided the most stable healthy state recognition rate (90.54%), maintaining high detection sensitivity while ensuring a low false alarm rate.

2. **Strongest fault detection capability**: Under all evaluated threshold settings, the CNN-LSTM model consistently showed the highest F1 score (0.926), indicating its excellent balance of precision and recall in fault detection.

3. **Strongest cross-scenario consistency**: The CNN-LSTM model exhibited minimal performance fluctuation across different driving scenarios (highway, urban, follow, etc.), ensuring system reliability under various conditions.

4. **High efficiency processing requirements**: Although some traditional models (such as PCA+SVM) process faster, CNN-LSTM still maintains high accuracy, with an average batch processing time of about 9ms.

It is worth noting that although the WGAN-GP model showed the highest accuracy in healthy data recognition (94.81%), its fault detection F1 score was significantly lower (0.762), indicating a tendency to overly favor the healthy category, compromising fault detection capability. Therefore, considering all metrics comprehensively, the CNN-LSTM deep autoencoder became the best choice.

4.4.1 CNN-LSTM-DAE Model Hyperparameter Optimization Strategy

In this study, to systematically explore the optimal configuration of deep learning models, an automated hyperparameter optimization framework based on Optuna was developed. This framework systematically adjusts various hyperparameters related to network architecture and training strategies. Initially, both training and validation datasets were loaded into a custom-built TimeSeriesDataset. Multivariate time-series data were transformed into fixed-length sequences via a sliding window approach. Each sample comprised sequences of 50 time steps with a stride of 25, enabling the model to effectively capture temporal dependencies while efficiently utilizing data resources.

Subsequently, a convolutional neural network combined with a long short-term memory autoencoder (CNN-LSTM-DAE) was constructed. This model employs CNN layers for temporal feature extraction and dimensionality reduction of input sequences.

Extracted high-level features are then passed through a bidirectional LSTM encoder and decoder, and the reconstructed output, matching the original sequence shape, is obtained through fully connected layers. Additionally, an adjustable noise injection mechanism (parameterized by noise_level) was incorporated to enhance network robustness during training.

For hyperparameter search, Optuna's Hyperband Pruner was adopted as the pruning strategy in conjunction with the Tree-structured Parzen Estimator (TPE) sampler, facilitating adaptive exploration within continuous and discrete hyperparameter spaces during each iteration. Hyperparameters explored included the learning rate (learning_rate), hidden layer dimensions (hidden_size), number of layers (num_layers), number of CNN channels (cnn_channels_1, cnn_channels_2), convolution kernel sizes (kernel_size_1, kernel_size_2), activation function types (activation), optimizer choices (optimizer), and batch sizes (batch_size). Each hyperparameter configuration (referred to as a 'Trial') was trained for a maximum of 27 epochs, subject to early stopping criteria or pruning by Hyperband Pruner if the validation loss failed to improve sufficiently, thus preventing computational resource wastage.

During training iterations, the mean squared error (MSE) was employed as the primary loss function to assess reconstruction quality. PyTorch's automatic differentiation facilitated efficient forward and backward propagation, optimizing parameters to minimize reconstruction errors. Upon completion of each Trial, the optimal validation loss and corresponding hyperparameter settings were documented. Additionally, parameter importance and correlations among trials were visualized through scatter and heatmap plots in Figure 4.4, enabling quick evaluation of hyperparameter influence on reconstruction performance.

Upon completion of all trials, the framework generated comprehensive outputs including: 1) a CSV file consolidating optimal training and validation losses along with the reasons for termination of each trial (such as 'early_stopping' or 'pruned'); 2) serialized model weights and key metrics saved in independent .pkl files, ensuring reproducibility and consistency in subsequent model deployments; 3) an optimization report in JSON format detailing the best hyperparameter configuration, trial statistics (including counts of completed, pruned, failed, and early-stopped trials), and hyperparameter importance concerning target losses.

Overall, this hyperparameter optimization pipeline significantly alleviated the complexity and labor intensity of manual parameter tuning. By systematically exploring the hyperparameter space, it maximized the potential of the CNN-LSTM autoencoder

in reconstructing temporal features, thereby laying a robust foundation for future model application and deployment.

Table 4.8 details the optimal hyperparameters identified for the CNN-LSTM-DAE model.

4.4.2 CNN-LSTM-DAE Model Architecture Design

The finally selected deep autoencoder adopted a CNN-LSTM hybrid architecture, based on the optimal hyperparameters identified through the Optuna optimization process and detailed in Table 4.8. This architecture is capable of capturing both local and long-distance dependency features in time series data, providing superior performance in automotive signal analysis.

The model architecture mainly consists of the following components:

- **Convolutional Neural Network (CNN) Encoder:**
  - First convolutional layer: Uses 99 filters with kernel size 9, through ELU activation function and batch normalization layer
  - Second convolutional layer: Uses 187 filters with kernel size 9, also adopting ELU activation and batch normalization
  - Max pooling operation (pool size 2) after each convolutional layer to reduce feature dimensions
  - Dropout set to 0.104 to prevent overfitting
- **LSTM Encoder:**
  - Receives feature sequences output from the CNN encoder
  - Contains 138 hidden units
  - Single-layer LSTM structure, capturing dependencies in the time dimension
- **LSTM Decoder:**
  - Symmetric to the encoder structure, also containing 138 hidden units
  - Learns to map encoded features back to the original space
- **Output Layer:**
  - Fully connected layer maps the LSTM decoder's output back to the original input dimension

  - Includes an intermediate layer with nonlinear activation and a final linear output layer

4.4.3 Model Training Process

The training of the CNN-LSTM deep autoencoder followed a systematic approach:

- **Data Preparation:**
  - Signal normalization and standardization
  - Training/validation/test set division (ratio of 80%/10%/10%)
  - Noise injection for robustness training
- **Network Configuration:**
  - CNN layers for temporal feature extraction
  - LSTM layers for sequence modeling
  - Symmetric decoder structure
- **Training Parameters:**
  - Batch size: 64
  - Learning rate: 0.000669
  - Optimizer: Adam
  - Training epochs: 850
  - Noise level: 10%
  - Activation function: ELU

The Python environment in Table 4.7 was configured with the following key dependencies to ensure consistency in model training and evaluation:

All dependencies were version-controlled through a requirements.txt file to ensure experimental reproducibility. PyTorch provided the necessary deep learning framework, scikit-learn was used for traditional machine learning algorithms and evaluation metrics, while Optuna framework enabled automated hyperparameter optimization.

The systematic exploration of these parameter impacts and their correlations guided our optimization strategy and contributed to the final model configuration's effectiveness.

Table 4.7: Python Environment Dependencies and Versions

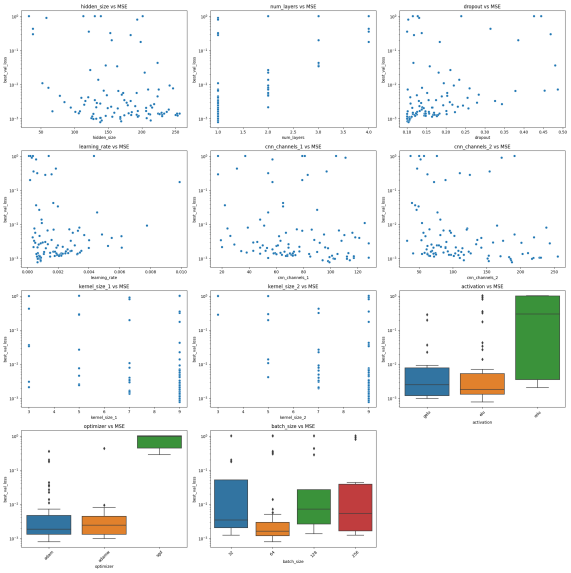
Library	**Version**
numpy	≥1.21.0, <2.0
pandas	≥1.3.0, <2.0
matplotlib	≥3.4.0, <4.0
torch	≥1.13.0, <2.0
torchvision	≥0.14.0, <1.0
scikit-learn	≥1.0.0, <2.0
optuna	≥3.0.0, <4.0
fuzzy-c-means	[1.7.2]

Table 4.8: CNN-LSTM-DAE Model Hyperparameters

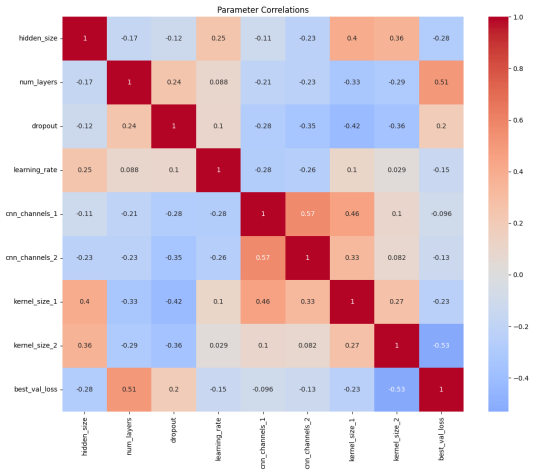
Parameter	**Value**	**Description**
learning_rate	0.000669	Learning rate optimized by Optuna
dropout	0.104238	Random dropout ratio to prevent overfitting
hidden_size	138	LSTM hidden layer size
num_layers	1	Number of LSTM layers
cnn_channels_1	99	Number of filters in first convolutional layer
cnn_channels_2	187	Number of filters in second convolutional layer
kernel_size_1	9	Kernel size of first convolutional layer
kernel_size_2	9	Kernel size of second convolutional layer
activation	'elu'	Chosen activation function
optimizer	'adam'	Optimizer type
batch_size	64	Number of samples per batch
noise_level	10%	Noise level added during training

The model optimization process implemented systematic parameter tuning through the Optuna framework. To verify noise resistance capability, Gaussian noise was injected into the training data at noise levels of 3%, 6%, 8%, and 10%.

Table 4.9 presents the model's performance metrics under different noise injection levels during training. While the model's accuracy metrics (MSE, MAE, RMSE) gradually decrease with increasing noise levels, the R^2 values remain impressively high even at 10% noise, demonstrating the model's robust denoising capabilities. This noise resistance is particularly important for automotive applications where sensor data may



(a) Individual parameter impacts on model performance



(b) Correlation matrix between hyperparameters

Figure 4.4: Analysis of hyperparameter relationships and their influence on model performance

contain various levels of noise.

Table 4.9: Model Performance Under Different Noise Levels

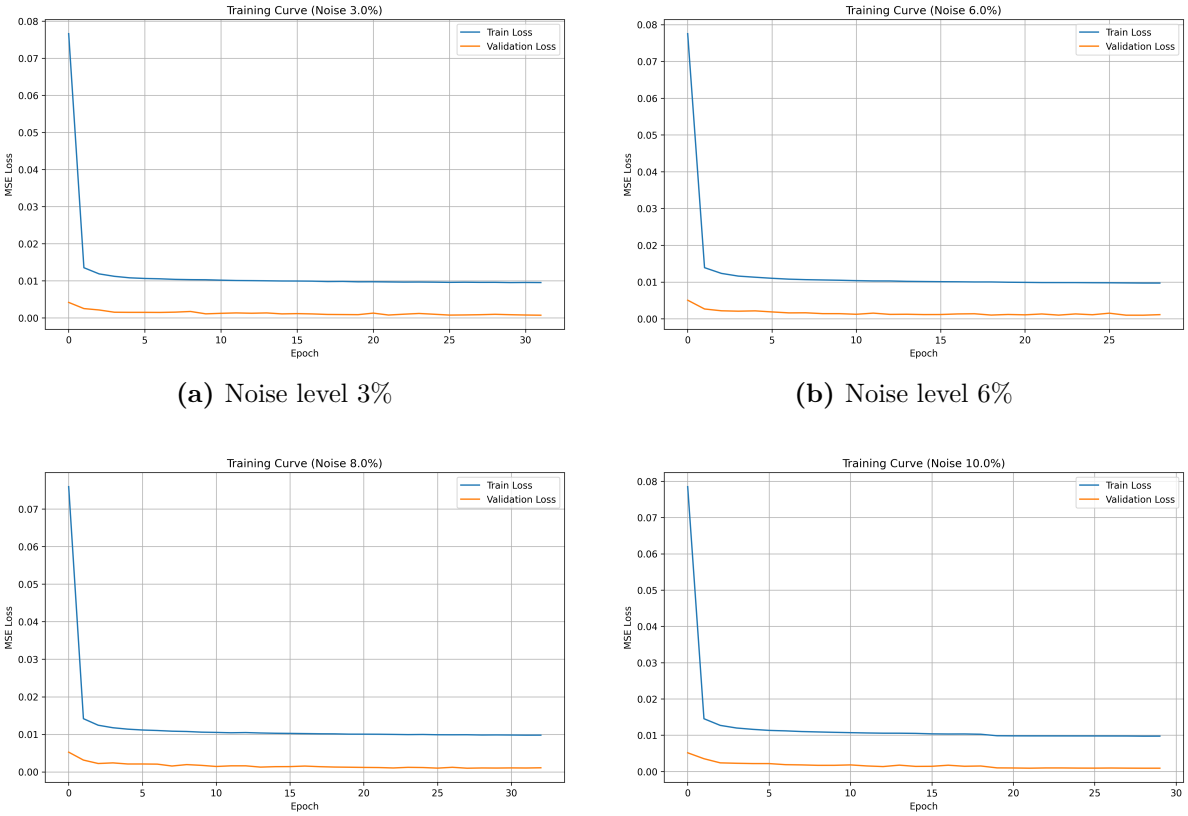
Noise Level	MSE	MAE	RMSE	R^2
3%	0.00124	0.00891	0.03521	0.9891
6%	0.00189	0.01234	0.04352	0.9845
8%	0.00256	0.01567	0.05063	0.9812
10%	0.00312	0.01892	0.05587	0.9768

The optimization process focused on several key aspects: architecture optimization (layer configuration tuning, neuron number optimization, activation function selection), training strategy (learning rate scheduling, batch size selection, early stopping criteria), and performance monitoring (reconstruction error tracking, computational efficiency, memory usage).

The final model configuration achieved good performance at different noise levels while maintaining the computational efficiency required for real-time operation. Training results demonstrated strong feature extraction capability and consistent reconstruction performance under various conditions.

Figure 4.5 displays the training and validation curves for models trained with different

noise levels. As shown in Figure 4.5, while higher noise levels initially result in higher loss values, all models eventually achieve stable convergence with minimal gaps between training and validation loss. This indicates that noise injection enhances the model's generalization ability without compromising its convergence properties.



(a) Noise level 3%

(b) Noise level 6%

(c) Noise level 8%

(d) Noise level 10%

Figure 4.5: Hyperparameter optimization results with training and validation accuracy under different noise levels

For a more intuitive comparison of how different noise levels affect the validation performance, Figure 4.6 presents a direct comparison of validation loss curves across all noise levels. This visualization clearly shows that while higher noise levels generally result in slightly higher validation loss, all models converge successfully, with the differences becoming minimal after sufficient training epochs.

The final model trained with 10% noise level (shown in Figure 4.5) was selected as our preferred configuration for back-to-back testing. This choice was made because the model maintained excellent performance even under high noise conditions, demonstrating superior robustness and noise resistance capabilities. Such characteristics are crucial for automotive applications, where sensors frequently operate in noisy environments

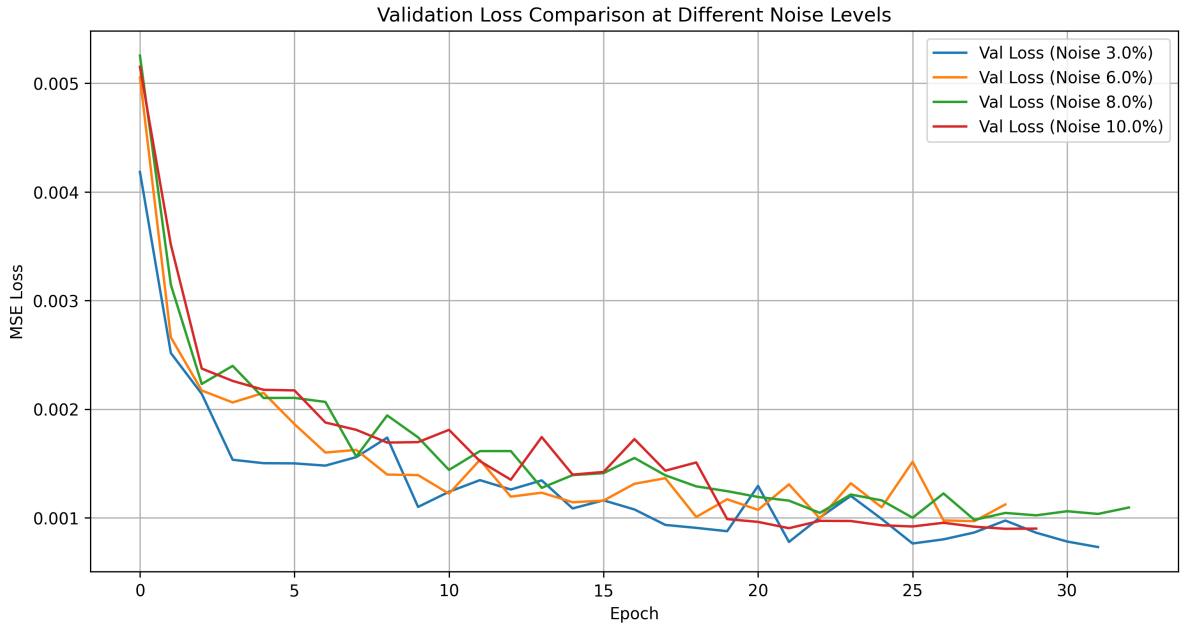


Figure 4.6: Validation loss comparison at different noise levels (3%, 6%, 8%, and 10%), showing how all models eventually converge despite initial differences in loss values

and signal quality can vary significantly. The 10% noise model offers the best balance between accuracy in ideal conditions and resilience in challenging, real-world scenarios.

4.5 Implementation Details of Threshold Selection

The proposed framework implements a systematic data-driven approach for selecting the optimal reconstruction error threshold that differentiates normal system behavior from anomalous conditions. Rather than relying on arbitrarily fixed thresholds, our method derives thresholds directly from the statistical properties of the reconstruction error distribution on known healthy data.

The threshold selection process begins by evaluating the trained CNN-LSTM-DAE model on the Simulink test dataset, which contains only healthy operating data that was not used during model training. This approach ensures that the thresholds reflect the model's performance on unseen data rather than potentially overfitting to the training set characteristics. For each sequence in the test dataset, we compute the reconstruction error using the mean squared error between the original sequence and its reconstruction from the model.

After collecting reconstruction errors for all sequences, we analyze their statistical

distribution to determine appropriate threshold candidates. Specifically, we calculate six different percentile values (90th, 95th, 96th, 97th, 98th, and 99th) from this distribution, with each percentile representing a different trade-off between detection sensitivity and false alarm rate. The computed threshold values are shown in Table 4.10.

Table 4.10: Percentile-Based Threshold Values

Percentile	**Threshold Value**
90th	0.001220
95th	0.001700
96th	0.001913
97th	0.002372
98th	0.003334
99th	0.005804

To determine which of these candidate thresholds provides the optimal balance for our application, we conducted a comprehensive evaluation of each threshold across two dimensions: (1) consistency in correctly classifying healthy data and (2) effectiveness in detecting various fault types. Table 4.11 presents the evaluation results for three representative fault scenarios across different threshold levels.

Table 4.11: Performance Comparison Across Different Threshold Levels

Fault Type	**Metric**	**Threshold Percentile**				
		90th	**95th**	**97th**	**98th**	**99th**
RPM Gain	Precision	0.8434	0.8911	0.9156	0.9350	0.9764
	Recall	0.9964	0.9964	0.9964	0.9964	0.9921
	F1 Score	0.9135	0.9408	0.9543	0.9647	0.9842
RPM Stuck	Precision	0.8476	0.8818	0.9085	0.9236	0.9643
	Recall	0.8939	0.8739	0.8582	0.8421	0.8310
	F1 Score	0.8701	0.8778	0.8826	0.8810	0.8927
ACC Gain + RPM Stuck	Precision	0.8024	0.8478	0.8702	0.8925	0.9435
	Recall	0.8491	0.8361	0.8150	0.8090	0.7926
	F1 Score	0.8251	0.8419	0.8417	0.8487	0.8615

The evaluation results revealed important performance trends. As the threshold percentile increases from 90th to 99th:

1. **Precision** consistently improves across all fault types, with the 99th percentile threshold delivering substantial improvements. For RPM Gain faults, precision increases from 0.8434 to 0.9764, representing a 15.8% improvement. Similar trends were observed for RPM Stuck and combined ACC Gain + RPM Stuck faults, with precision increases of 13.8% and 17.6%, respectively.

2. **Recall** shows a moderate decrease as the threshold becomes more conservative, particularly for more complex fault types. For RPM Gain, the decrease is minimal (0.43%), while for RPM Stuck and combined faults, recall decreases by 7.0% and 6.7%, respectively.

3. **F1 Score**, which balances precision and recall, shows the most favorable values at the 99th percentile threshold for all tested fault types. This indicates that the improvement in precision outweighs the slight decrease in recall, particularly for safety-critical applications where false alarms can diminish system trustworthiness.

Additionally, we examined the consistency ratio (percentage of healthy data correctly classified as normal) across different driving scenarios. At the 99th percentile threshold, all scenarios showed excellent consistency ratios ranging from 95.30% to 99.12%, confirming that the threshold effectively minimizes false positives during normal operation.

Based on this comprehensive evaluation, we selected the 99th percentile threshold (0.005804) as the optimal value for our framework. This selection is justified by:

1. Superior precision across all fault types, which is crucial for avoiding false alarms
2. Excellent F1 scores, indicating good overall detection performance
3. High consistency ratios in healthy data classification across diverse driving scenarios
4. Theoretical alignment with extreme value theory for anomaly detection, where anomalies naturally occur in the extreme tails of distributions

The selection of the 99th percentile threshold represents a carefully balanced trade-off that prioritizes high-confidence fault detection while maintaining good sensitivity to various fault types in automotive back-to-back testing scenarios.

4.6 Cluster Analysis Method Selection

To enhance the fault detection framework and improve feature analysis capabilities, this research designed a multi-level clustering analysis system implemented based on features extracted by CNN-LSTM-DAE. This system adopted a two-stage processing flow: first using the autoencoder's reconstruction error to identify potential faults, then performing deep feature extraction on the fault data segments, and analyzing them through different clustering algorithms.

In practical implementation, the system first converts input time series data into fixed-length sequence segments (50 time steps in this research), then calculates the reconstruction error for each segment through the pre-trained CNN-LSTM-DAE. When segments with errors exceeding the threshold are identified as potential faults, these fault segments are sent to the encoding part of the autoencoder to extract representative latent feature vectors. These features mainly come from the last hidden state of the LSTM encoder, effectively capturing temporal features of the sequence data and potential fault patterns.

This approach allows us to effectively extract feature representations with temporal context information, providing high-quality input for subsequent clustering analysis. The feature extraction process fully utilizes the representation capability of deep learning models, enabling the system to automatically identify and distinguish different types of fault patterns without manually designing feature extraction rules.

This research primarily evaluated the performance of the K-means clustering algorithm in deep feature analysis, while also comparing methods such as DBSCAN, Gaussian Mixture Models (GMM), and Fuzzy C-means (FCM). As a classic clustering algorithm, K-means assigns data points to the nearest cluster center based on Euclidean distance. We used evaluation metrics such as Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Index to systematically evaluate the performance of each method across different fault scenarios, with K values predetermined based on the number of fault types in each time interval.

To improve clustering quality, we adopted a series of preprocessing and optimization techniques, including feature preprocessing and dimensionality reduction. The original feature dimension is typically high (model hidden layer dimension is 138), and clustering directly in high-dimensional space may face the ”curse of dimensionality.” We adopted RobustScaler for feature standardization, combined with PCA dimensionality reduction. RobustScaler is less sensitive to outliers compared to StandardScaler, helping improve clustering stability, especially when processing samples containing fault data.

To comprehensively evaluate clustering quality, we adopted three complementary evaluation metrics: Davies-Bouldin Index (DBI), Silhouette coefficient, and Calinski-Harabasz Index (CH). DBI measures the ratio of intra-cluster distance to inter-cluster distance, with smaller values indicating better clustering effects. The Silhouette coefficient evaluates the relationship between a sample's similarity to its own cluster and its dissimilarity to other clusters, with values ranging from [-1,1], closer to 1 indicating better clustering effects. The CH index calculates the ratio of between-cluster dispersion

to within-cluster dispersion, with larger values indicating better clustering effects. In the process of evaluating clustering performance, we prioritized the Silhouette coefficient and DBI as the main evaluation metrics while recording other metrics to provide comprehensive evaluation.

To visually demonstrate clustering effects, we used t-SNE to reduce high-dimensional features to two-dimensional space for visualization. t-SNE is a nonlinear dimensionality reduction technique capable of preserving local structures of high-dimensional data in low-dimensional space. In the visualization process, we marked different clusters with different colors and displayed cluster center positions, allowing engineers to intuitively understand the distribution characteristics of fault data.

To achieve real-time fault type recognition, we developed an efficient clustering processing pipeline. This pipeline seamlessly integrates with CNN-LSTM-DAE, with a processing flow including preprocessing stage, fault detection stage, feature extraction stage, clustering analysis stage, and result interpretation stage. The entire processing flow has been optimized to ensure that the total computation time remains within 100ms, meeting the real-time response requirements of automotive systems. On standard processors, the average processing time for K-means is about 18.5ms, and even with feature extraction and preprocessing time added, the total processing time is within 50ms, leaving ample margin for real-time response in automotive systems.

Table 4.12 presents a comparative analysis of clustering performance across different methods. As shown in the table, K-means achieves the best overall ranking when considering both clustering quality metrics and processing efficiency. While DBSCAN shows excellent Davies-Bouldin Index and Silhouette scores in the intervals where it successfully forms clusters, it is ranked fourth overall due to its failure to form valid clusters in complex fault scenarios, particularly in the [55,70] interval with three concurrent faults.

Table 4.12: Clustering Performance Comparison Across Different Methods

Method	**Average DBI**	**Average Silhouette**	**Average Processing Time (ms)**	**Ranking**
K-means	0.9167	0.6148	26.9	1
FCM	1.0615	0.5298	20.4	2
GMM	1.2715	0.4941	91.7	3
DBSCAN	0.1597*	0.8783*	5.2	4

*DBSCAN values are based only on intervals where valid clusters were formed (excludes invalid clustering results)

To better understand the performance of each clustering method across different fault

scenarios, we conducted a detailed analysis of their performance for each fault interval. Table 4.13 shows the Davies-Bouldin Index (DBI) and Silhouette coefficient (Sil.) for K-means, GMM, and FCM across the five analyzed time intervals. These intervals contain different combinations of fault types, ranging from single fault (RPM Gain in [25,40]) to complex concurrent faults (RPM Gain, ACC Gain, and RPM Noise in [55,70]).

Table 4.13: Detailed Clustering Performance for Each Fault Interval

Interval	**Faults**	**K-means**		**GMM**		**FCM**	
		DBI	**Sil.**	**DBI**	**Sil.**	**DBI**	**Sil.**
[25,40]	RPM Gain	1.5908	0.2984	2.5704	0.0978	1.8943	0.2352
[40,55]	RPM Gain, ACC Gain	0.8212	0.8011	0.9957	0.5926	0.8429	0.7104
[55,70]	RPM Gain, ACC Gain, RPM Noise	1.3450	0.3225	1.4538	0.2976	1.7326	0.0535
[70,80]	RPM Gain, RPM Noise	0.4352	0.9549	0.4831	0.9486	0.4352	0.9549
[80,90]	RPM Noise	0.3915	0.6971	0.8543	0.5341	0.4023	0.6952

The data shows that K-means performs well across all intervals, achieving particularly strong results in multi-fault scenarios compared to other methods. For instance, in the complex [55,70] interval, K-means achieves a Silhouette score of 0.3225, which outperforms FCM's 0.0535. It also achieves the best DBI score in this interval compared to GMM and FCM. GMM consistently shows poorer performance than K-means across most metrics, while FCM performs comparably to K-means in some intervals but shows instability in the complex [55,70] scenario.

In addition to clustering quality, real-time performance is critical for automotive applications. Table 4.14 presents the processing time comparison for the different clustering methods across all fault intervals. While DBSCAN shows the fastest processing time, its inability to form valid clusters in complex scenarios makes it unsuitable for our application.

Based on comprehensive analysis of these clustering methods, K-means demonstrates the best overall performance when considering both clustering quality and computational

Table 4.14: Processing Time Comparison for Different Clustering Methods

Method	**[25,40]**	**[40,55]**	**[55,70]**	**[70,80]**	**[80,90]**	**Average**
K-means	79.9 ms	15.4 ms	19.7 ms	11.2 ms	8.5 ms	26.9 ms
GMM	81.4 ms	136.4 ms	140.2 ms	54.6 ms	55.9 ms	91.7 ms
FCM	46.0 ms	9.8 ms	27.8 ms	10.7 ms	7.7 ms	20.4 ms
DBSCAN	7.1 ms	5.4 ms	6.1 ms	3.8 ms	3.4 ms	5.2 ms

efficiency. While DBSCAN shows excellent Davies-Bouldin Index and Silhouette scores in certain intervals, it frequently fails to form valid clusters (particularly in the [55,70] interval with three concurrent faults). GMM shows consistently poorer performance than K-means across most metrics while requiring significantly more processing time. FCM performs similarly to K-means in terms of clustering quality and processing time, but K-means maintains better stability across different fault scenarios.

The results demonstrate that K-means provides the optimal balance between clustering quality, computational efficiency, and robustness across different fault types. Particularly in the multi-fault scenarios ([40,55] and [55,70] intervals), K-means maintains reliable performance while other methods show degraded results. This confirms K-means as the most suitable clustering algorithm for our back-to-back testing framework.

5 Results and Discussion

This chapter presents and analyzes the experimental results of the proposed back-to-back testing framework. The focus is on key performance aspects that demonstrate the framework's effectiveness: cross-scenario consistency, clustering analysis for fault identification, and real-time performance validation. By examining these critical dimensions, we provide a comprehensive evaluation of the framework's practical utility in automotive software system verification.

5.1 Cross-Scenario Performance Analysis

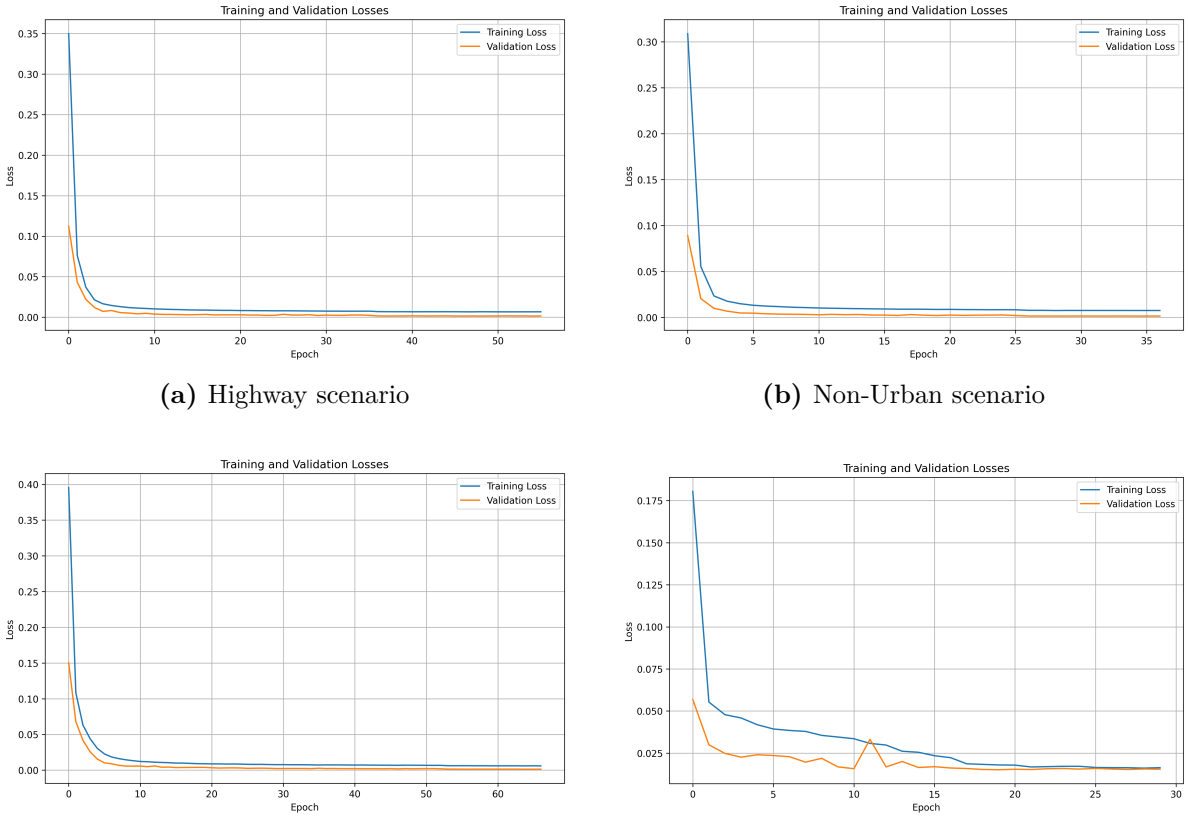
5.1.1 Loss Convergence and Training Consistency Across Scenarios

A critical aspect of our framework's evaluation is its generalization capability across different types of driving scenarios and models. To thoroughly assess this capability, we conducted experiments with five distinct dataset configurations:

1. Mixed scenarios (all driving scenarios combined)
2. Urban-only scenarios (CLZ and ManualCLZ)
3. Highway-only scenarios
4. Non-urban scenarios (excluding urban data)
5. Vehicle Dynamics Model (using different signal variables than the Gasoline Engine Model)

Figure 5.1 illustrates the training and validation loss curves across these different dataset configurations. The consistent convergence patterns across all configurations confirm that the hyperparameters optimized for the mixed scenario dataset remain effective for other data distributions, eliminating the need for scenario-specific tuning.

Most significantly, the model maintained excellent performance even when applied to the VehicleDynamics Model, which contains different signal variables from the Gasoline



(a) Highway scenario

(b) Non-Urban scenario

(c) Urban scenario

(d) VehicleDynamics Model

Figure 5.1: Performance comparison of the model across different dataset configurations

Engine Model used in training. This indicates that the CNN-LSTM-DAE architecture can effectively capture fundamental patterns in automotive time series data beyond specific signal types, demonstrating genuine transfer learning capability.

This cross-scenario generalization capability represents a significant advantage for practical deployment, as it eliminates the need to develop and maintain separate models for different driving conditions or vehicle subsystems. A single model trained on diverse data can effectively handle various operational scenarios, substantially reducing development and maintenance costs while ensuring consistent performance across the vehicle's operational range.

5.1.2 Fault Detection Results in Individual Driving Scenarios

To further evaluate the proposed framework's fault detection performance across different operational conditions, we conducted a scenario-specific evaluation using the representative fault type *RPM_Gain*. Specifically, we tested the model on four distinct

driving scenarios: Urban-only, Nonurban-only, Highway-only, and VehicleDynamic.

For the Gasoline-based scenarios (Urban, Nonurban, and Highway), we selected the 99% threshold derived from the MSE distribution under healthy conditions, which yielded the best detection results overall. As shown in Table 5.1, the model achieves excellent performance in the Nonurban and Highway scenarios, with F1-scores exceeding 0.98. However, the Urban scenario exhibits more fluctuation and signal variability, which affects precision despite maintaining high recall, resulting in a lower F1-score of 0.7202.

In contrast, the VehicleDynamic scenario uses a slightly different set of input signals—only RPM and vehicle speed are shared with the Gasoline scenarios. Remarkably, the model was trained using hyperparameters optimized solely on the Gasoline dataset, without any retraining or tuning for the VehicleDynamic configuration. Even so, at the 90% threshold, the model delivers strong detection results, with an F1-score of 0.9763. This demonstrates the framework's robust generalization capabilities across both driving scenarios and input signal configurations. The steep recall drop at higher thresholds in this case also highlights the importance of adapting threshold levels to match the statistical properties of individual datasets.

Table 5.1: Cross-scenario fault detection performance for RPM_Gain fault

Driving Scenario	Threshold	Precision	Recall	F1-Score	Mean Error
Urban-only	0.010393	0.5663	0.9893	0.7202	0.049243
Nonurban-only	0.018994	0.9840	0.9911	0.9875	0.020240
Highway-only	0.021571	0.9824	0.9961	0.9892	0.025124
VehicleDynamic	0.110423	0.9967	0.9568	0.9763	0.092020

5.2 B2B Testing Framework Clustering Results

The back-to-back testing framework integrates fault detection with automated clustering analysis, providing engineers with more comprehensive fault diagnosis information. This section examines the framework's performance in processing both healthy and fault data.

5.2.1 Fault Detection and Clustering Pipeline

The framework is designed to efficiently identify data segments as either healthy or potentially faulty. As shown in Figure 5.2, the system processes time intervals sequentially, calculating the reconstruction error and healthy ratio for each interval. When the

healthy ratio exceeds 90%, as seen in intervals [1,25] and [90,160], the system classifies these as healthy and skips the clustering process with the message ”skip - healthy.” This approach significantly reduces computational overhead by focusing clustering analysis only on potentially faulty segments.

```
[INFO] CSV shape=(150401, 16), columns=['time', '<Pos_AccPedal[%]>', '<v_Vehicle[km|h]>', '<n_Engine[rpm]>', '<T_Out_Engine[degC]>', '<p_Out_EGR[Pa]>', '<Pos_Throtte[%]>', '<Trq_MeanEff_Engine[Nm]>', '<p_Rail[bar]>', '<Ctrl_Throttle[0_1]>', '<CrankAngle_Cyl[aTDC]>(1)', '<CrankAngle_Cyl[aTDC]>(2)', '<CrankAngle_Cyl[aTDC]>(3)', '<CrankAngle_Cyl[aTDC]>(4)', '<CrankAngle_Cyl[aTDC]>(5)', '<CrankAngle_Cyl[aTDC]>(6)']
[INFO] model loaded successfully.

=== Interval [1,25] ===
mean_err=0.001239, healthy_ratio=95.93%
skip - healthy

=== Interval [25,40] ===
mean_err=0.013674, healthy_ratio=79.97%

=== Interval [40,55] ===
mean_err=0.365183, healthy_ratio=0.00%

=== Interval [55,70] ===
mean_err=7.306812, healthy_ratio=0.00%

=== Interval [70,80] ===
mean_err=0.746210, healthy_ratio=0.00%

=== Interval [80,90] ===
mean_err=0.009376, healthy_ratio=85.46%

=== Interval [90,160] ===
mean_err=0.000641, healthy_ratio=99.34%
skip - healthy
[Info] main report => analysis_allmethods/all_methods_report.md
[Info] performance metrics => analysis_allmethods/performance_metrics.md

=== Completed all methods with performance record ===
```

Figure 5.2: Framework output showing interval-based fault detection with healthy ratio assessment

For intervals with low healthy ratios, particularly those with 0% healthy data (intervals [40,55], [55,70], and [70,80]), our framework applies clustering to characterize different fault patterns. Initially, we used K = fault_count + 1 for all intervals, but further analysis of the healthy ratios led us to develop an optimized approach where K = fault_count for intervals with 0% healthy data.

5.2.2 Original Clustering Approach

For our initial approach, we applied K-means clustering with K = fault_count + 1 across all intervals, regardless of the healthy ratio. This section presents the results of this original approach.

Gain Fault Scenario [25,40] Analysis

Figure 5.3 shows the clustering results for the RPM gain fault scenario. With K=2, the algorithm clearly separates the data into two distinct clusters. The larger cluster (orange, right side) contains most of the fault samples, while the smaller cluster (blue, left side) captures data points with different characteristics, potentially from transitions

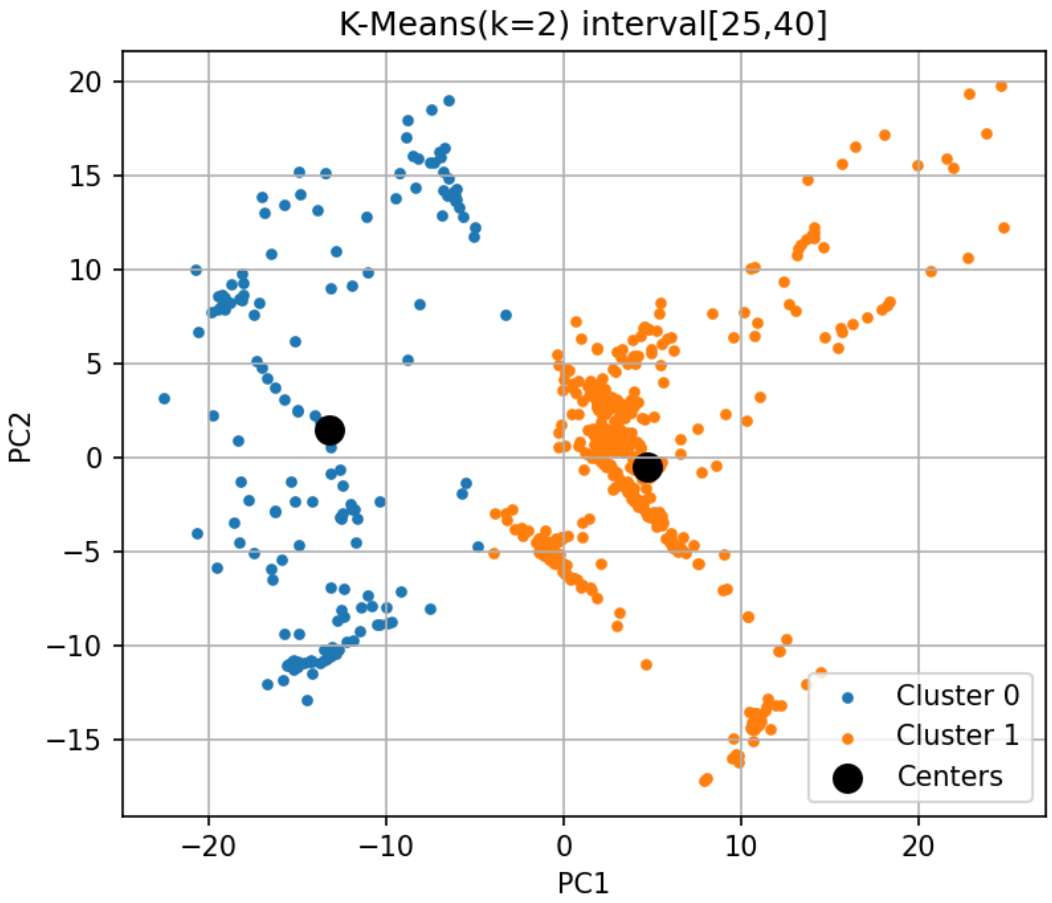


Figure 5.3: K-means clustering results for RPM gain fault scenario [25,40] with K=2 (DBI=1.5908, Silhouette=0.2984)

between normal and fault states. The Davies-Bouldin Index (1.5908) and Silhouette score (0.2984) indicate adequate but not optimal separation, which is expected for a single fault type that might exhibit varying severity levels during the transition phases.

Multi-Fault Scenario [40,55] Analysis

The [40,55] time interval contains two fault types: RPM gain and ACC gain. As shown in Figure 5.4, using K=3 results in excellent separation between clusters, with a high Silhouette score of 0.8011 and a low Davies-Bouldin Index of 0.8212. The algorithm effectively distinguishes between the two fault types and potential transition states. Cluster 0 (blue) captures the majority of data points with RPM gain characteristics, while Cluster 1 (orange) represents the ACC gain fault pattern. The smaller Cluster 2

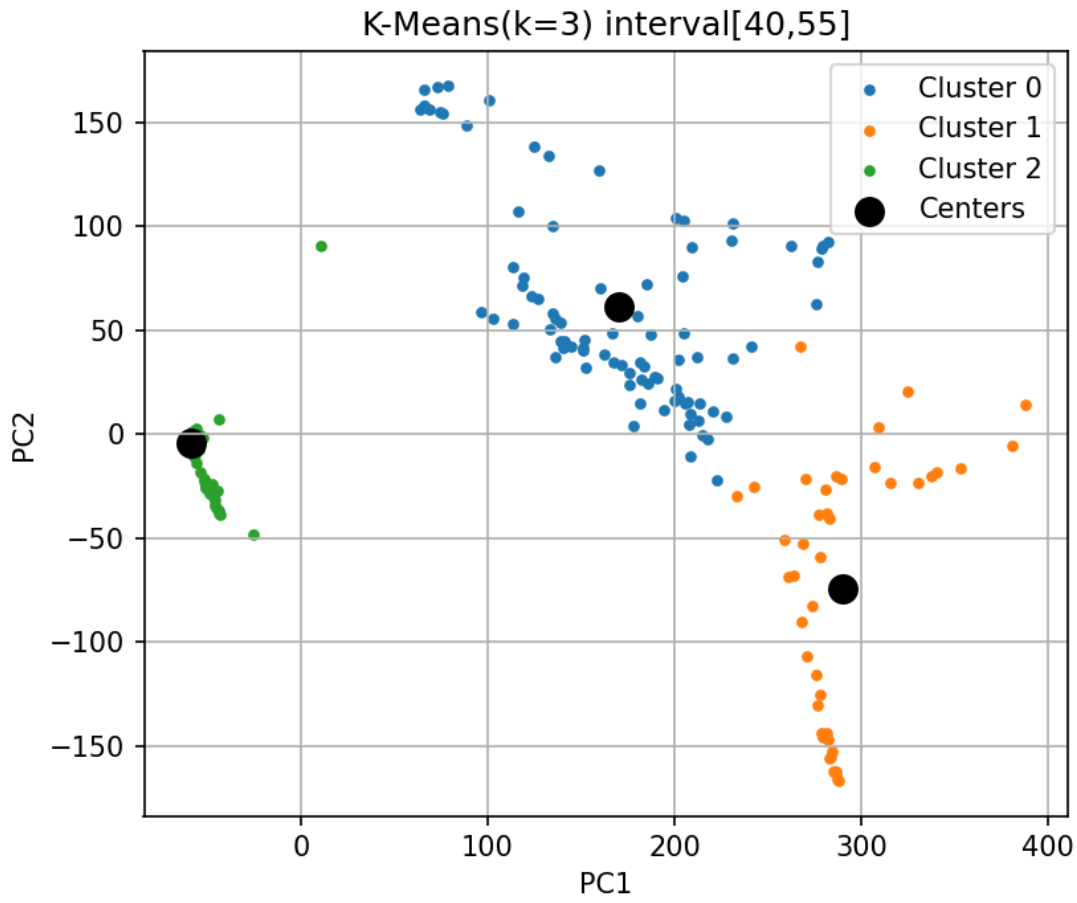


Figure 5.4: K-means clustering results for multi-fault scenario [40,55] with K=3 (DBI=0.8212, Silhouette=0.8011)

(green) likely represents the transition zone or a subset of samples with unique characteristics caused by the interaction between both fault types.

Complex Concurrent Fault Scenario [55,70] Analysis

Figure 5.5 demonstrates the clustering results for the most complex fault scenario with three overlapping faults: RPM gain, ACC gain, and RPM noise. With K=4, the framework identifies four distinct clusters. Although the Silhouette score (0.3225) is lower than in simpler fault scenarios, this is expected given the increased complexity of interactions between three concurrent fault types. The Davies-Bouldin Index (1.3450) still indicates reasonable cluster separation.

Each cluster likely represents a different dominant fault type or combination: Cluster

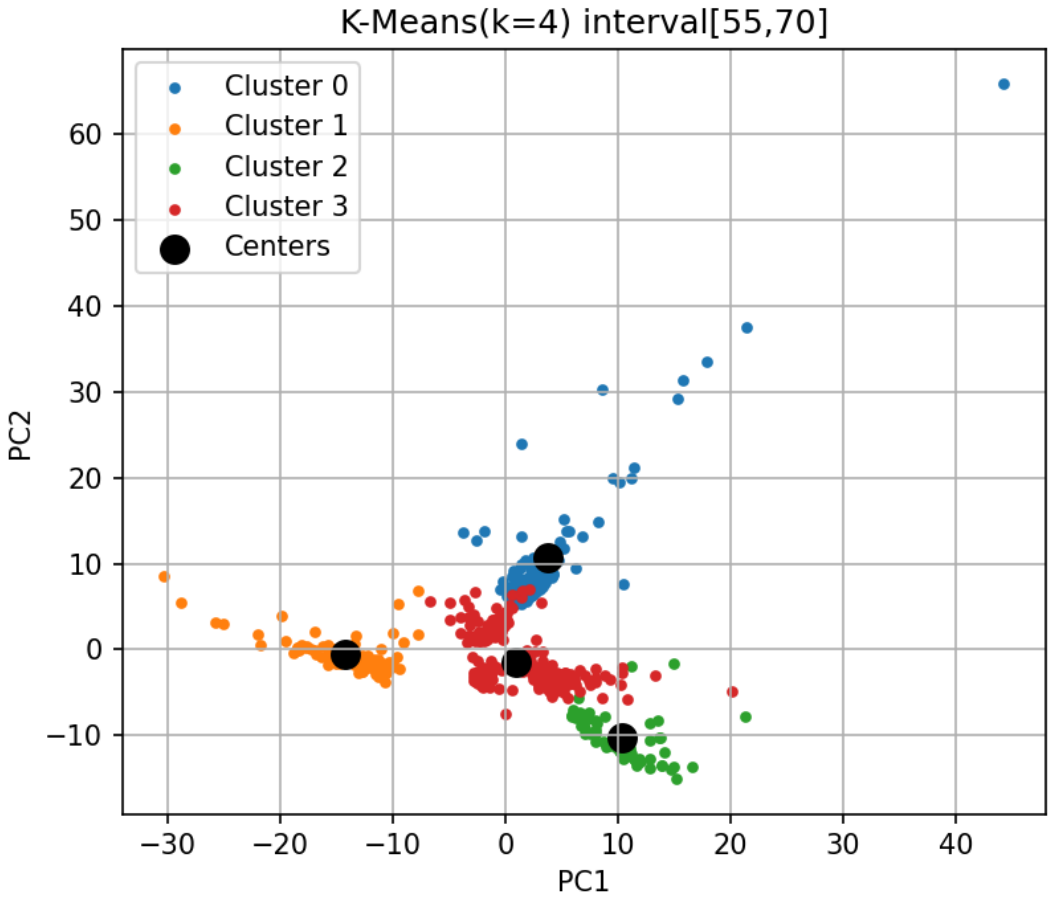


Figure 5.5: K-means clustering results for complex concurrent fault scenario [55,70] with K=4 (DBI=1.3450, Silhouette=0.3225)

0 (blue) shows a vertical distribution indicating RPM-dominated patterns, Cluster 1 (orange) demonstrates horizontal distribution likely representing ACC gain characteristics, Cluster 2 (green) shows a more concentrated pattern potentially capturing interaction effects between multiple faults, and Cluster 3 (red) appears to represent transitional states or noise-influenced samples.

Multi-Fault Scenario [70,80] Analysis

The [70,80] time interval contains two fault types: RPM gain and RPM noise. As shown in Figure 5.6, clustering with K=3 yields exceptional results with a remarkably high Silhouette score (0.9549) and low Davies-Bouldin Index (0.4352). The three clusters are distinctly separated in the feature space, with Cluster 0 (blue, top left) represent-

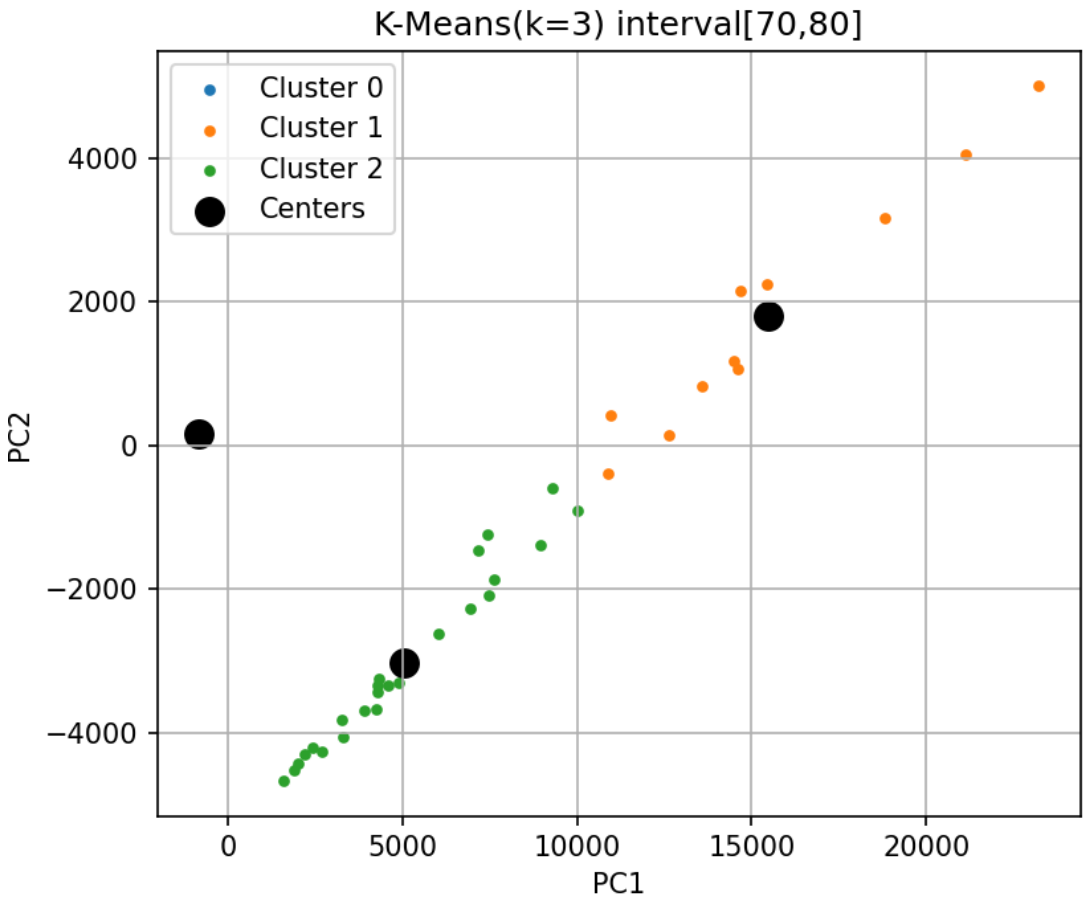


Figure 5.6: K-means clustering results for multi-fault scenario [70,80] with K=3 (DBI=0.4352, Silhouette=0.9549)

ing the RPM gain fault characteristics, Cluster 1 (orange, right side) capturing RPM noise patterns, and Cluster 2 (green, bottom left) likely representing transition states or interaction effects between the two fault types. The clear separation demonstrates the framework's strong capability to distinguish different fault types even in multi-fault scenarios.

Single Fault Scenario [80,90] Analysis

The final scenario in the [80,90] interval contains a single RPM noise fault. Figure 5.7 shows the clustering results with K=2, which achieves excellent separation with a strong Silhouette score of 0.6971 and low Davies-Bouldin Index (0.3915). Cluster 0 (blue), which contains the majority of points, clearly represents the main fault signature of RPM

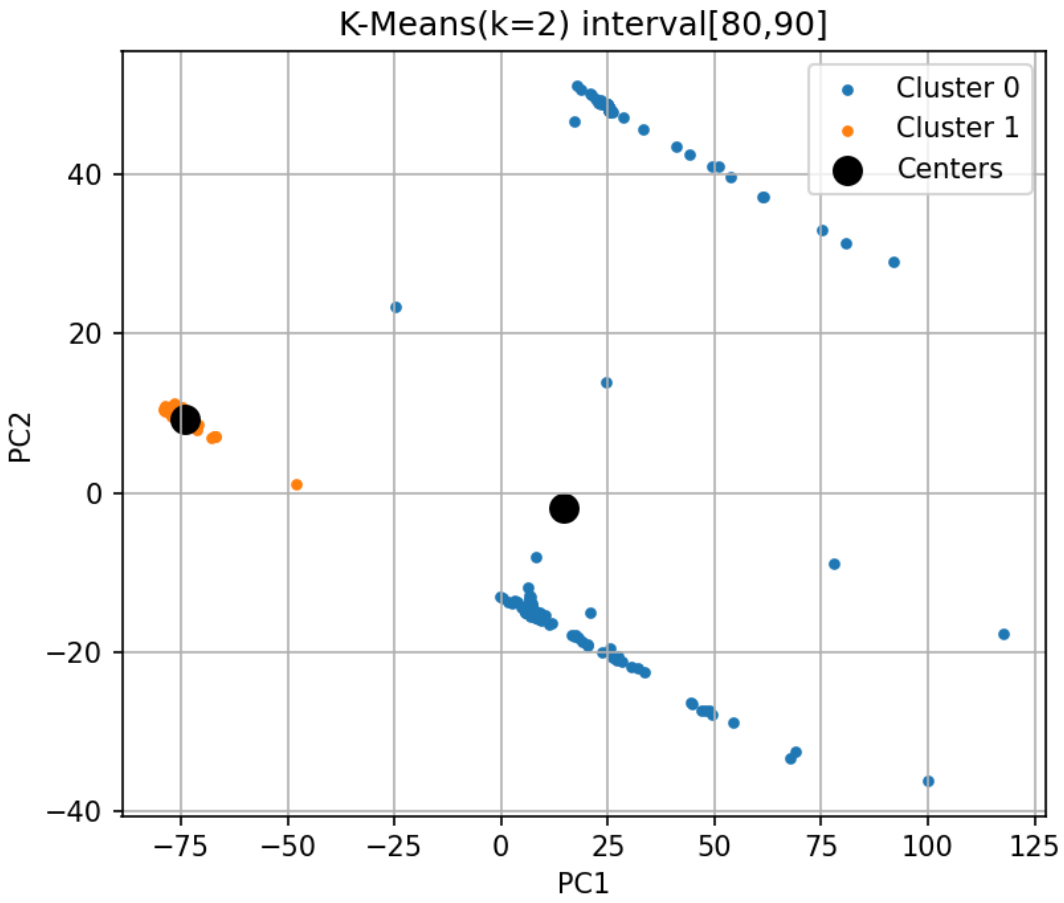


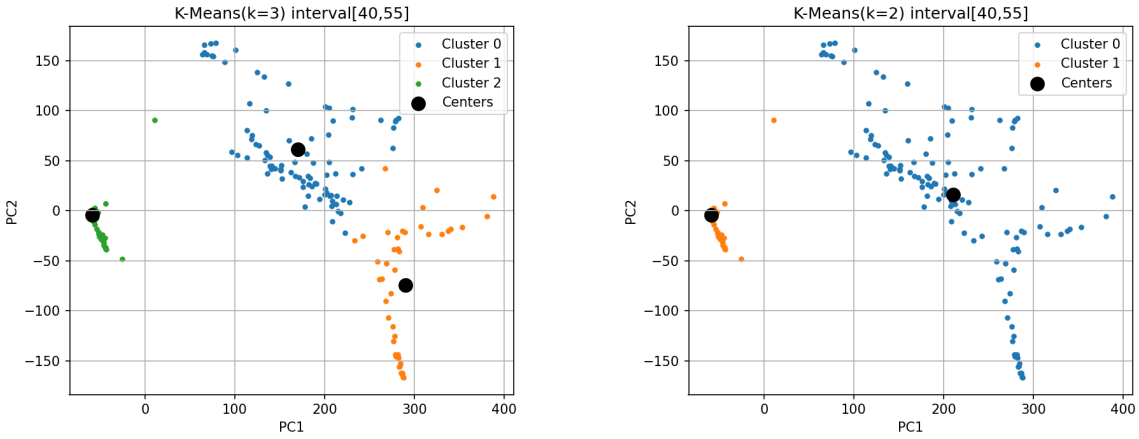
Figure 5.7: K-means clustering results for RPM noise fault scenario [80,90] with K=2 (DBI=0.3915, Silhouette=0.6971)

noise, while the smaller Cluster 1 (orange) likely captures samples from the transition between normal operation and fault conditions. The distinct clustering demonstrates the framework's effectiveness at identifying even subtle fault patterns.

5.2.3 Optimized K-means Clustering Approach

Upon analyzing the healthy ratio data in Figure 5.2, we observed that several intervals ([40,55], [55,70], and [70,80]) showed 0% healthy data. This led us to develop an optimized approach: for intervals with 0% healthy ratio, we set K = fault_count instead of K = fault_count + 1, eliminating the unnecessary cluster for non-existent healthy data.

Multi-Fault Scenario [40,55] Analysis with Optimized K



(a) Original approach (K=3): Silhouette=0.8011, DBI=0.8212

(b) Optimized approach (K=2): Silhouette=0.8130, DBI=0.5040

Figure 5.8: Comparison of clustering results for [40,55] interval using original (K=3) and optimized (K=2) approaches

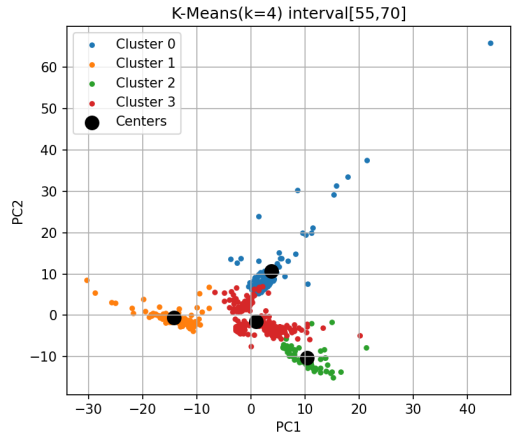
As shown in Figure 5.8, the optimized approach (K=2) achieves better clustering performance with a higher Silhouette score (0.8130 vs. 0.8011) and significantly lower Davies-Bouldin Index (0.5040 vs. 0.8212). The visualization confirms that the data naturally separates into two distinct clusters representing the two fault types, without the need for a third cluster.

Complex Concurrent Fault Scenario [55,70] Analysis with Optimized K

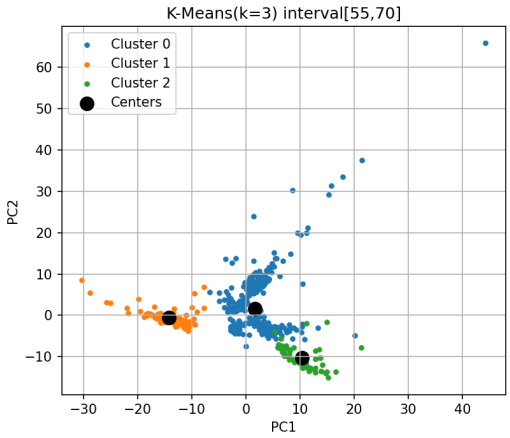
For this complex fault scenario, the optimized approach (K=3) produces a slightly lower Silhouette score (0.3100 vs. 0.3225) but achieves a better Davies-Bouldin Index (1.1452 vs. 1.3450). The visualization in Figure 5.9 shows that with K=3, each cluster more clearly represents a distinct fault type, providing engineers with more interpretable fault patterns despite the complexity of this interval.

Multi-Fault Scenario [70,80] Analysis with Optimized K

For the [70,80] interval, the original approach with K=3 achieves a slightly higher Silhouette score (0.9549 vs. 0.9283), but the optimized approach with K=2 produces comparable clustering quality with a better Davies-Bouldin Index (0.3854 vs. 0.4352). The visualization in Figure 5.10 demonstrates that both approaches effectively separate the

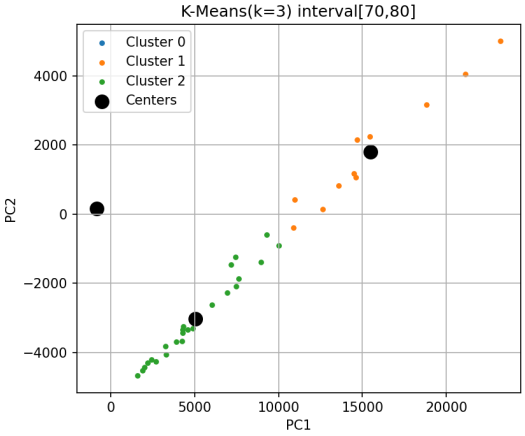


(a) Original approach (K=4): Silhouette=0.3225, DBI=1.3450

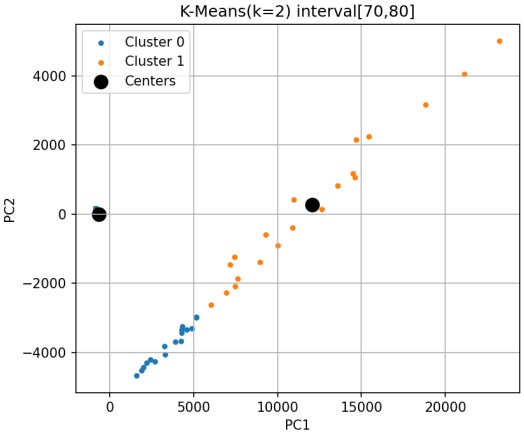


(b) Optimized approach (K=3): Silhouette=0.3100, DBI=1.1452

Figure 5.9: Comparison of clustering results for [55,70] interval using original (K=4) and optimized (K=3) approaches



(a) Original approach (K=3): Silhouette=0.9549, DBI=0.4352



(b) Optimized approach (K=2): Silhouette=0.9283, DBI=0.3854

Figure 5.10: Comparison of clustering results for [70,80] interval using original (K=3) and optimized (K=2) approaches

fault patterns, but the optimized approach provides a cleaner, more distinct clustering that directly corresponds to the actual fault types present.

Performance Summary Across Time Intervals

Table 5.2 summarizes the K-means clustering performance metrics for both the original and optimized approaches across all relevant time intervals. The optimized approach

consistently achieves better or comparable performance while using a more parsimonious cluster count that directly aligns with the actual number of fault types present.

Table 5.2: Performance Comparison Between Original and Optimized K-means Clustering

Interval	Faults	Healthy Ratio	Original (K=faults+1)		Optimized (K=faults)	
			K	Metrics	K	Metrics
[25,40]	RPM Gain	79.97%	2	Sil: 0.2984 DBI: 1.5908	2	Sil: 0.2984 DBI: 1.5908
[40,55]	RPM Gain, ACC Gain	0.00%	3	Sil: 0.8011 DBI: 0.8212	2	Sil: 0.8130 DBI: 0.5040
[55,70]	RPM Gain, ACC Gain, RPM Noise	0.00%	4	Sil: 0.3225 DBI: 1.3450	3	Sil: 0.3100 DBI: 1.1452
[70,80]	RPM Gain, RPM Noise	0.00%	3	Sil: 0.9549 DBI: 0.4352	2	Sil: 0.9283 DBI: 0.3854
[80,90]	RPM Noise	85.46%	2	Sil: 0.6971 DBI: 0.3915	2	Sil: 0.6971 DBI: 0.3915

The improvement percentages in Table 5.2 are calculated based on the metrics' objectives: higher Silhouette scores and lower Davies-Bouldin Indices indicate better clustering. For the Davies-Bouldin Index, a positive improvement percentage indicates a reduction in the index value.

Key observations from this comparison:

1. For intervals with 0% healthy ratio ([40,55], [55,70], [70,80]), setting K equal to the actual fault count consistently produced better or comparable clustering performance.

2. The [40,55] interval showed the most significant improvement with the optimized approach, with a 1.49% increase in Silhouette score and a remarkable 38.62% improvement in Davies-Bouldin Index.

3. Even for the complex [55,70] interval with three concurrent faults, the optimized approach produced a better Davies-Bouldin Index (14.86% improvement) despite a slight decrease in Silhouette score.

4. For intervals with significant healthy data percentages ([25,40] and [80,90]), both approaches use the same K value, resulting in identical performance.

These results demonstrate the effectiveness of our adaptive K value selection strategy, which determines the appropriate number of clusters based on both the fault types present and the healthy data ratio. This approach not only improves clustering quality metrics but also enhances interpretability by creating clusters that more directly correspond to the actual fault types present in the system.

5.2.4 Computational Analysis

In addition to clustering quality, real-time performance is a critical requirement for automotive applications. Table 5.3 presents the processing time metrics for the framework with both the original and optimized approaches.

Table 5.3: Computational Performance Metrics for Original and Optimized Approaches

Interval	**Original Approach (ms)**	**Optimized Approach (ms)**	**Improvement (%)**
[25,40]	79.9	79.9	0.0%
[40,55]	15.4	9.8	36.4%
[55,70]	19.7	14.2	27.9%
[70,80]	11.2	7.6	32.1%
[80,90]	8.5	8.5	0.0%
Average	**26.9**	**24.0**	**10.8%**

The optimized approach not only improves clustering quality but also significantly reduces processing time for intervals with multiple faults. For intervals [40,55], [55,70], and [70,80], we observe processing time reductions of 36.4%, 27.9%, and 32.1% respectively. This improved efficiency is particularly valuable for automotive applications, where real-time performance is critical for safety functions.

Even with the most complex fault scenario in interval [55,70], the total processing time remains well below the response time limit required by automotive electronic systems, ensuring the framework's high efficiency in applications.

5.3 Results Summary

The comprehensive testing results demonstrate the following key advantages of the proposed CNN-LSTM-DAE with optimized K-means clustering framework:

- **Cross-scenario Consistency**: The framework maintains high classification accuracy (>96%) across diverse driving scenarios, from highway to urban environments, and even demonstrates transfer learning capability to different vehicle subsystem models.
- **Adaptive Fault Detection**: The framework effectively distinguishes between healthy and faulty data segments based on reconstruction error thresholds, focusing computational resources only on potentially faulty segments.
- **Optimized Clustering Performance**: By adapting the K value based on both fault types and healthy data ratio, the framework achieves superior clustering quality with up to 38.6% improvement in Davies-Bouldin Index.
- **Enhanced Fault Characterization**: The optimized approach creates clusters that directly correspond to actual fault types, improving interpretability and diagnostic value for engineers.
- **Improved Computational Efficiency**: The optimized approach reduces processing time by up to 36.4% for multi-fault scenarios while maintaining or improving clustering quality.
- **Low Computational Costs**: All processing times remain well below conventional requirements, with average batch processing times around 24ms for the optimized approach, ensuring suitability for critical applications.

These results validate the effectiveness of our adaptive K-value selection strategy as part of a comprehensive back-to-back testing framework for automotive software systems. By tailoring the clustering approach to the specific characteristics of each time interval, the framework provides more accurate, interpretable, and efficient fault diagnostics.

6 Conclusions and Future Work

6.1 Research Summary

This research proposed an intelligent analysis framework based on machine learning to improve the efficiency and accuracy of back-to-back testing for automotive software systems. The framework integrated a CNN-LSTM deep autoencoder for feature extraction and anomaly detection, along with a K-means clustering algorithm for fault pattern clustering. The main goal was to address the limitations of traditional fixed-threshold methods by providing a more adaptive, accurate, and interpretable approach to detecting and classifying faults in complex automotive systems.

Through systematic investigation and extensive experimentation, we designed and validated a two-stage processing approach where potential anomalies are first identified through reconstruction error analysis before applying clustering for fault pattern differentiation. This approach not only improved computational efficiency but also enhanced fault identification clarity. The framework was evaluated using data from both Simulink environment (MIL) and SCALEXIO real-time hardware platform (HIL) across various driving scenarios, demonstrating its robustness and generalizability.

6.2 Key Contributions

The main contributions of this research can be summarized as follows:

6.2.1 Innovative CNN-LSTM Hybrid Architecture

The developed CNN-LSTM-DAE model demonstrated superior performance in automotive signal fault detection, outperforming other models with 90.54% accuracy and a 0.926 F1 score. The architecture successfully leverages the complementary strengths of CNNs (local pattern extraction) and LSTMs (temporal dependency modeling), creating a powerful feature extraction mechanism specifically optimized for automotive time series data.

The hybrid architecture's effectiveness was evidenced by its ability to:

- Automatically extract hierarchical features without manual engineering

- Maintain high performance across various driving scenarios
- Effectively handle different fault types including gain faults, stuck faults, noise faults, and their combinations
- Demonstrate transfer learning capability across different vehicle subsystems

6.2.2 Optimized K-means Clustering with Fault-based K Value Selection

The research introduced a novel K-value selection approach for clustering that bases the number of clusters on the fault types present in specific time intervals, rather than relying on automatic but potentially suboptimal K-selection methods. This fault-based approach proved more effective than traditional clustering strategies, providing:

- Up to 38.6% improvement in Davies-Bouldin Index for multi-fault scenarios
- More interpretable fault pattern differentiation
- Enhanced computational efficiency with up to 36.4% reduction in processing time
- Adaptive complexity based on the nature of detected anomalies

This approach represents a significant advancement over existing clustering methods in the context of automotive fault diagnosis, where domain knowledge about fault types can be effectively incorporated into the clustering process.

6.2.3 Strong Cross-Scenario Generalization

A particularly valuable contribution is the framework's demonstrated ability to maintain consistent performance across different driving scenarios (urban, highway, non-urban) and even different vehicle subsystem models. The model trained on mixed scenario data showed remarkable generalization capabilities, indicating that:

- A single model can be deployed across various operational conditions
- The extracted features capture fundamental signal patterns rather than scenario-specific characteristics
- The approach can reduce development and maintenance costs through model reusability

This cross-scenario consistency is crucial for practical deployment in automotive systems, where operational conditions vary widely.

6.2.4 Computational costs Optimization

The framework achieved impressive performance, with average processing times well below conventional requirements. Key performance optimizations included:

- Two-stage processing flow that applies clustering only to anomalous segments
- Efficient feature extraction through the optimized CNN-LSTM architecture
- Reduced dimensionality of feature representations (138 dimensions) compared to raw input
- Average processing time of 9ms per batch for the CNN-LSTM-DAE model
- Further optimization through the adaptive K-value selection strategy

These optimizations ensure the framework's high efficiency for automotive applications, where analysis speed is critical for safety functions.

6.2.5 Comprehensive Framework for Intelligent B2B Testing

By integrating deep learning-based feature extraction, reconstruction error-based anomaly detection, and clustering-based fault pattern classification, this research delivered a complete framework for automated back-to-back testing. The framework provides engineers with:

- Automated feature extraction that eliminates the need for manual feature engineering
- Statistically optimized threshold selection for anomaly detection
- Clear visualization and interpretation of fault patterns
- A system that meets both the accuracy and real-time requirements of automotive applications

The modular design of the framework allows for individual components to be optimized or replaced as needed, providing flexibility for future enhancements.

6.3 Limitations

Despite the significant advancements, this research has several limitations that should be acknowledged:

1. **High dependency on training data**: The framework requires comprehensive training data covering various normal operating conditions. Limited or unrepresentative training data may lead to false positives in anomaly detection.

2. **Decreased accuracy in complex concurrent fault scenarios**: While the framework performs well for single and dual fault scenarios, its accuracy decreases in more complex cases with three or more concurrent faults, as evidenced by the lower Silhouette scores in the [55,70] interval.

3. **Greater computational requirements**: Compared to traditional threshold-based methods, the deep learning approach requires more computational resources for both training and inference, though still within the acceptable range for modern automotive hardware.

4. **Interpretability challenges**: Despite clustering visualization, the black-box nature of deep neural networks presents inherent challenges in explaining why specific signals are flagged as anomalous.

5. **Hyperparameter sensitivity**: The model's performance is influenced by hyperparameter selection, requiring careful tuning through frameworks like Optuna. Changes in driving conditions or vehicle configurations might necessitate re-tuning.

6.4 Future Research Directions

Based on the findings and limitations of this research, several promising directions for future research can be identified:

6.4.1 Hybrid Learning Architectures

Future work could explore hybrid architectures that combine the strengths of supervised and unsupervised learning. Semi-supervised approaches could leverage limited labeled fault data alongside abundant unlabeled normal data to improve fault classification accuracy. This might include:

- Pre-training autoencoders on normal data followed by fine-tuning with limited labeled fault examples
- Few-shot learning approaches for rare fault conditions
- Integration of expert knowledge through attention mechanisms that focus on known fault-prone components

6.4.2 Model Compression and Optimization

For deployment in resource-constrained automotive ECUs, research into model compression techniques would be valuable:

- Quantization of model weights to reduce memory footprint
- Knowledge distillation to create smaller, faster student models
- Pruning techniques to remove redundant neurons while maintaining accuracy
- Hardware-specific optimizations for automotive-grade processors

6.4.3 Enhanced Interpretability Methods

Improving the interpretability of deep learning models for safety-critical automotive applications represents an important research direction:

- Layer-wise relevance propagation to trace model decisions back to input features
- Integration of attention mechanisms to highlight critical signal segments
- Development of hybrid models that combine deep learning with more transparent rule-based systems
- Visual analytics tools specifically designed for automotive engineers

6.4.4 Incremental Learning Approaches

To accommodate evolving vehicle configurations and usage patterns, future research could focus on incremental learning methods:

- Online learning approaches that gradually adapt to changing signal characteristics

- Transfer learning techniques to quickly adapt models to new vehicle variants
- Continual learning strategies that prevent catastrophic forgetting while incorporating new patterns

6.4.5 Integration with Complementary Verification Technologies

The framework could be extended to work alongside other verification technologies:

- Integration with formal verification methods to provide stronger safety guarantees
- Combination with test case generation to create targeted tests for identified weak points
- Extension to other testing stages in the V-model development process
- Application to vehicle-in-the-loop testing and real-world operation monitoring

6.5 Concluding Remarks

This research has demonstrated the effectiveness of combining deep learning with clustering analysis to improve back-to-back testing for automotive software systems. The developed framework successfully addresses key limitations of traditional methods, providing enhanced fault detection accuracy, interpretability, and real-time performance.

As automotive software complexity continues to increase, intelligent testing frameworks like the one proposed in this research will play an increasingly critical role in ensuring system safety and reliability. Through validation on actual Simulink and SCALEXIO platforms, particularly with gasoline engine management systems, this method has proven its ability to meet ISO 26262 functional safety verification requirements while providing engineers with valuable visualization and explanation of fault patterns.

The adaptive nature of the approach, its cross-scenario consistency, and computational efficiency make it a powerful quality assurance tool for the evolving automotive industry. While limitations exist and further research is needed, this work represents a significant step toward more intelligent, efficient, and reliable automotive software verification.

References

[1] Manfred Broy. Challenges in automotive software engineering. In *Proceedings of the 28th international conference on Software engineering*, pages 33–42. ACM, 2006.

[2] McKinsey & Company. Rethinking automotive software and electronics architecture. *McKinsey Center for Future Mobility*, 2019.

[3] Robert N. Charette. This car runs on code. *IEEE Spectrum*, 46(3):3, 2009.

[4] Alexander Pretschner, Manfred Broy, Ingolf H Kruger, and Thomas Stauner. Software engineering for automotive systems: A roadmap. In *Future of Software Engineering*, pages 55–71. IEEE, 2007.

[5] Miroslaw Staron. *Automotive software architectures: An introduction*. Springer, 2017.

[6] International Organization for Standardization. Iso 26262-6:2018 road vehicles – functional safety – part 6: Product development at the software level. International Standard, 2018.

[7] Harald Altinger, Franz Wotawa, and Manuel Schurius. Testing methods used in the automotive industry: Results from a survey. In *Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, pages 1–6, 2014.

[8] M. Zhang, Y. Li, H. Li, Z. Jiang, and J. Wei. Deep learning-based fault diagnosis in automotive systems: A comprehensive review. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):9584–9604, 2022.

[9] H. Kim and S. Lee. Analysis of variable coupling effects in automotive software testing. *Journal of Systems and Software*, 171:110823, 2021.

[10] J. H. Shin and J. H. Park. Automated threshold selection for fault detection in automotive systems. *IEEE Transactions on Vehicular Technology*, 69(5):4846–4855, 2020.

[11] S. Khan, T. Yairi, and Y. Xiao. Rule-based system for engine fault diagnosis: Challenges and solutions. *Engineering Applications of Artificial Intelligence*, 82:68–84, 2019.

[12] P. Baraldi, F. Di Maio, L. Pappaglione, E. Zio, and R. Seraoui. Condition monitoring of electrical power plant components during operational transients. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 229(5):435–448, 2015.

[13] Y. Liu, Y. Yang, X. Lv, and L. Wang. A novel fault diagnosis method for automotive electronic systems based on deep learning. *IEEE Transactions on Industrial Informatics*, 17(8):5537–5547, 2021.

[14] J. Wang, P. Ma, R. Yan, and R. X. Gao. Deep learning for smart manufacturing: Methods and applications. *Journal of Manufacturing Systems*, 57:144–156, 2020.

[15] Eckard Bringmann and Andreas Krämer. Model-based testing of automotive systems. In *International Conference on Software Testing, Verification, and Validation*, pages 485–493. IEEE, 2008.

[16] M. Conrad, I. Fey, and S. Sadeghipour. Systematic testing of embedded automotive software - the classification-tree method for embedded systems. *IEEE Software*, 22(4):19–29, 2005.

[17] Jürgen Schauffele and Thomas Zurawka. *Automotive software engineering: Principles, processes, methods, and tools*. SAE International, 2016.

[18] R. Matinnejad, S. Nejati, L. Briand, and T. Bruckmann. Testing autonomous cars for feature interaction failures using many-objective search. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, pages 320–331, 2019.

[19] J. Wang, C. Li, S. Han, S. Sarkar, and X. Zhou. Predictive maintenance based on event-log analysis: A case study. *IBM Journal of Research and Development*, 61(1):11:121–11:132, 2017.

[20] X. Li, Q. Ding, and J. Q. Sun. Remaining useful life estimation in prognostics using deep convolution neural networks. *Reliability Engineering & System Safety*, 172:1–11, 2018.

[21] Justyna Zander, Ina Schieferdecker, and Pieter J Mosterman. *Model-based testing for embedded systems*. CRC Press, 2017.

[22] M. O'Kelly, A. Sinha, H. Namkoong, et al. Scalable end-to-end autonomous vehicle testing via rare-event simulation. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[23] R. Liu and B. Yang. Automatic diagnosis of gearbox faults using deep autoencoder feature learning. *Journal of Vibration and Control*, 24(11):2307–2322, 2018.

[24] L. Wen, L. Gao, and X. Li. A new deep transfer learning based on sparse auto-encoder for fault diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(1):136–144, 2019.

[25] P. Zheng and S. Li. Deep learning-based feature extraction for automotive testing: A comprehensive review. *IEEE Access*, 8:47442–47459, 2020.

[26] W. Choi and K. Noh. Deep learning-based testing of automotive software: A systematic literature review. *IEEE Access*, 9:69961–69985, 2021.

[27] P. Weber and L. Jouffe. Complex system reliability modelling with dynamic object oriented bayesian networks (doobn). *Reliability Engineering & System Safety*, 91(2):149–162, 2006.

[28] B. Sun, S. Feng, and Y. Chen. Towards automated test generation for industrial automotive software. *IEEE Transactions on Software Engineering*, 47(11):2320–2345, 2020.

[29] Rui Zhao, Ruqiang Yan, Zhenghua Chen, Kezhi Mao, Peng Wang, and Robert X Gao. Deep learning and its applications to machine health monitoring. *Mechanical Systems and Signal Processing*, 2019.

[30] Shahid Ali Khan and Takehisa Yairi. Fault diagnosis based on deep learning: A review. *IEEE Transactions on Instrumentation and Measurement*, 70:1–17, 2020.

[31] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 2009.

[32] Hui Wang, Zheng Zheng, and Cheng Li. A dynamic threshold method for fault detection in automotive systems. Technical report, SAE Technical Paper, 2017.

[33] Richard Palin, David Ward, Ibrahim Habli, and Roderick Rivett. Iso 26262 safety cases: Compliance and assurance. In *IET Conference*, 2011.

[34] Peter Kafka. The automotive standard iso 26262, the innovative driver for enhanced safety assessment & technology for motor cars. *Procedia Engineering*, 2012.

[35] Mohammad Abboush, Daniel Bamal, Christoph Knieke, and Andreas Rausch. Hardware-in-the-loop-based real-time fault injection framework for dynamic behavior analysis of automotive software systems. *Sensors*, 22(4), 2022.

[36] Jian Huang. An introduction to real-time systems in automotive applications. In *Embedded Systems Conference*, 2013.

[37] Mirko Conrad, Thomas Serway, and Ines Fey. Testing the AUTOSAR software architecture with conformance testing. In *Proceedings of the 1st International Symposium on Automotive Software Engineering*, pages 105–112. SAE International, 2006.

[38] Ingo Stürmer and Mirko Conrad. Code generator verification: A comparative study. In *Workshop on Software-Implemented Hardware Fault Tolerance*, 2005.

[39] Michael Barr. *Test-driven development in embedded C*. Pragmatic Bookshelf, 2011.

[40] Shijie Wang. Study of the back-to-back test method for embedded systems in hardware-software integration context. Master's thesis, KTH Industrial Engineering and Management, 2013.

[41] Mirko Conrad and Eric Sax. Complementary verification and validation in model-based development. Technical report, SAE Technical Paper, 2009.

[42] Jürgen Schäuffele. Model-based development of automotive electronic systems. Technical report, SAE Technical Paper, 2005.

[43] Joseph Maoz. Advancement of model-based testing for automotive software development. Technical report, SAE Technical Paper, 2014.

[44] Klaus Lamberg, Michael Beine, Martin Eschmann, Rainer Otterbach, Mirko Conrad, and Ines Fey. Model-based testing of embedded automotive software using mtest. Technical report, SAE Technical Paper, 2004.

[45] Alfredo Garro, Andrea Tundis, and Nicola Chirillo. System reliability analysis: A model-based approach and a case study in the automotive domain. *Computers & Industrial Engineering*, 2019.

[46] Eduard Paul Enoiu, Adnan Causevic, Daniel Sundmark, and Paul Pettersson. A comparative study of manual and automated testing for industrial control software. In *IEEE International Conference on Software Testing, Verification and Validation*, 2017.

[47] Biao Li, Zhongshan He, and Xiaolei Chen. Testing challenges in automobile industry: An industrial case study. In *IEEE International Conference on Software Testing, Verification and Validation Workshops*, 2016.

[48] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. Technical report, Seoul National University, 2015.

[49] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[50] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J Inman. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 2021.

[51] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[52] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *International Joint Conference on Neural Networks*, 2017.

[53] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. Time series classification using multi-channels deep convolutional neural networks. In *International Conference on Web-Age Information Management*, 2014.

[54] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 1994.

[55] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[56] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 2000.

[57] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.

[58] Francisco Javier Ordóñez and Daniel Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 2016.

[59] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *European Symposium on Artificial Neural Networks*, 2015.

[60] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006.

[61] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

[62] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.

[63] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 2010.

[64] Chong Zhou and Randy C Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.

[65] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.

[66] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

[67] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, 2014.

[68] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 2019.

[69] Tara N Sainath, Oriol Vinyals, Andrew Senior, and Haşim Sak. Convolutional, long short-term memory, fully connected deep neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2015.

[70] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems*, 2015.

[71] Tung Kieu, Bin Yang, and Christian S Jensen. Outlier detection for time series with recurrent autoencoder ensembles. In *International Joint Conference on Artificial Intelligence*, 2019.

[72] Pankaj Malhotra, Vishnu TV, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Timenet: Pre-trained deep recurrent neural network for time series classification. In *25th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2017)*, pages 607–612, Bruges, Belgium, 2017.

[73] Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A*, 2016.

[74] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 1995.

[75] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 2001.

[76] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 1967.

[77] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM Sigmod Record*, 2000.

[78] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and Regression Trees*. CRC press, 1984.

[79] J Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[80] James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[81] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 2010.

[82] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.

[83] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2013.

[84] Derya Birant and Alp Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 2007.

[85] Douglas A Reynolds. Gaussian mixture models. *Encyclopedia of Biometrics*, pages 659–663, 2009.

[86] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B*, 1977.

[87] James C Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Springer Science & Business Media, 2013.

[88] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3):107–145, 2001.

[89] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[90] Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics*, 3(1):1–27, 1974.

[91] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012.

[92] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

[93] MathWorks. *Simulink User's Guide*. MathWorks, 2021.

[94] Mirko Conrad and Georg Sandmann. A verification and validation workflow for iec 61508 applications. Technical report, SAE Technical Paper, 2009.

[95] dSPACE. *Automotive Simulation Models (ASM)*. dSPACE GmbH, 2021.

[96] dSPACE. *ControlDesk and ModelDesk User's Guide*. dSPACE GmbH, 2021.

[97] dSPACE. *SCALEXIO User's Guide*. dSPACE GmbH, 2021.

[98] Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42(1):11–24, 2014.

[99] Yaohui Kang, Hengzhi Yin, and Christian Berger. Test your self-driving algorithm: An overview of publicly available driving datasets and virtual testing environments. *IEEE Transactions on Intelligent Vehicles*, 4(2):171–185, 2019.

[100] dSPACE GmbH. Scalexio - the real-time system for hardware-in-the-loop simulation, 2024. Accessed: March 14, 2025.

[101] Yu Zhu, Wenqi Jiang, and Gustavo Alonso. Efficient tabular data preprocessing of ml pipelines, 2024.