R PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND

SCIENTIFIC RESEARCH

University of Echahid Hamma Lakhdar - El-Oued

Faculty of Exact Sciences - Informatique department

Option: Internet of Things and Cybersecurity

Master's Thesis

Machine learning to detect and classify engine and car faults through sound

Presented by:

- Baha Eddine Mennani

Supervisor:

- Dr.Mohammed Charaf Eddine MEFTAH

- Dr. Messaoud Abbas — **Presedent**
- Dr. Zayez Fawzi — **Examiner**

Academic year 2024/2025

Abstract

Engine fault diagnosis is a critical task in automotive maintenance, and recent advances in artificial intelligence have enabled new possibilities for non-invasive and efficient detection methods. This work presents a comprehensive approach for detecting and classifying engine faults using sound signals combined with machine learning (ML) and deep learning (DL) models. Real-world engine audio recordings, both healthy and faulty, were collected and processed to extract MFCC and spectrogram features. Multiple models—including K-NN, Random Forest, XGBoost, ANN, CNN, and LSTM—were trained and evaluated under both binary and multiclass classification scenarios for gasoline and diesel engines. The results demonstrate the potential of AI-based acoustic analysis as a reliable tool for early fault detection, with CNN and LSTM models showing superior performance in complex classification tasks. A lightweight TFLite-based system was also implemented for real-time mobile diagnosis.

Keywords: Engine fault diagnosis, machine learning, deep learning, MFCC, CNN, sound classification, acoustic analysis, TFLite, mobile diagnosis.

Résumé

Le diagnostic des pannes moteur est une tâche essentielle dans la maintenance automobile. Grâce aux progrès récents de l'intelligence artificielle, de nouvelles méthodes non invasives et efficaces ont vu le jour. Ce travail propose une approche complète de détection et classification des pannes moteur à partir de signaux sonores, en utilisant des modèles d'apprentissage automatique (ML) et d'apprentissage profond (DL). Des enregistrements audio réels de moteurs sains et défectueux ont été collectés et analysés via des caractéristiques MFCC et spectrogrammes. Plusieurs modèles (K-NN, Random Forest, XGBoost, ANN, CNN, LSTM) ont été évalués dans des scénarios de classification binaire et multiclasses, pour moteurs à essence et diesel. Les résultats montrent que l'analyse acoustique basée sur l'IA constitue une solution fiable pour la détection précoce des défauts, avec des performances remarquables des modèles CNN et LSTM. Une version légère du système a également été implémentée en TFLite pour un diagnostic mobile en temps réel.

Mots-clés : Diagnostic des pannes moteur, apprentissage automatique, apprentissage profond, MFCC, CNN, classification sonore, analyse acoustique, TFLite, diagnostic mobile.

ملخص

يُعد تشخيص أعطاب المحركات من المهام الحيوية في صيانة المركبات، وقد فتحت تقنيات الذكاء الاصطناعي آفاقًا جديدة نحو وسائل فعّالة وغير تدخّلية للكشف عن هذه الأعطاب. في هذا العمل، تم تقديم منهجية شاملة لتصنيف أعطاب المحركات باستخدام الإشارات الصوتية ونماذج التعلم الآلي والتعلم العميق. تم تسجيل أصوات محركات سليمة ومعطوبة في ظروف واقعية، وتم استخراج خصائص MFCC والمخططات الطيفية منها. ثم تم تدريب وتقييم عدة نماذج، من بينها K-NN و Random Forestو XGBoost و ANN و CNN و LSTM، في سيناريوهات تصنيف ثنائي ومتعدد الفئات لكل من محركات البنزين والديزل. أظهرت النتائج كفاءة عالية للتحليل الصوتي المعتمد على الذكاء الاصطناعي، خاصة باستخدام نماذج CNN و .LSTM كما تم تطوير نظام خفيف باستخدام TFLite لتوفير تشخيص فوري عبر تطبيق هاتف محمول.

الكلمات المفتاحية : تشخيص أعطاب المحركات، تعلم آلي، تعلم عميق، MFCC، CNN، تصنيف صوتي، تحليل صوتي، TFLite، تشخيص محمول.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

General Introduction

GENERAL INTRODUCTION

Background

In the era of Industry 4.0, the integration of intelligent systems into traditional mechanical domains has significantly transformed engineering practices in maintenance, diagnostics, and optimization. One major advancement is the application of machine learning in predictive maintenance, where algorithms can analyze sensor data to detect early signs of failure. Among various sensing modalities, acoustic signals have proven to be particularly valuable, especially for diagnosing internal combustion engines. These sounds often carry subtle indicators of mechanical wear or malfunction, making them an essential source of information for fault detection.

Motivation

The motivation behind this study stems from the crucial role engines play in modern transportation and industry, as well as the limitations of current diagnostic methods. Mechanical degradation over time can lead to unusual acoustic patterns that signal specific faults. However, diagnosing such faults typically requires the expertise of trained technicians, which is neither scalable nor cost-effective. Additionally, environmental noise, diverse engine configurations, and recording device variability pose significant challenges to the development of reliable, real-time diagnostic tools. This thesis seeks to address these challenges by creating an intelligent, sound-based engine fault diagnosis system that is accessible, accurate, and suitable for practical deployment.

Thesis Contribution

This thesis presents a comprehensive machine learning pipeline designed to detect and classify engine faults based on sound data collected from real-world environments. The system employs both traditional machine learning models (e.g., K-Nearest Neighbors, Random Forest, XGBoost) and deep learning approaches (e.g., Artificial Neural Networks, Convolutional Neural Networks, Long Short-Term Memory networks). Features are extracted using Mel-Frequency Cepstral Coefficients (MFCC) and spectrogram analysis to convert raw audio into structured data suitable for classification.

Furthermore, the trained models are integrated into a mobile application capable of recording, analyzing, and visualizing engine sounds in real time. This application aims to democratize engine diagnostics by making it accessible to everyday users without the need for mechanical expertise. The project also investigates the impact of classification strategies (binary vs. multi-class), quantization of models, and real-world deployment through a cloud-based Flask API connected to the Flutter application.

Thesis Organisation

The remainder of this thesis is organized as follows:

- **Chapter 1** presents a general overview of internal combustion engine faults and introduces foundational concepts in machine learning.
- **Chapter 2** provides a critical review of the literature on sound-based engine diagnostics, highlighting key methodologies, datasets, and existing limitations.
- **Chapter 3** outlines the proposed methodology, including data collection, preprocessing, feature extraction, and model design strategies.
- **Chapter 4** details the experimental evaluation of both binary and multi-class classification using ML/DL models, along with performance analysis and model optimization.
- **Chapter 5** describes the system's architecture and implementation details, including UML diagrams, backend deployment (Flask + Google Cloud), Firebase integration, and mobile application design using Flutter.

Engine Faults and AI Techniques

CHAPTER 1

ENGINE FAULTS AND AI TECHNIQUES

1.1 Introduction

Engine fault diagnosis has long been a critical aspect of vehicle and machinery maintenance. Traditional diagnostic approaches often depend on expert knowledge and manual inspection, which can be time-consuming, costly, and prone to human error. With the advancement of artificial intelligence, particularly machine learning and deep learning, new possibilities have emerged for developing automated, efficient, and accurate diagnostic systems.

This chapter introduces the fundamental concepts relevant to this study. It begins with an overview of common engine faults and the role of acoustic signals in fault detection. Then, it presents essential AI techniques used for sound classification, including feature extraction and supervised learning models. These foundational insights provide the technical background necessary to understand the methodology and system design proposed in the later chapters.

1.2 Engine Faults: Overview and Real-World Challenges

Internal combustion engines have long served as the backbone of modern transportation systems, powering everything from personal cars to heavy-duty trucks and industrial machines. Despite the growing trend toward electrification, combustion engines remain predominant, especially in regions where electric infrastructure is limited or fuel costs are subsidized. These engines are valued for their high power output, portability, and adaptability to a wide range of conditions. However, their mechanical complexity and continuous exposure to thermal and mechanical stress make them prone to faults and degradation over time [15].

At a basic level, the primary function of an internal combustion engine is to convert the chemical energy of a fuel–typically gasoline or diesel–into mechanical energy. This conversion process relies on a series of thermodynamic events governed by the four-stroke combustion cycle: intake, compression, power, and exhaust. While this cycle is fundamentally the same across most engines, the method of initiating combustion differs between gasoline (spark-ignition) and diesel (compression-ignition) engines [16].

1.2.1 Gasoline Engines and the Spark-Ignition Process

In gasoline engines, fuel is typically mixed with air before entering the combustion chamber. Once the piston compresses the air-fuel mixture, a spark plug ignites it, resulting in a rapid expansion of gases that push the piston downward.

The idealized stoichiometric combustion reaction for octane (C_8H_{18}), a common component in gasoline, is given by:

$$2\,C_8H_{18} + 25\,O_2 \longrightarrow 16\,CO_2 + 18\,H_2O \tag{1.1}$$

This equation assumes complete combustion in a perfectly oxygen-rich environment. In real conditions, however, variations in fuel quality, mixture ratio, or timing may lead to incomplete combustion, producing pollutants like carbon monoxide (CO), unburned hydrocarbons (HC), or nitrogen oxides (NO_x).

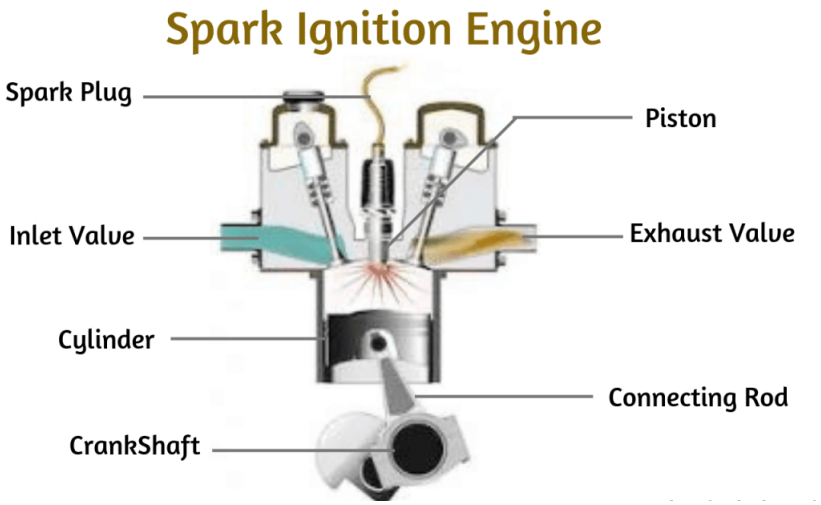

Figure 1.1: Working of Spark Ignition Engine [1]

1.2.2 Diesel Engines and Compression-Ignition

Diesel engines operate without spark plugs. Instead, they compress air to a high temperature, and fuel is injected at the end of the compression stroke. The heat from the compressed air ignites the fuel spontaneously. Diesel combustion tends to be more efficient and generate more torque, making it suitable for heavy vehicles [17].

The typical combustion reaction for diesel fuel (approximated as dodecane, $C_{12}H_{26}$) is:

$$2\,C_{12}H_{26} + 37\,O_2 \longrightarrow 24\,CO_2 + 26\,H_2O \tag{1.2}$$

However, just like in gasoline engines, deviations from ideal conditions can lead to incomplete combustion, mechanical strain, and increased emissions.

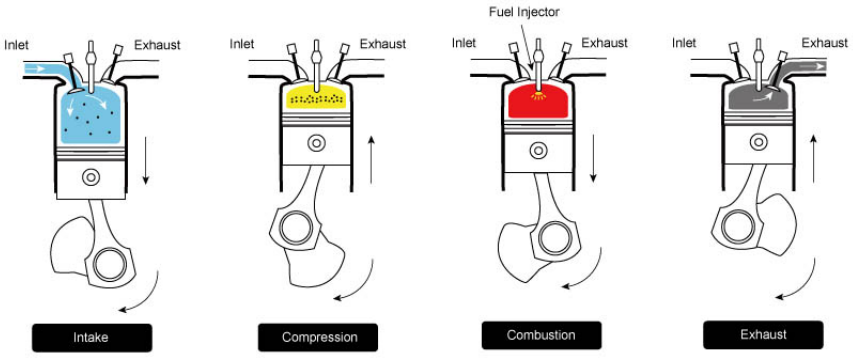

Figure 1.2: Working of Compression-Ignition Engine [2]

1.2.3 Similarities and Differences Between Gasoline and Diesel Engines

Although gasoline 1.1 and diesel 1.2 engines both operate based on the four-stroke internal combustion cycle—intake, compression, power, and exhaust—they differ significantly in terms of ignition method, fuel delivery, efficiency, and mechanical characteristics. Understanding these differences is essential for analyzing engine behavior, particularly when using audio signals for fault detection.

Similarities:

- Both convert chemical energy into mechanical energy via internal combustion.
- Both rely on pistons, cylinders, crankshafts, and valves to operate.
- The operational cycle (Otto or Diesel cycle) consists of the same four strokes.
- Both require precise air-fuel mixtures and timing for efficient operation.
- Both produce exhaust gases that must be managed via emission systems.

Differences:

These differences also explain why sound-based diagnosis is highly dependent on engine type. For example, knocking in a gasoline engine often indicates abnormal combustion (pre-ignition or detonation), while in diesel engines, knocking can be part of normal operation or linked to delayed ignition timing [18].

By recognizing both the shared framework and unique properties of these engines, intelligent diagnostic systems can be trained more effectively to identify abnormal conditions, even when similar symptoms arise from different root causes.

Feature	**Gasoline Engine (Spark Ignition)**	**Diesel Engine (Compression Ignition)**
Ignition Method	Uses a spark plug	Uses heat from compressed air
Fuel Delivery	Pre-mixed air and fuel	Air compressed first, fuel injected directly
Compression Ratio	Lower (8:1 to 12:1)	Higher (14:1 to 25:1)
Efficiency	Lower thermal efficiency	Higher thermal efficiency
Sound Characteristics	Smoother and quieter	Louder with characteristic knocking
Fuel Type	Lighter fuels like gasoline	Heavier fuels like diesel
Emissions	Higher CO and HC	Higher NO_x and particulates
Maintenance Needs	More frequent tune-ups	More robust but costlier repairs

Table 1.1: Comparison Between Gasoline and Diesel Engines

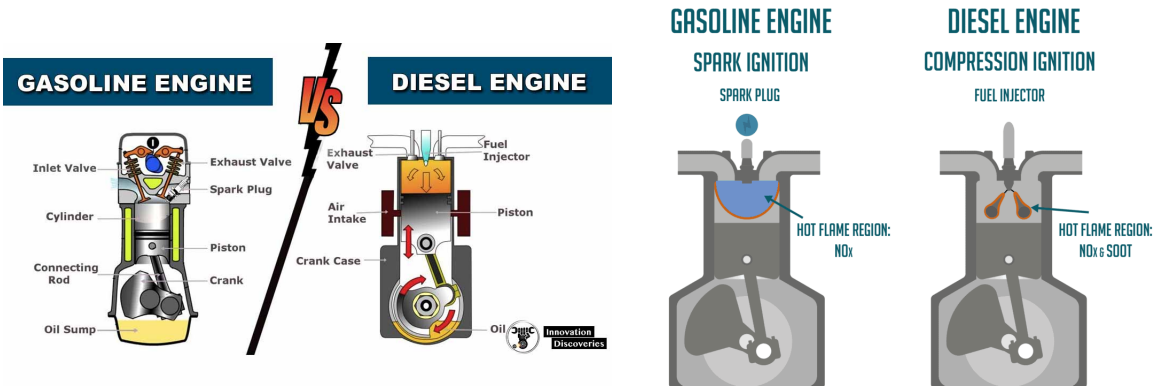

Figure 1.3: Comparison between the two engines [3] [4]

1.2.4 Importance of Early Fault Detection

Early fault detection plays a crucial role in maintaining engine reliability, reducing maintenance costs, and ensuring passenger safety. When engine issues are identified and addressed at an early stage, it prevents the escalation of minor faults into major mechanical failures that may result in vehicle breakdown, costly repairs, or even accidents. Moreover, early detection contributes to reducing fuel consumption and emissions by preserving optimal combustion conditions.

From an economic perspective, proactive maintenance based on early fault identification

minimizes unplanned downtime and extends the engine's lifespan. In industrial applications or commercial fleets, this translates into significant savings and increased operational efficiency. In personal vehicles, it offers peace of mind to drivers and ensures the availability of the vehicle when needed.

1.2.5 The Role of Sound in Fault Detection

Sound is one of the most immediate and accessible indicators of an engine's condition. Unlike visual inspections or sensor-based diagnostics, which often require specialized tools or data interpretation, sound anomalies can be detected intuitively—even by untrained drivers. Changes in engine sound may indicate abnormal combustion, component misalignment, wear, or mechanical interference.

Each fault type produces a distinct acoustic signature. For example:

- Tapping or knocking noises may point to improper ignition timing or worn bearings.
- Hissing or whistling could indicate air leaks in the intake system.
- Grinding or squealing often signals issues with belts or rotating components.

Traditional fault detection systems rely on physical sensors (vibration, pressure, or temperature) to infer abnormal conditions [19]. However, acoustic-based diagnostics offer a non-intrusive, real-time, and cost-effective alternative. This approach becomes especially valuable in the context of vehicles where access to internal components is limited.

1.2.6 From Sound to Smart Diagnostics: A Transition to Machine Learning

While human intuition can detect obvious sound anomalies, it falls short when analyzing subtle or overlapping signals. Moreover, subjective interpretation varies from person to person, limiting the accuracy and consistency of manual diagnoses. This is where machine learning (ML) becomes a powerful ally.

By analyzing audio data captured from engines—typically using microphones and processed into features such as Mel-Frequency Cepstral Coefficients (MFCC) or spectrograms—machine learning algorithms can be trained to detect and classify faults with high precision. These systems learn from labeled examples of faulty and healthy engine sounds, enabling them to recognize complex patterns that would be imperceptible to the human ear.

This intelligent approach allows for:

- Automated real-time fault classification
- Scalability across vehicle types
- Continuous learning and adaptation
- Integration into mobile or embedded diagnostic tools

In the next sections, we will explore these algorithms in depth, assess related work, and present our implementation of an ML-powered system to detect and classify engine faults based on audio inputs.

1.3 Artificial Intelligence

This section provides a comprehensive overview of artificial intelligence (AI) techniques applied to fault detection systems, with a particular focus on machine learning (ML) and deep learning (DL) models. These technologies enable the automatic identification of engine anomalies by learning from data—especially sound signals—without relying on manually programmed rules.

1.3.1 Machine Learning Fundamentals

Definition

Machine Learning (ML) is a branch of artificial intelligence (AI) that enables computer systems to learn and make decisions from data without being explicitly programmed. Rather than following fixed rules coded by humans, ML systems identify patterns and relationships within large datasets and use them to make predictions or classifications. This capacity to adapt to data makes ML particularly valuable for complex tasks such as sound-based engine fault detection, where patterns may not be obvious or easily definable through manual rules.

Main Types of Machine Learning Tasks

ML techniques are generally divided into three main categories:

- **Supervised Learning:** In this paradigm, models are trained using labeled data, where the correct output is known. The model learns to map inputs to outputs. It is especially

suitable for classification tasks like determining whether an engine sound is normal or faulty.

- **Unsupervised Learning:** Here, the model tries to find structure in unlabeled data, such as grouping similar engine sounds without prior knowledge of categories.
- **Reinforcement Learning:** This approach involves learning through interaction with an environment, receiving rewards or penalties based on actions. Though powerful, it is less commonly applied in sound classification contexts.

Key Algorithms Used in This Work

In this research, we employ several supervised ML algorithms, each with unique characteristics:

- **Logistic Regression (LR):** A statistical method used for binary classification. Despite its simplicity, it often provides a strong baseline.
- **K-Nearest Neighbors (K-NN):** A non-parametric method that classifies samples based on the most common label among their k nearest neighbors in the feature space.
- **Random Forest (RF):** An ensemble learning technique based on decision trees, where multiple trees vote to determine the final output. It is robust to noise and overfitting.
- **Support Vector Machine (SVM):** A powerful classifier that aims to find the optimal hyperplane that separates different classes with maximum margin.
- **XGBoost:** A scalable, high-performance implementation of gradient boosting that excels in structured data classification tasks.

These algorithms are applied to numerical features extracted from audio data, specifically Mel-Frequency Cepstral Coefficients (MFCC), to differentiate between healthy and faulty engine sounds.

1.3.2 Supervised Learning Algorithms Overview

Supervised learning is the most commonly used paradigm in machine learning, especially in classification problems like fault detection. In supervised learning, each input sample is associated with a known output label, and the algorithm learns to map input features to the

correct output. Below, we provide a detailed explanation of the main supervised learning algorithms used in this research.

K-Nearest Neighbors (K-NN)

K-NN is a simple yet effective instance-based learning algorithm. It does not create an internal model; instead, it classifies new samples based on the majority label among the k closest training samples, using a distance metric (usually Euclidean distance).

Distance formula (Euclidean):

$$d(x, x') = \sqrt{\sum_{i=1}^{n}(x_i - x_i')^2}$$

Where x and x' are two feature vectors.

Pros: Easy to implement, interpretable.

Cons: Slow on large datasets, sensitive to irrelevant features or noisy data.

Figure 1.4 shows an example of three clusters .

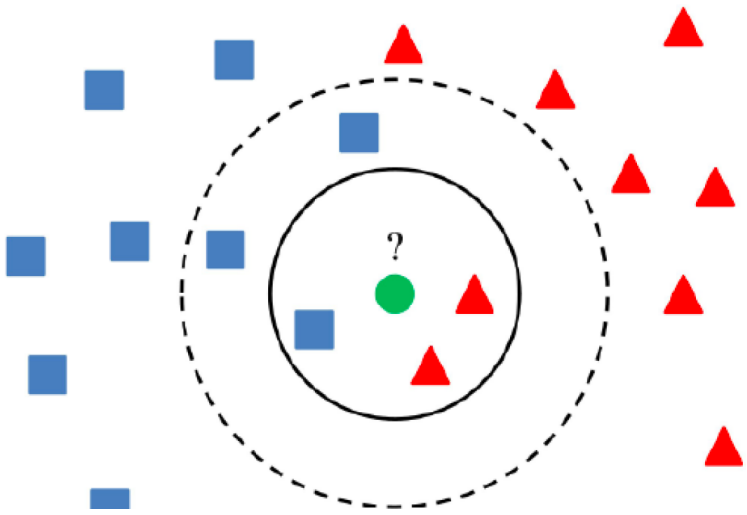

Figure 1.4: KNN [5]

Random Forest (RF)

Random Forest is an ensemble learning method based on decision trees. It builds multiple decision trees during training and outputs the class that is the mode of the classes predicted by individual trees.

Each tree is trained on a random subset of the data and features (bootstrap aggregating or "bagging"), which improves generalization and reduces overfitting.

Decision Tree Splitting Criterion (Gini Index):

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Where p_i is the probability of a class in a node.

Pros: Handles high-dimensional data, robust to noise.

Cons: Can be less interpretable, larger in memory.

Figure 1.5 A simple diagram of how SVM works.

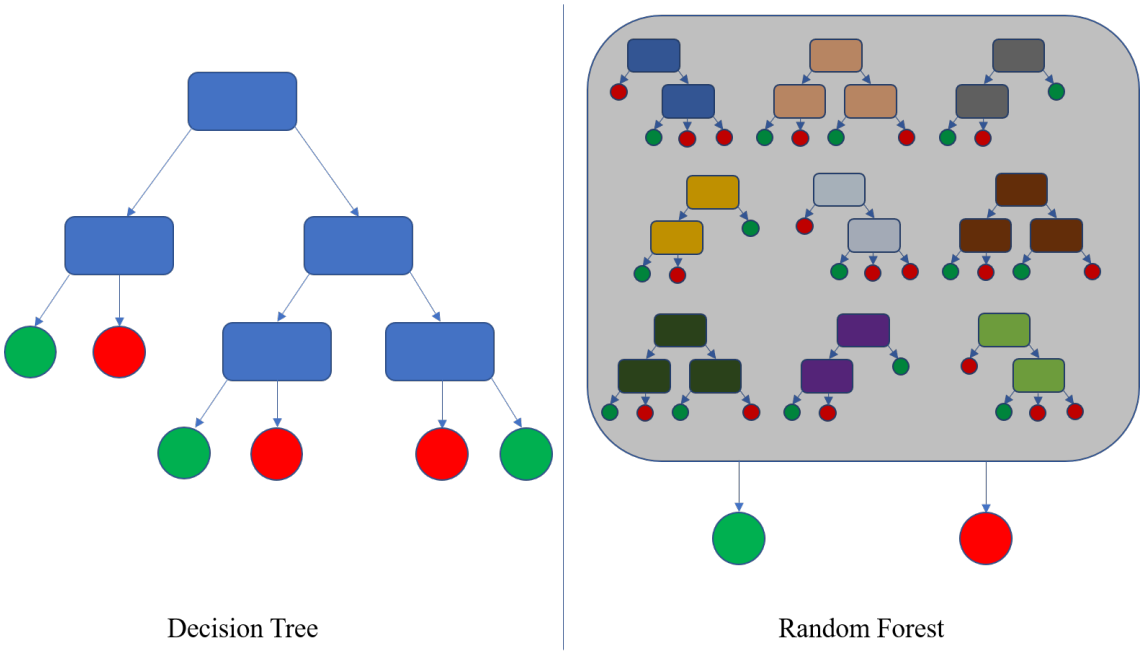

Figure 1.5: Random Forest [6]

XGBoost (Extreme Gradient Boosting)

XGBoost is a powerful gradient boosting algorithm that builds additive models in a forward stage-wise fashion. It minimizes a regularized loss function to prevent overfitting and uses tree ensembles.

Objective Function in XGBoost:

$$Obj(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Where:

- l is the loss function (e.g., log-loss),
- Ω is the regularization term,

- f_k is an individual regression tree.

Pros: Extremely fast and accurate, handles missing values well.

Cons: Complex to tune, computationally intensive.

Figure 1.6 is a simple example and illustration of how XGBoost works.

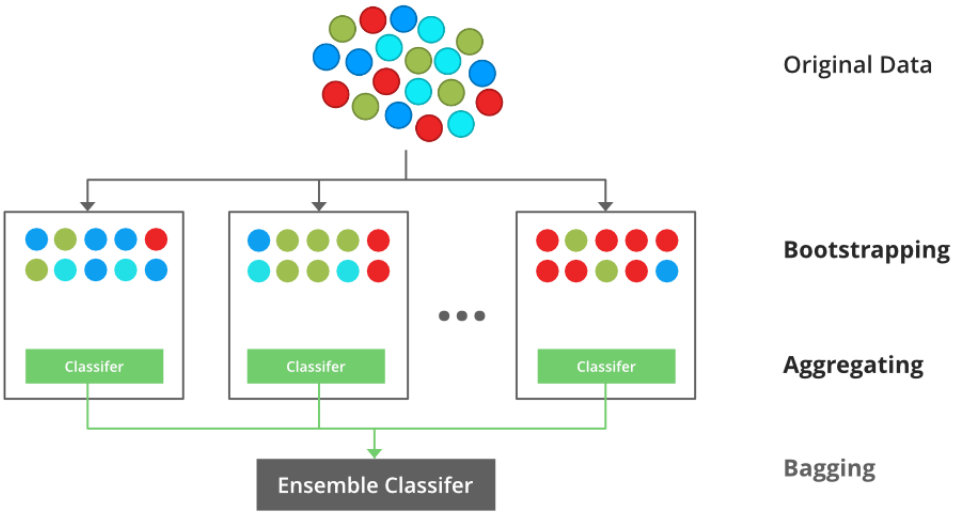

Figure 1.6: XGBoos [7]

Logistic Regression (LR)

Despite its name, logistic regression is a classification algorithm. It models the probability of a binary outcome using the logistic (sigmoid) function.

Sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad \text{where } z = w^T x + b$$

with w as the weights and b as the bias.

Pros: Interpretable, works well when the relationship between features and output is linear.

Cons: Limited in handling complex relationships.

Figure 1.7 is an illustration of how LR works

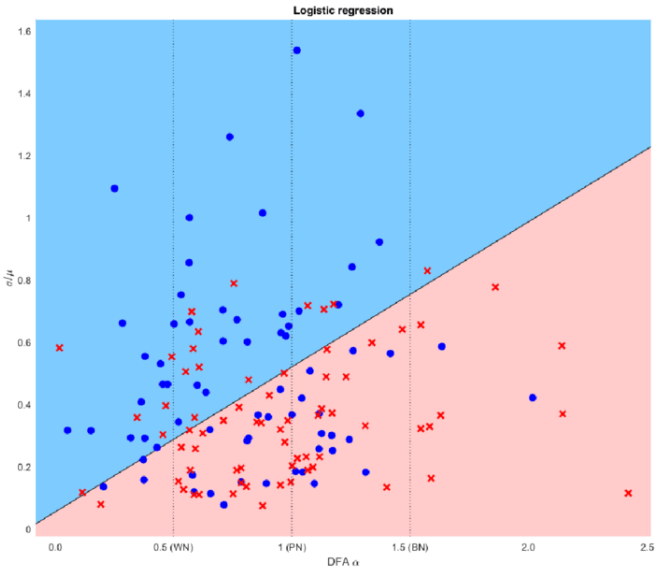

Figure 1.7: Logistic Regression [8]

Support Vector Machine (SVM)

SVM tries to find the optimal hyperplane that separates the data into classes with the largest possible margin.

Optimization Objective:

$$\min \frac{1}{2}\|w\|^2 \quad \text{subject to } y_i(w^T x_i + b) \geq 1$$

Pros: Effective in high-dimensional spaces, works well with clear margins.

Cons: Not efficient for large datasets, sensitive to choice of kernel and parameters.

Figure 1.8 is an illustration of how SVM works

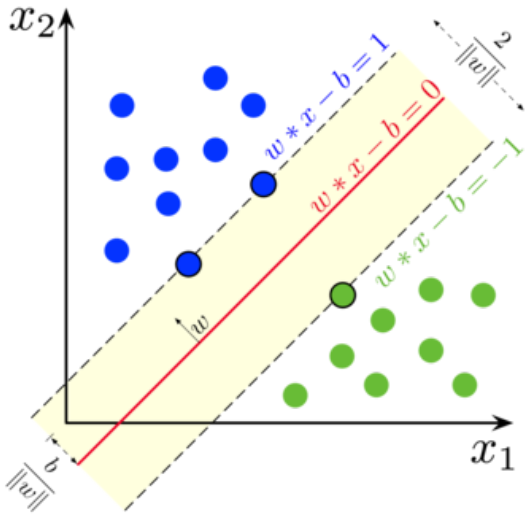

Figure 1.8: Support Vector Machine [9]

1.3.3 Deep Learning Approaches

Deep learning (DL) is a subfield of machine learning that employs multi-layered neural networks to automatically extract and learn complex representations from data. These models are particularly powerful for tasks involving unstructured data such as images, audio signals, and time-series sequences — making them highly suitable for engine fault detection through sound analysis.

Definition of Deep Learning

Deep learning models are inspired by the structure and function of the human brain. They consist of artificial neurons organized into layers: an input layer, one or more hidden layers, and an output layer. These networks are capable of hierarchical feature learning, allowing them to extract low-level and high-level abstractions directly from raw data, such as spectrograms or MFCCs.

Deep Learning Models Used in This Work

In this research, we implement three widely used deep learning architectures, each designed to handle a specific data structure.

Artificial Neural Network (ANN)

ANNs are composed of fully connected layers in which each neuron receives input from all neurons in the previous layer. These models are well-suited for tabular or numerical data such as MFCC features.

Forward propagation in a single neuron:

$$a = \sigma(w^T x + b)$$

Where:

- x is the input vector,
- w is the weight vector,
- b is the bias,
- σ is the activation function (e.g., ReLU or sigmoid).

Use case in our work: Classification of engine condition using MFCC features.

Advantages: Simple to implement and train.

Disadvantages: Limited in capturing spatial or temporal features.

Figure 1.9 shows how ANN works.

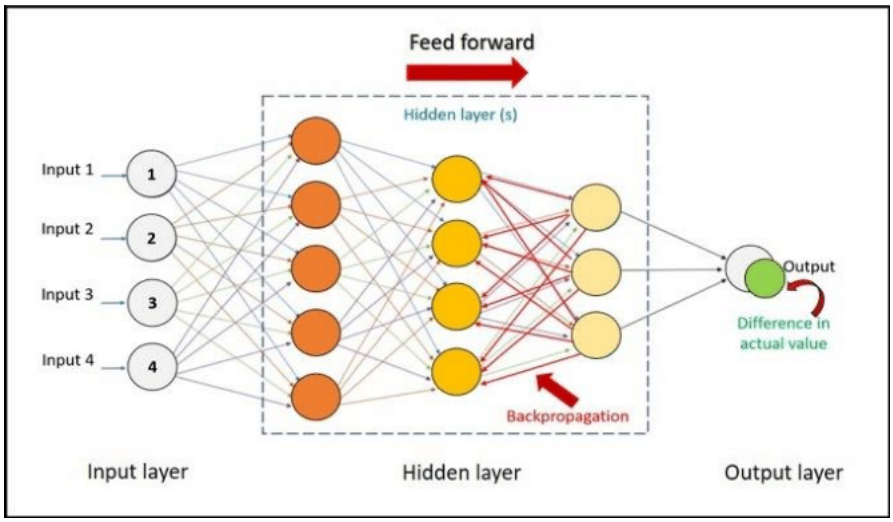

Figure 1.9: ANN [10]

Convolutional Neural Network (CNN)

CNNs are specialized in processing data with spatial hierarchies, such as images or 2D audio representations (spectrograms). They use convolutional layers to automatically learn local patterns.

2D Convolution operation:

$$S(i,j) = (X * K)(i,j) = \sum_m \sum_n X(i+m, j+n) \cdot K(m,n)$$

Where:

- X is the input image (e.g., spectrogram),
- K is the filter (kernel),
- S is the resulting feature map.

Use case in our work: Classification of engine faults from spectrogram images.

Advantages: Excellent at capturing spatial features in time-frequency audio data.

Disadvantages: Requires more computational power and training data.

Figure 1.10 shows the stages through which audio is converted to a spectrogram and then to the final classification.

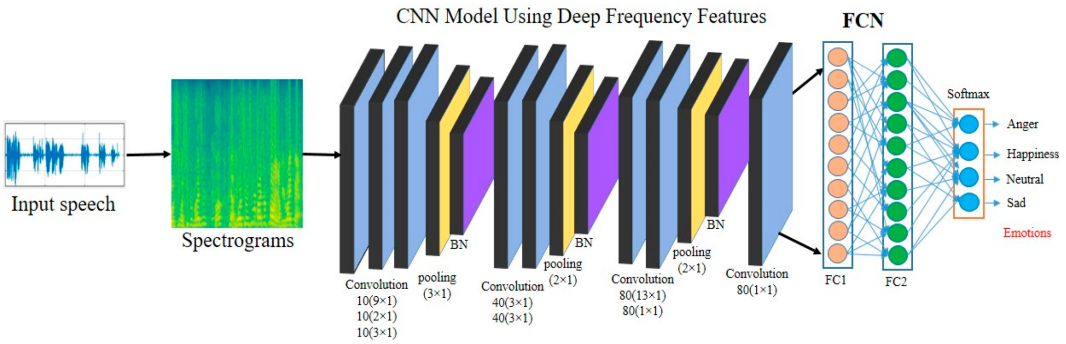

Figure 1.10: CNN [11]

Long Short-Term Memory (LSTM)

LSTM networks are a type of recurrent neural network (RNN) designed to handle sequential or time-dependent data. They are particularly useful for analyzing the temporal evolution of audio signals.

LSTM Cell Equations:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c x_t + U_c h_{t-1} + b_c)$$
$$h_t = o_t \odot \tanh(c_t)$$

Where:

- f_t, i_t, o_t are forget, input, and output gates respectively,
- c_t is the cell state, h_t is the hidden state,
- x_t is the input at time t,
- $\odot$ denotes element-wise multiplication.

Use case in our work: Sequential modeling of MFCC vectors from engine audio.

Advantages: Captures long-term dependencies in time-series data.

Disadvantages: Training is slower and requires careful tuning.

Figure 1.11: A simple diagram showing how LSTM works.

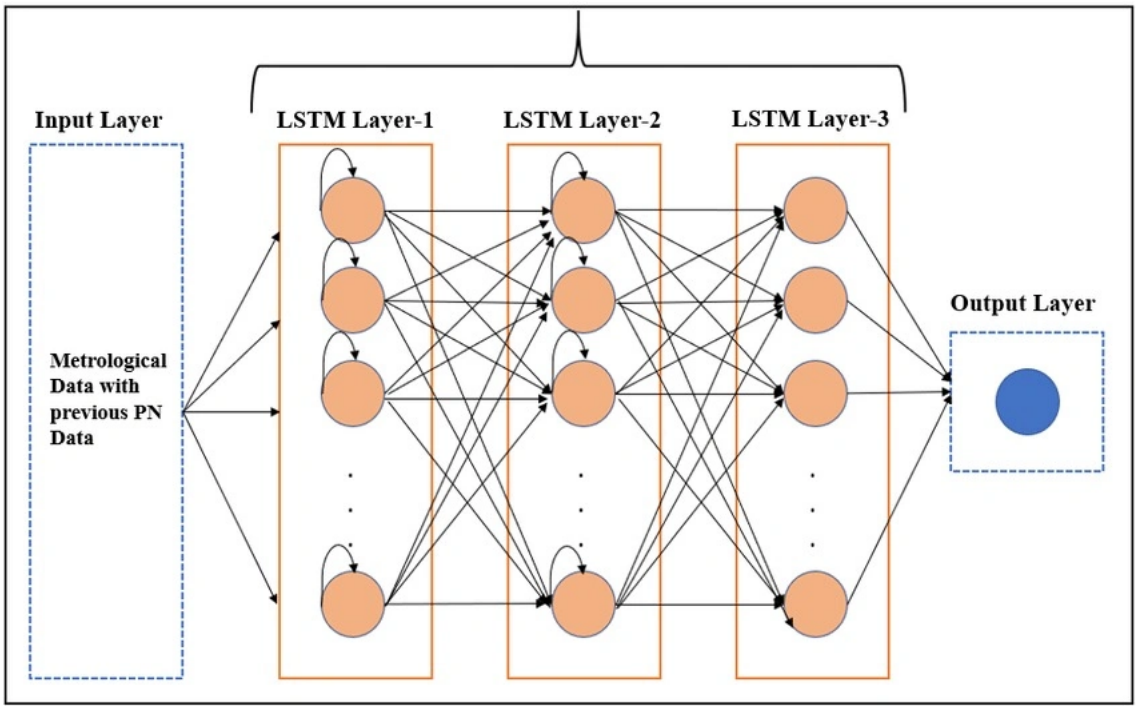

Figure 1.11: LSTM [12]

1.3.4 ML Models Evaluation Metrics

To ensure that a machine learning model performs well and generalizes beyond the training data, it must be evaluated using reliable and informative performance metrics. These metrics provide insight into how accurately the model classifies new, unseen data — especially important in critical applications such as engine fault detection, where incorrect classification may result in costly or dangerous consequences.

Below are the key metrics used in this research:

Accuracy

Accuracy measures the overall correctness of the model and is defined as the ratio of correctly predicted observations to the total observations:

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

Where:

- TP: True Positives (faulty engines correctly predicted),
- TN: True Negatives (healthy engines correctly predicted),
- FP: False Positives (healthy engines incorrectly classified as faulty),

- FN: False Negatives (faulty engines incorrectly classified as healthy).

Limitation: Accuracy may be misleading if the dataset is imbalanced.

Precision

Precision indicates how many of the predicted positive cases were actually positive. It is especially important when the cost of false alarms is high.

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

High precision means fewer false positives.

Recall (Sensitivity or True Positive Rate)

Recall measures how well the model identifies all relevant instances, i.e., the proportion of actual positive cases that were correctly identified.

$$\text{Recall} = \frac{TP}{TP + FN}$$

High recall is critical in applications where missing a faulty engine is dangerous.

F1-Score

The F1-score is the harmonic mean of precision and recall, offering a balance between them. It is especially useful when the dataset is imbalanced.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Confusion Matrix

A confusion matrix is a tabular visualization of model predictions. It provides a complete picture of correct and incorrect classifications across all classes.

	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

It helps to identify which types of errors are most frequent.

Receiver Operating Characteristic (ROC) Curve and AUC

The ROC curve plots the true positive rate (Recall) against the false positive rate. The AUC (Area Under the Curve) summarizes the model's ability to discriminate between classes:

- AUC = 1: Perfect classifier,
- AUC = 0.5: No discrimination (random guess).

Each of these metrics contributes to a complete and nuanced understanding of model performance. In this research, we apply all of them to evaluate both traditional ML and deep learning models, ensuring a fair and comprehensive comparison.

1.3.5 Comparison Between Traditional Machine Learning and Deep Learning

Machine Learning (ML) and Deep Learning (DL) are two major approaches in artificial intelligence, each offering different advantages and facing distinct limitations depending on the application. In the context of engine fault detection through sound, understanding the trade-offs between these methods is critical for selecting the right model.

A. Key Differences

Aspect	**Traditional ML**	**Deep Learning**
Feature Engineering	Requires manual feature extraction (e.g., MFCCs, statistical features)	Automatically extracts features (e.g., from raw spectrograms)
Data Requirement	Works well with small to medium datasets	Requires large datasets for optimal performance
Computation	Lightweight, runs faster	Computationally intensive, requires GPU for training
Model Complexity	Simpler models (e.g., K-NN, RF)	Complex multi-layered architectures (e.g., CNN, LSTM)
Interpretability	Easier to interpret (e.g., decision trees)	Considered a black box, less interpretable
Training Time	Generally short	Longer training times due to more parameters
Performance	Good with engineered features	Potentially better with large data and rich patterns

Table 1.2: Comparison between Traditional Machine Learning and Deep Learning

B. Implementation Challenges

Data Preprocessing and Labeling:

ML models require carefully selected features (like MFCCs), while DL models demand raw or lightly processed inputs, often making labeling and preprocessing more time-consuming.

Overfitting Risk:

DL models, with their large number of parameters, are more prone to overfitting, especially on small datasets. Regularization techniques such as Dropout or Early Stopping are necessary.

Hardware Requirements:

While ML models can run efficiently on CPUs, DL architectures (e.g., CNN or LSTM) benefit significantly from GPU acceleration.

Hyperparameter Tuning:

Both ML and DL models require tuning (e.g., learning rate, number of trees, layers), but DL often involves more parameters, making tuning more complex.

Model Deployment:

ML models are usually lightweight and easier to deploy on edge devices. DL models may require optimization (e.g., quantization, pruning) before deployment on resource-constrained systems like mobile phones or microcontrollers.

C. Models Trained in This Work

We applied both ML and DL models to compare their effectiveness on engine fault classification from sound features:

- **Machine Learning Models:**
  - K-Nearest Neighbors (K-NN)
  - Random Forest
  - XGBoost
  - Logistic Regression
  - Support Vector Machine (SVM)
- **Deep Learning Models:**
  - Artificial Neural Network (ANN)
  - Convolutional Neural Network (CNN)
  - Long Short-Term Memory (LSTM)

These models were evaluated based on the metrics discussed earlier (accuracy, precision, recall, F1-score), and their performances were compared to determine the best approach for our use case.

1.4 Conclusion

Traditional ML remains a strong choice for structured data and limited datasets, offering fast training and interpretability. However, DL significantly outperforms in learning complex patterns from rich data like audio spectrograms. Therefore, selecting the optimal model depends on the nature of the input data, available computational resources, and deployment constraints.

Literature Review on Sound-Based Engine Fault Detection Using AI

CHAPTER 2

LITERATURE REVIEW ON SOUND-BASED ENGINE FAULT DETECTION USING AI

2.1 Introduction

In recent years, automotive fault diagnosis has evolved significantly with the integration of artificial intelligence (AI) techniques. Traditional methods, based on manual inspection or threshold-based sensors, are increasingly being replaced by data-driven approaches that leverage machine learning (ML) and deep learning (DL). Among these, sound-based diagnostics have gained attention due to their non-invasive, low-cost nature and the valuable information embedded in engine acoustics.

This chapter reviews key research related to AI-based engine fault detection using sound signals. We analyze methodologies, datasets, feature extraction techniques, and models used in previous studies to identify current trends and gaps. Particular attention is given to two representative works—one using traditional ML for early fault detection, and another applying deep learning for detailed fault classification. These studies are compared with our approach to highlight its novelty and relevance within the broader research context.

2.2 Overview of Existing Research on Engine Fault Detection

2.2.1 Study 1: Intelligent Sound-Based Early Fault Detection System for Vehicles

This study presents a machine learning-based framework designed for the identification and classification of engine faults using acoustic signals recorded from vehicles. The motivation stems from the growing demand for intelligent, non-invasive, and scalable diagnostic systems that are adaptable to real-world automotive conditions.

Sr#	Sounds Type	Number of Sounds
1	Bad Exhaust	40
2	Brake Pad	25
3	Car Stopping Metal to Metal	13
4	Engine Running Without Oil	18
5	Engine Seized	14
6	Failing Water Pump	28
7	Hole in Muffler	37
8	Loose Heat Shield	17
9	Piston Slapping	18
10	Radiator Boiling	18
11	Struts	20
12	Transmission Slipping	28
13	Unworn Serpentine Belt	19
14	Vacuum Hose Leak	28
15	Valves Tapping	28

Table 2.1: Training dataset containing 15 types of problems [13]

Dataset and Feature Extraction

A total of 565 audio recordings were collected, consisting of 351 faulty engine samples and 214 healthy engine samples. The data were sourced from various automotive repair workshops, ensuring a realistic representation of typical engine fault scenarios. To prepare the data for classification, the following feature extraction techniques were employed across different domains:

- **Short-Time Fourier Transform (STFT):** Utilized for time-frequency domain analysis to capture dynamic spectral characteristics.
- **Mel-Frequency Cepstral Coefficients (MFCC):** Applied for perceptual modeling of audio features, reflecting the way humans perceive sound.
- **Time-domain Features:** Extracted to represent statistical and temporal characteristics of the raw audio signals.

In addition, noise reduction techniques and segmentation into audio frames were performed to ensure temporal consistency and enhance feature reliability.

Classification Models and Results

Several supervised machine learning models were evaluated for both binary and multi-class classification tasks:

- Support Vector Machine (SVM)
- K-Nearest Neighbors (K-NN)
- J48 Decision Tree
- Random Tree
- Random Forest

Figure 2.1 shows that random forest performs the highest results on time and frequency domain features. So, we applied random forest on time-frequency domain features, and it produced the highest true positive (TP) results. It correctly classifies 92.74% of instances [13].

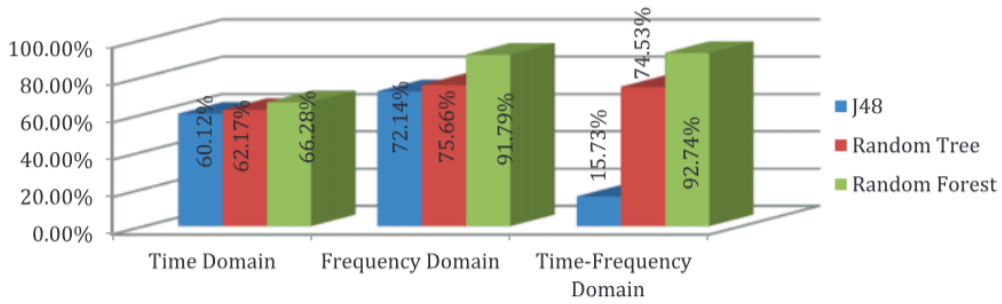

Figure 2.1: Comparison of different classifiers [13]

Initially, binary classification was carried out to distinguish between healthy and faulty engine sounds. This was followed by multi-class classification, where the system attempted to categorize faulty sounds into one of 15 specific fault types.

Among the tested models, the **Random Forest** classifier achieved the highest accuracy of **66%** in fault classification based solely on time-domain features. The authors suggested that this performance could be further improved by incorporating data augmentation techniques and expanding the feature space.

In Figure 2.2 we show the performance of the random forest on all three sets of features and in the context of different performance measures. Random Forest classifier outperforms in both Frequency and Time-Frequency domains as compared to Time Domain [13].

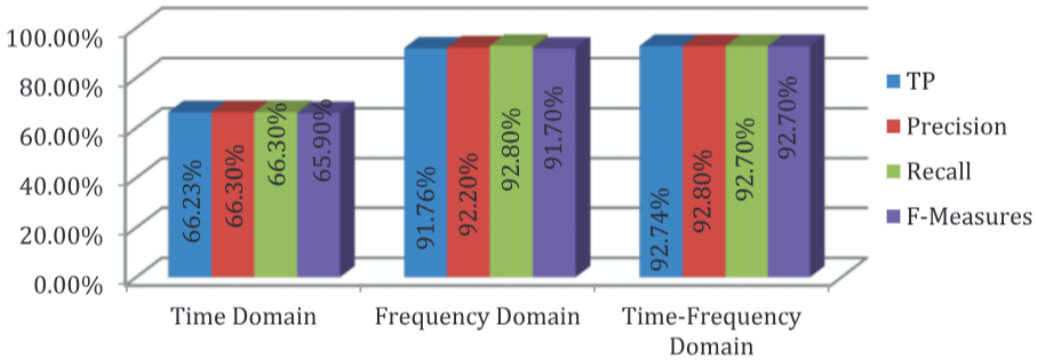

Figure 2.2: (Classification) Random forest different performance measures [13]

In Figure 2.3 a detailed overview of the accuracy performance of the Random Forest Classifier on one of the 15 problems identified earlier is shown. As we focus on the fifteen (15) different problems. The problems are not only about the engine of the vehicle, but some noisy sounds originating from the other parts of the vehicle [13].

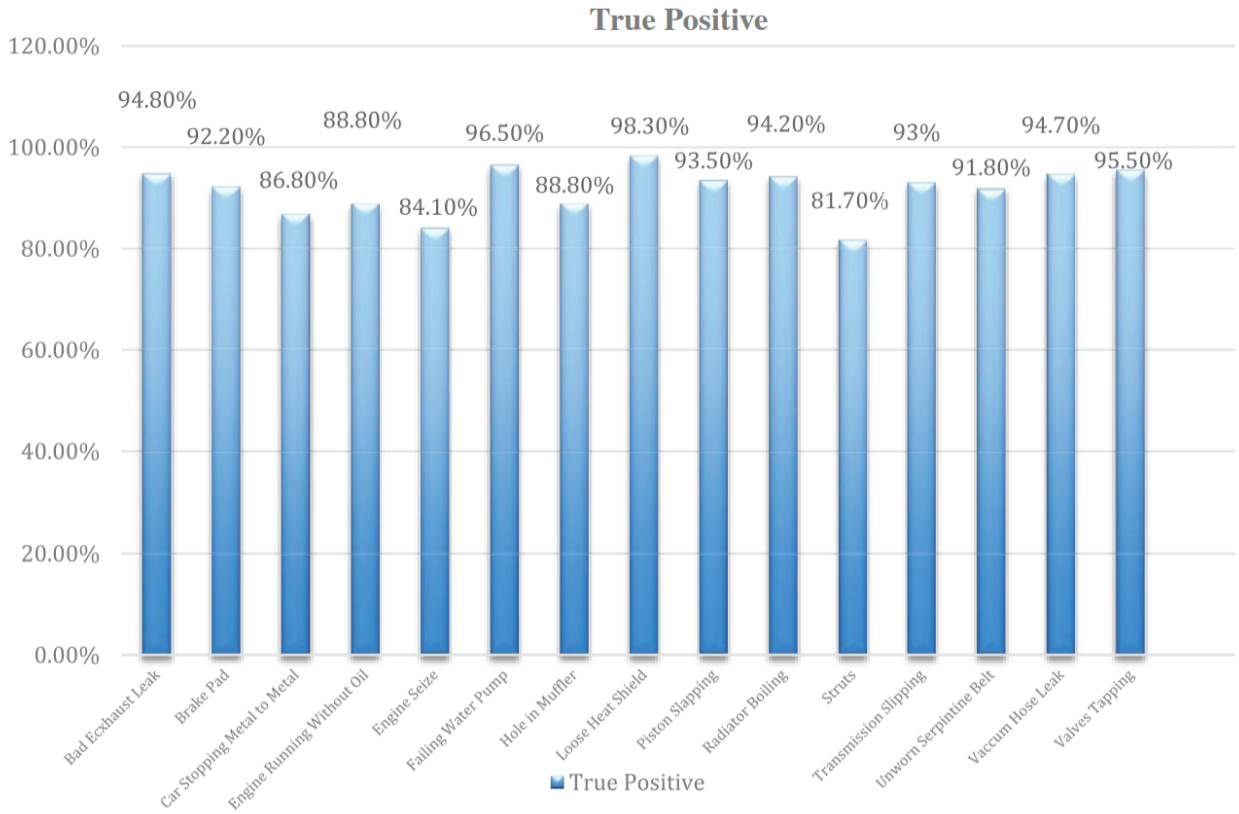

Figure 2.3: (Random Forest) Performance measures of different classes [13]

Key Contributions and Limitations

This study demonstrated the feasibility of using sound-based diagnostics for early fault detection in vehicles. However, it also highlighted several limitations:

- Moderate accuracy in multi-class classification due to reliance on time-domain features.
- Limited dataset size, potentially constraining generalizability.

Relevance to the Present Work

The findings of this study are closely aligned with the objectives of this thesis, as both investigations:

- Focus on audio-based detection of vehicle engine faults.
- Utilize real-world acoustic data collected from automotive environments.
- Include **Random Forest** as one of the main classification algorithms.

However, the present research extends the prior work by integrating a richer set of features, including MFCCs and spectrograms, and exploring both traditional machine learning and deep learning models such as **ANN, CNN, and LSTM**. This more comprehensive approach aims to improve classification accuracy and robustness across diverse engine fault types.

2.2.2 Study 2: Fine-Grained Engine Fault Sound Event Detection Using Multimodal Signals

This study proposes a fine-grained fault detection framework for automotive engines by integrating multimodal signal processing, particularly audio and vibration data. The main objective is to improve the precision of fault classification by exploiting complementary information from different physical domains [14].

Dataset and Multimodal Signal Acquisition.

The dataset used in this study comprises synchronized audio and vibration recordings collected from a diverse set of vehicles. Each sample is labeled according to one of ten predefined engine fault categories. Audio signals capture the acoustic emissions of the engine, while vibration signals provide structural feedback that may not be evident in the audio domain. This multimodal acquisition aims to address the limitations of unimodal diagnostic systems.

Class	Avg. Sound Duration (s)	# Events Train	Valid.	Test
Engine Knock	9.07	531	97	117
Belt Squeal	12.00	93	19	21
Exhaust Noise	9.66	460	86	102
Unstable Idle	4.40	130	24	29
Internal Tick	10.48	211	39	46
Ambiguous Tick	9.58	381	70	83
Accessory Noise	6.82	735	136	155
Engine Rattle	5.25	807	145	168
Startup Rattle	1.21	199	37	42
Trouble Starting	2.09	234	42	49

Table 2.2: Engine fault sound event detection dataset overview [14].

Feature Extraction and Processing Techniques.

To convert raw signals into machine-readable features, the authors applied:

- **Spectrogram analysis** to visualize the time-frequency characteristics of the sound signals.
- **Temporal analysis techniques** to capture sequential patterns in both audio and vibration signals.
- **Preprocessing steps** such as normalization and temporal alignment between modalities to ensure consistency across the dataset.

The features extracted from both modalities were fused into a unified multimodal feature vector for classification.

Model Architecture and Performance.

A Convolutional Neural Network (CNN) architecture was developed to handle multimodal inputs. The network consists of parallel branches that process audio and vibration spectrograms separately. These branches are later fused to form a joint feature representation. This design significantly improved classification accuracy compared to unimodal audio-based models.

Experimental results confirmed that multimodal fusion increased robustness and detection performance, particularly in noisy or ambiguous conditions.

Figure 2.4 shows : a) Proposed engine fault sound event detection architecture using audio and vibration engine recordings. b) Proposed pretraining strategy using a large-scale weakly-labeled engine fault dataset with our supervised contrastive loss. Best viewed with zoom and color [14].

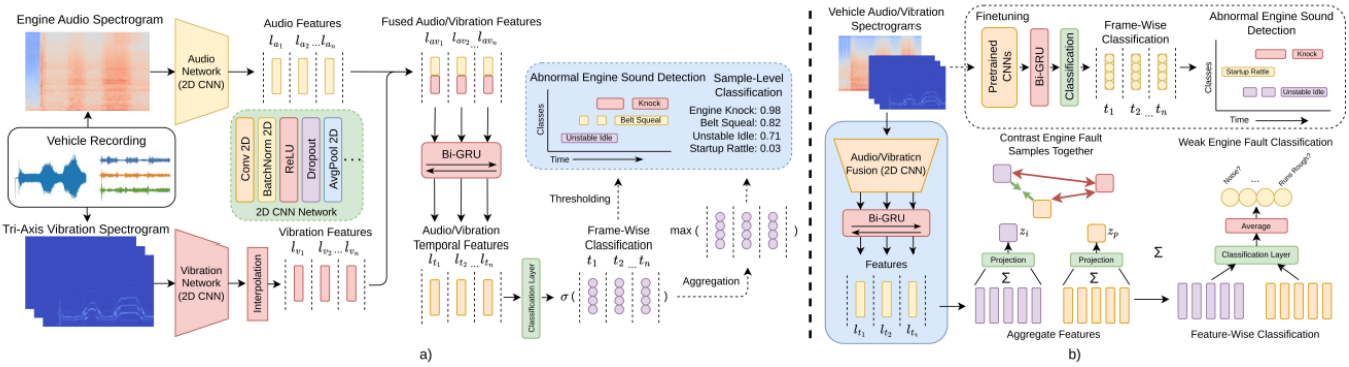

Figure 2.4: CNN Architecture for Comprehensive Study [14]

Key Contributions and Limitations.

This research underscores the value of sensor fusion in fine-grained engine fault detection. The integration of audio and vibration data enables more accurate differentiation between faults with overlapping sound profiles. However, the system's dependency on dual-sensor input may limit its deployment in low-cost or resource-constrained settings where vibration sensors are unavailable.

Relevance to This Work.

The current thesis shares several conceptual parallels with this study:

- Both adopt deep learning approaches and leverage time-frequency representations such as spectrograms.
- This study validates the strength of CNNs in sound-based fault classification, supporting our use of CNNs in a similar context.
- Although this thesis focuses solely on audio data, the multimodal framework discussed here provides a valuable benchmark and future direction for expanding diagnostic capabilities.

In summary, this study demonstrates how combining multiple signal sources can enhance diagnostic accuracy—an insight that can guide future enhancements to single-modality systems like the one developed in this thesis.

2.3 Comparative Analysis

A brief comparison is made between two recent studies and the current work, focusing on data type, fault categories, and applied models to highlight methodological relevance.

Criterion	**Study 1 2.2.1 [13]**	**Study 2 2.2.2 [14]**	**This Work**
Data Type	Audio recordings from real workshops	Audio + vibration signals from various vehicles	Audio recordings from real workshop conditions
Number of Fault Types	15 different engine faults	10 specific fault categories	5 fault types (including healthy samples)
Feature Extraction	MFCC, STFT, denoising, time segmentation	Spectrograms, temporal and vibration analysis	MFCC and Spectrogram
Models Used	SVM, K-NN, Random Forest	CNN with multimodal fusion	K-NN, Random Forest, XGBoost, SVM, Logistic Regression; ANN, CNN, LSTM
Accuracy and Evaluation	Random Forest achieved up to 66% accuracy using time-domain features	Higher accuracy using CNN with fused audio-vibration data	Varies by model; evaluated with multiple metrics (accuracy, F1-score, etc.)
Similarity to This Work	High similarity in using real-world sound data and preprocessing techniques	High relevance in using CNN and spectrograms; potential for future expansion to multimodal fusion	Direct application to real audio signals using a wide range of ML and DL techniques

Table 2.3: Comparative analysis of related studies and the present research.

Key Observations:

Both studies share strong methodological similarities with the current work, particularly in the preprocessing and classification phases:

- **Study 1** is particularly relevant due to its use of workshop-recorded audio data, aligning closely with this work's real-world conditions. The application of MFCC and STFT, along with traditional classifiers such as SVM and Random Forest, serves as a solid baseline for comparison.
- **Study 2** advances the field by incorporating multimodal sensor data (audio and vibration), and leveraging CNN architectures for improved granularity and robustness. Although not directly implemented in this work, its fusion-based framework offers valuable insights for future enhancements.
- **This study** extends the literature by applying both traditional and deep learning models on a curated dataset, enabling a comprehensive performance comparison across different modeling strategies. The diversity of models and metrics used enriches the analysis of trade-offs in terms of complexity, performance, and resource needs.

2.4 Conclusion

This chapter reviewed key literature on sound-based engine fault detection, emphasizing two recent studies. The first used traditional ML methods like Random Forest and SVM with handcrafted features (MFCC, STFT), while the second employed deep learning (CNNs) with multimodal data. These works highlight the potential of both approaches in noisy, real-world scenarios.

Building on these foundations, the present study combines real workshop recordings with both ML and DL models, using MFCC and spectrogram features for robust classification. The findings from this review support the chosen methodology and offer directions for future enhancements, such as multimodal integration and improved feature design.

The next chapter presents the full methodology, covering data preparation, feature extraction.

Dataset Preparation and Feature Engineering Techniques

CHAPTER 3

DATASET PREPARATION AND FEATURE ENGINEERING TECHNIQUES

3.1 Introduction

The effectiveness of any machine learning model relies heavily on the quality and structure of the data used during training. In the context of engine fault detection through sound, proper dataset preparation and feature engineering play a pivotal role in capturing the acoustic characteristics that distinguish healthy engines from faulty ones. Unlike traditional diagnostic approaches, sound-based analysis provides a non-invasive, cost-effective, and real-time means of fault identification by exploiting the unique auditory patterns associated with mechanical issues.

This chapter details the entire data preparation pipeline adopted in this study. It includes the systematic collection of audio recordings under controlled conditions, the organization of samples into binary and multi-class categories, and the application of data augmentation to improve generalization. Furthermore, feature extraction techniques such as Mel-Frequency Cepstral Coefficients (MFCCs) and spectrograms are introduced, forming the foundation for subsequent training of machine learning and deep learning models.

3.2 Data Acquisition

3.2.1 Data Collection Stages and Challenges

The data collection process was conducted in multiple phases, each presenting unique challenges. One of the main difficulties was ensuring consistent recording conditions and avoiding environmental noise, which could negatively affect the quality of the engine sound samples.

3.2.2 Recording Conditions

To guarantee high-quality acoustic data, recordings were performed in quiet settings, either outdoors away from traffic or indoors within a workshop, avoiding background noise from other machines. A controlled environment was essential for capturing clean and distinguishable engine sounds.

Figure 3.1 shows the workshop where many of the sounds were recorded.

Figure 3.1: Workshop

3.2.3 Engine Types and Fault Scenarios

This study focused on two common types of internal combustion engines: gasoline and diesel. Each fault scenario was diagnosed by a professional before the recording to ensure the accuracy of the collected samples. This approach ensured the reliability of the labels used for training and evaluation.

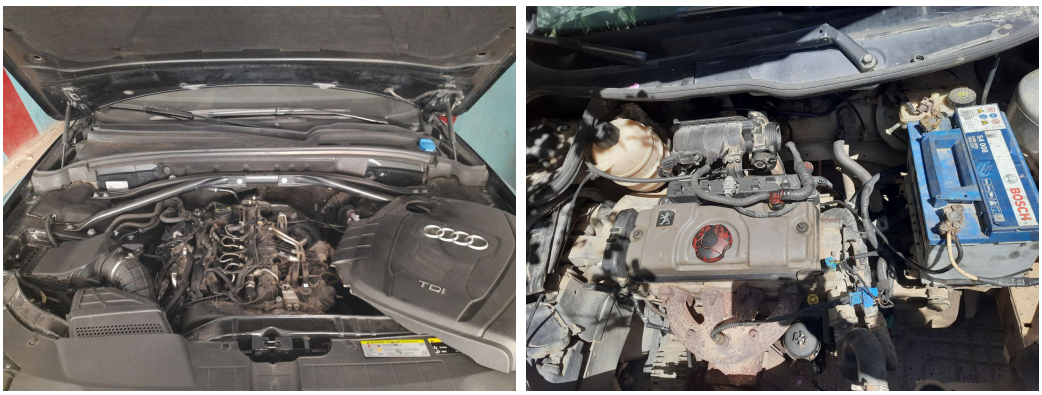

Figure 3.2: car engine

3.2.4 Recording Equipment and Settings

A standard smartphone was used for recording, taking advantage of its built-in microphone. To achieve optimal audio capture, the hood was raised as shown in the figure 3.2, and the

phone was placed close to the engine while it was running. This provided a live and detailed recording of the engine's mechanical behavior..

3.2.5 Data Labeling and Organization

The collected data was organized into labeled categories based on engine condition. For gasoline engines, four different fault types were recorded along with healthy samples. For diesel engines, three fault types were included, also accompanied by normal engine recordings. This structured dataset supports both binary classification (healthy vs. faulty) and multi-class classification based on fault type.

Figure 3.3 illustrates the recorded audio files, while Figure 3.4 presents the organization of the engine fault categories into separate directories.

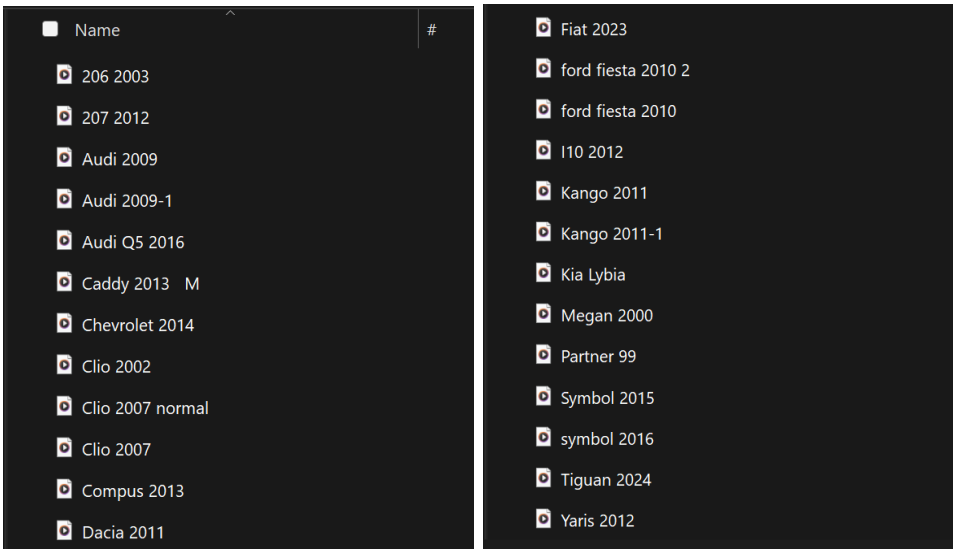

Figure 3.3: Sound files

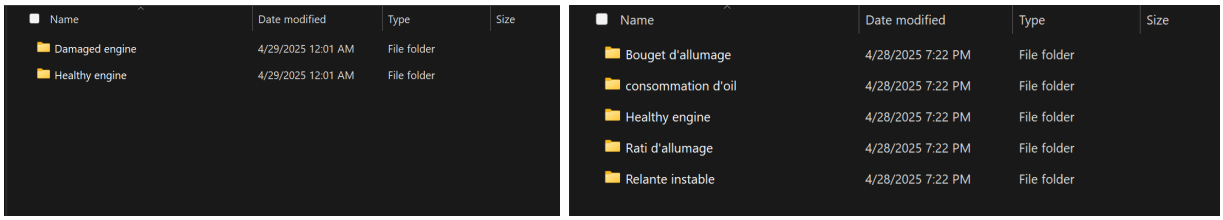

Figure 3.4: Sound folders

3.3 Data Preprocessing and Feature Engineering

3.3.1 Audio Segmentation

To ensure uniform input sizes for training the models, all engine audio recordings were segmented into fixed-length samples of 5 seconds each. This duration balances between capturing enough temporal information and maintaining computational efficiency.

Listing 27 is a Python script to split audio files into 5-second chunks.

```
import librosa
import numpy as np
import os
import soundfile as sf

def split_audio(input_file, output_dir, segment_length=5):
    y, sr = librosa.load(input_file, sr=None)
    samples_per_segment = segment_length * sr
    os.makedirs(output_dir, exist_ok=True)

    filename = os.path.splitext(os.path.basename(input_file))[0]

    for i, start in enumerate(range(0, len(y), samples_per_segment)):
        end = start + samples_per_segment
        segment = y[start:end]

        if len(segment) < samples_per_segment:
            segment = np.pad(segment, (0, samples_per_segment - len(segment)),
                mode='constant')

        output_path = os.path.join(output_dir, f"Rati d'allumage_{i}.wav")
        sf.write(output_path, segment, sr)
        print(f" Saved: {output_path}")

input_file = r"C:\Users\user\OneDrive\Bureau\dataset Essence\sound\Rati d'
    allumage\Val_train\206 2003.m4a"
output_dir = r"C:\Users\user\OneDrive\Bureau\dataset Essence\5s\Rati d'allumage\
    Val_train"
split_audio(input_file, output_dir)
```

Listing 3.1: Split audio files

3.3.2 Data Augmentation Techniques

To improve the robustness and generalization of the models, three augmentation techniques were applied only to the training set :

- **Gaussian Noise Addition:** White noise was added to simulate realistic disturbances and improve model resilience.
- **Time Shifting:** Audio signals were shifted slightly to the left or right to simulate temporal variations.
- **Amplitude Scaling:** Signals were multiplied by a scaling factor to mimic volume fluctuations during recording.

These methods help reduce overfitting and expand the diversity of the training data.

Listing 15 represents audio data augmentation: noise, shift, and expansion.

```
#Gaussian
def add_noise(data, noise_level=0.01):
    noise = np.random.normal(0, noise_level, data.shape)
    return data + noise

#(shift)
def time_shift(data, shift_max=2):
    shift = np.random.randint(-shift_max, shift_max + 1)
    return np.roll(data, shift)

#(scaling)
def scale(data, scale_range=(0.8, 1.2)):
    scale_factor = np.random.uniform(*scale_range)
    return data * scale_factor
```

Listing 3.2: Audio Data Augmentation

3.3.3 MFCC Feature Extraction

Mel Frequency Cepstral Coefficients (MFCCs) were extracted as key features for training traditional ML models As shown in Listing 34:

```
import librosa
import numpy as np
```

```
import os
import pandas as pd
from tqdm import tqdm

def extract_features(file_path, n_mfcc=20):
    y, sr = librosa.load(file_path, sr=None)
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc)
    mfccs_mean = np.mean(mfccs, axis=1)
    return mfccs_mean

def process_all_segments(input_dir, output_csv):
    features = []
    filenames = []
    labels = []

    for filename in tqdm(os.listdir(input_dir)):
        if filename.endswith(".wav"):
            file_path = os.path.join(input_dir, filename)
            mfccs = extract_features(file_path)
            features.append(mfccs)
            filenames.append(filename)
            labels.append("Damaged engine")

    df = pd.DataFrame(features, index=filenames)
    df['label'] = labels
    df.to_csv(output_csv)
    print(f"Features saved in: {output_csv}")

input_dir = r"C:\Users\user\OneDrive\Bureau\AI 2\dataset Essence\Binary class\
    damaged test"
output_csv = r"C:\Users\user\OneDrive\Bureau\AI 2\dataset Essence\Binary class\
    Test\Damaged engine.csv"
process_all_segments(input_dir, output_csv)
```

Listing 3.3: Extracting MFCC features and saving to CSV

- For initial experiments, 13 MFCCs were extracted from a single recording of each fault, and divided into training, validation, and test sets.
- In a more advanced setup, 20 MFCCs were extracted from multiple recordings, where training and validation sets were combined from various samples, while the testing set

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
name	mfcc 1	mfcc 2	mfcc 3	mfcc 4	mfcc 5	mfcc 6	mfcc 7	mfcc 8	mfcc 9	mfcc 10	mfcc 11	mfcc 12	mfcc 13	label
bougies_allumage_308.wa	-6.84326	101.2371	-28.8021	8.35597	-12.9624	19.6639	-42.5746	30.00125	-32.0818	28.28591	-19.637	15.72657	-12.7159	Spark plugs
Oil_water_pollution_176.v	-129.078	82.45339	-38.6655	15.70412	-25.5203	15.94014	-20.9442	29.28908	-37.6388	24.78892	-29.3069	28.49183	-22.1632	Water Contamination in Engine Oil
normal_y_116.wav	-176.428	59.21809	-8.55031	15.50653	-1.14071	13.07081	-11.163	19.60408	-14.4633	15.88276	-14.1874	10.85817	-6.65991	healthy engine
doigts_151.wav	-114.088	115.8667	-53.0606	44.04248	-21.8019	16.92681	-26.0007	26.23805	-4.02514	17.00835	-4.08709	9.331358	3.831129	doigts de culbuteur
manque_puissance_209.w	-79.7957	97.75919	-58.2269	34.88519	-28.4073	31.40185	-41.3731	23.15772	-33.6248	4.985626	-23.3381	1.078895	-10.1418	manque de puissance
Oil_water_pollution_235.v	-107.091	83.76303	-34.5939	23.87494	-20.4921	15.0358	-14.3468	28.2494	-40.7472	26.70912	-33.6298	25.84728	-21.0676	Water Contamination in Engine Oil
Oil_water_pollution_361.v	-113.707	82.10725	-35.8404	18.08632	-20.6805	13.61042	-16.003	25.98205	-42.8208	25.8596	-32.3206	28.45361	-22.0607	Water Contamination in Engine Oil
doigts_266.wav	-114.944	117.668	-49.3965	44.52072	-21.4235	17.02045	-25.3647	24.45834	-5.37347	17.15578	-4.21504	10.3789	3.404471	doigts de culbuteur
doigts_181.wav	-105.759	124.4043	-45.219	47.57483	-18.4015	18.70012	-24.7458	25.36501	-5.92482	17.25553	-4.55883	6.157148	-1.45832	doigts de culbuteur
manque_puissance_148.w	-84.2578	99.20769	-53.5293	32.08726	-30.5161	33.18181	-41.0447	23.3681	-33.9026	11.82808	-17.5177	7.154284	-6.16767	manque de puissance
doigts_95.wav	-115.493	117.7117	-48.8737	44.73183	-21.8325	16.94527	-24.0325	24.87786	-5.08073	16.39068	-3.40389	10.47413	4.251324	doigts de culbuteur
doigts_369.wav	-115.103	118.0589	-49.8105	44.0452	-22.484	16.73609	-25.5742	25.29848	-5.40361	17.25938	-4.33639	11.00677	3.744587	doigts de culbuteur
manque_puissance_260.w	-78.5762	98.83073	-58.2441	34.66391	-26.3947	33.46423	-40.8425	25.46751	-33.63	12.46998	-17.5607	5.015697	-7.9691	manque de puissance
normal_f_6.wav	-129.174	68.18949	-11.1365	24.13625	-7.77918	13.32238	-12.6424	3.515116	-11.0474	4.162941	-13.5514	14.58298	-7.9591	healthy engine
doigts_337.wav	-113.915	115.9046	-53.5389	42.97941	-23.6127	17.04969	-27.2384	25.90521	-5.48851	16.53571	-5.24737	9.519712	3.227667	doigts de culbuteur

Figure 3.5: Extract 13 MFCC

was kept distinct by using a different recording of the same fault type, ensuring that the model generalizes well to unseen data and avoids overfitting.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
mfcc_1	mfcc_2	mfcc_3	mfcc_4	mfcc_5	mfcc_6	mfcc_7	mfcc_8	mfcc_9	mfcc_10	mfcc_11	mfcc_12	mfcc_13	mfcc_14	mfcc_15	mfcc_16	mfcc_17	mfcc_18	mfcc_19	mfcc_20	label	
-137.64	103.05	-32.089	6.55191	-5.626	4.87325	-10.92	18.245	-14.138	11.68385	-17.106	12.239	-10.387	6.08025	-11.3882	4.45398	-9.03945	7.370397	-7.37099	10.0967	Relante instable	
-115.28	106.56	-14.761	8.84217	-11.789	14.09624	0.2254	18.7953	-13.1146	13.78784	-12.117	2.04916	-8.3631	9.07844	-3.95268	11.12084	-3.97737	8.51971	-9.11113	5.703806	Bouget d'allumage	
-173.73	78.732	-3.0414	7.99594	3.47624	6.15563	-17.946	8.83969	-23.0182	8.919232	-23.115	3.57053	-11.395	1.7629	-5.80792	10.97487	1.625552	12.95154	-8.38926	9.964821	consommation d'oil	
-97.971	76.876	-12.083	7.53115	-15.688	16.79088	-4.9159	24.4299	-17.1427	18.1539	-18.76	9.0594	-6.3803	17.0216	-6.23708	8.376928	-9.95934	4.418342	-12.5398	15.95307	Rati d'allumage	
-137.09	102.62	-29.589	10.712	-1.8932	7.912591	-7.9866	20.7076	-12.5463	13.06878	-14.609	15.2731	-9.8947	6.27789	-9.19717	4.980387	-9.27518	8.421723	-6.09298	13.01332	Relante instable	
-95.334	77.795	-11.72	8.16691	-15.357	14.73819	-5.5749	22.3994	-13.7228	18.37856	-17.41	9.16053	-8.438	14.7786	-5.50356	10.00401	-9.48318	6.606116	-12.5162	15.50277	Rati d'allumage	
-177.1	80.6	-6.0828	9.45046	2.38109	7.651016	-20.123	10.1781	-24.439	8.960987	-23.46	6.54071	-13.334	1.25964	-5.69733	13.59717	-0.57957	13.40776	-9.23142	10.70543	consommation d'oil	
-106.65	100.49	-19.872	7.80157	-15.478	13.2512	-3.9623	15.4133	-19.4632	10.13174	-13.302	3.31276	-13.384	9.10852	-7.63802	8.489174	-7.43412	10.26903	-10.8995	7.725575	Bouget d'allumage	
-118.37	107.78	-13.056	9.65457	-10.331	16.5639	1.4158	21.2859	-11.1445	15.63813	-10.766	3.35471	-7.8819	10.5844	-2.61338	12.13776	-4.04968	10.16221	-8.09204	7.287207	Bouget d'allumage	
-170.43	53.072	-8.4351	19.0641	0.1603	11.93365	-12.358	23.4258	-19.192	16.12042	-13.698	11.5408	-8.4152	9.19766	-0.92582	13.68991	-9.98034	6.933007	-9.86557	10.08765	Healthy engine	
-106.98	75.275	-13.562	2.91409	-17.176	14.94467	-7.6	24.9339	-14.5283	17.86779	-17.351	7.98644	-10.59	13.5558	-6.06757	12.39409	-9.8492	5.648077	-12.8424	14.98377	Rati d'allumage	
-177.78	59.195	-7.3927	15.8921	-1.1759	13.41813	-11.143	21.7412	-15.5556	16.59147	-13.591	12.2075	-7.1488	8.4408	1.983443	14.21657	-9.47097	6.431162	-8.73906	10.25711	Healthy engine	
-138.17	101.79	-30.521	7.75185	-3.6374	6.846344	-9.9061	19.6057	-12.5147	11.789	-16.153	13.9397	-11.833	4.6122	-11.2226	3.957434	-10.2411	7.764605	-7.04592	11.31787	Relante instable	
-176.09	58.403	-7.0119	16.681	-1.5828	12.72258	-10.936	21.6041	-15.7416	17.57243	-12.127	11.614	-7.7052	8.8389	1.706968	13.48168	-9.8627	5.717129	-8.80549	10.2746	Healthy engine	
-139.44	99.057	-31.799	8.11062	-4.3726	5.372908	-8.4777	18.8012	-13.8722	12.00722	-16.142	12.9456	-9.4577	5.52779	-11.0808	3.74181	-9.46636	7.633731	-6.87472	11.65973	Relante instable	
-100.3	100.32	-17.901	9.34522	-14.49	11.95129	-3.0886	15.9083	-12.463	10.31202	-15.857	5.43393	-12.463	11.5045	-9.33067	6.01174	-6.14876	7.685624	-12.0093	10.28489	Bouget d'allumage	
-174.08	58.987	-4.6723	18.1918	-0.5096	12.71728	-10.044	23.2449	-15.7295	17.86629	-12.421	11.4503	-5.9919	11.1247	1.581266	15.06535	-9.05445	6.285292	-9.93595	10.36088	Healthy engine	
-174.63	80.637	-7.4722	7.90345	1.7053	7.10178	-20.348	12.4197	-23.3613	10.42328	-24.005	3.90776	-12.521	1.35094	-7.22083	12.20937	0.877078	14.11711	-10.2571	9.886456	consommation d'oil	
-109.97	104.24	-21.672	6.33088	-15.899	13.67778	-7.5042	16.4669	-18.363	11.0538	-16.405	1.91575	-13.538	6.70486	-6.60764	9.363829	-9.97826	8.614935	-10.5961	6.461275	Bouget d'allumage	
-175.34	58.147	-7.527	16.4729	-0.8955	12.17214	-10.203	22.5611	-14.6596	17.15724	-12.638	11.3422	-7.7567	9.57319	2.184923	14.32746	-9.24503	5.335231	-9.45408	9.65134	Healthy engine	
-116.95	103.09	-18.591	7.85815	-17.355	15.21983	-3.1591	19.1658	-17.5346	13.96374	-15.664	2.10617	-11.723	9.25387	-7.57555	11.00183	-7.00191	10.43811	-11.9386	6.740786	Bouget d'allumage	
-102.8	77.58	-13.096	4.36578	-17.33	15.79632	-5.8433	25.5959	-15.2429	18.81326	-18.465	8.14988	-8.6967	14.6197	-6.55968	11.77142	-10.3089	5.859612	-12.4344	16.12459	Rati d'allumage	
-98.787	79.357	-10.509	8.23138	-15.473	17.09054	-6.0529	24.3469	-16.7278	17.68154	-18.314	8.41791	-7.6367	16.7525	-6.75561	8.275195	-8.86733	6.170224	-12.5956	15.35689	Rati d'allumage	

Figure 3.6: Extract 20 MFCC

3.3.4 Spectrogram Generation

Following the segmentation strategy discussed in Subsection 3.3.1 and implemented in Listing 27, each audio recording was divided into uniform 5-second segments. This step was essential for ensuring fixed-length inputs across all samples, a requirement that facilitates reliable and consistent feature extraction. Such standardization is particularly critical when training machine learning models—especially deep learning architectures—that expect input data of consistent shape.

Building on this, Listing 38 presents the procedure used to convert each audio segment into a mel spectrogram. These visual representations of the time-frequency domain capture essential acoustic patterns, enabling deep learning models such as CNNs to effectively learn fault-specific characteristics from the engine sounds.

```
1 import librosa
```

```
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
import os
import glob

def save_mel_spectrogram(audio_path, output_path, duration=5, sr=22050):
    try:
        y, sr = librosa.load(audio_path, sr=sr, duration=duration)
        S = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=128)
        S_DB = librosa.power_to_db(S, ref=np.max)

        plt.figure(figsize=(3, 3))
        librosa.display.specshow(S_DB, sr=sr, x_axis='time', y_axis='mel')
        plt.axis('off')
        plt.tight_layout()

        plt.savefig(output_path, bbox_inches='tight', pad_inches=0)
        plt.close()
    except Exception as e:
        print(f"Error processing file: {audio_path} --> {e}")

def process_dataset(input_folder, output_folder):
    os.makedirs(output_folder, exist_ok=True)

    for wav_file in glob.glob(os.path.join(input_folder, "*.wav")):
        print(f"Processing file: {wav_file}")
        file_name = os.path.splitext(os.path.basename(wav_file))[0]
        output_image = os.path.join(output_folder, file_name + ".png")

        save_mel_spectrogram(wav_file, output_image)

# Run the code
input_dataset = r"C:\Users\user\OneDrive\Bureau\AI 2\dataset Diesel\5s\Damaged engine\train_val"
output_images = r"C:\Users\user\OneDrive\Bureau\AI 2\dataset Diesel\Spectrogram\train_val\Damaged engine"
process_dataset(input_dataset, output_images)
```

Listing 3.4: Generating and saving mel spectrogram images

Figure 3.7 shows some images extracted from the spectrograms.

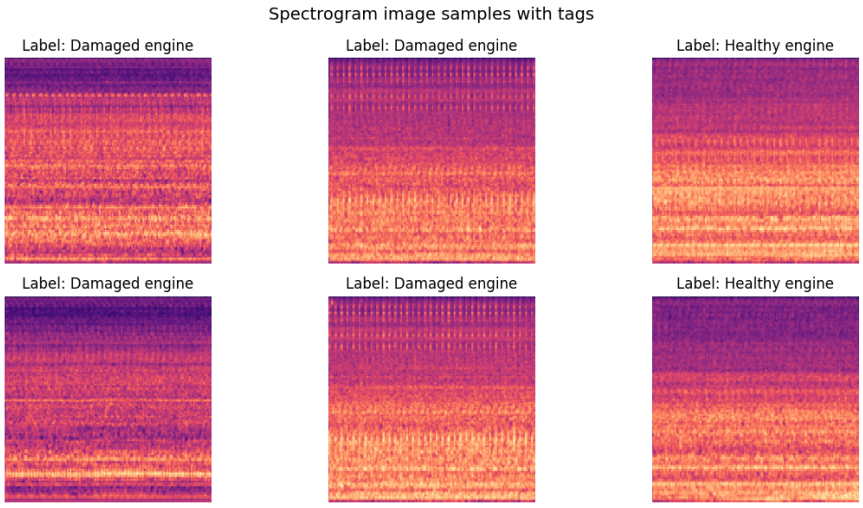

Figure 3.7: Spectrogram

3.3.5 CSV Creation and Label Encoding

Multiple CSV files were created to organize the extracted features and corresponding labels:

- A CSV file containing 13-MFCC features for earlier experiments.
- Separate CSV files for training/validation and testing datasets based on 20 MFCC features.
- Additional CSVs were prepared to store spectrogram image paths and labels, facilitating CNN training.

Labels were encoded numerically for efficient processing in classification tasks.

Figure 3.8 shows the gasoline and diesel folders containing the training files shown in Figure 3.9.

Figure 3.8: CSV Folders

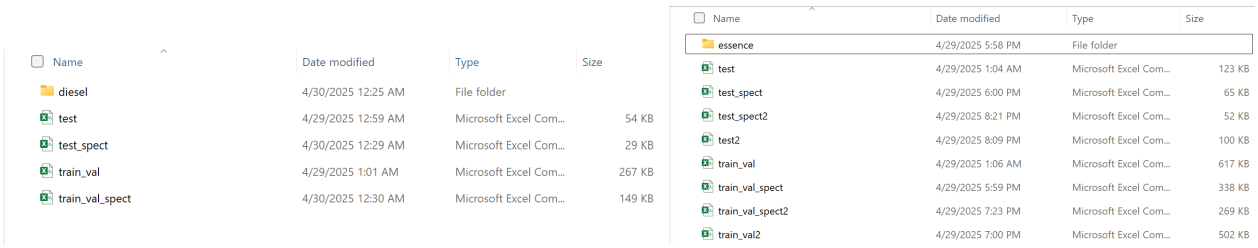

Figure 3.9: CSV files inside the folder

3.4 Dataset Structuring and Splitting

To facilitate the development of robust models, the collected dataset was systematically organized to support both binary classification (healthy vs. faulty engines) and multi-class classification (specific fault types). The data was split into training, validation, and test subsets to ensure proper generalization and unbiased evaluation.

3.4.1 Binary Classification Structure

In the binary classification scenario, the objective was to distinguish between healthy and damaged engines for two fuel types: gasoline and diesel.

Gasoline Engine Dataset

Class	Training	Validation	Test
Healthy Engine	600	150	150
Damaged Engine	600	150	150

Table 3.1: Gasoline engine dataset for binary classification

Diesel Engine Dataset

Class	Training	Validation	Test
Healthy Engine	1000	250	250
Damaged Engine	1000	250	250

Table 3.2: Diesel engine dataset for binary classification

This setup ensures that models are trained on a balanced dataset and tested on completely unseen samples. The training set undergoes data augmentation, as previously mentioned in Listing 15.

3.4.2 Multi-Class Classification Structure (Gasoline Only)

In the multi-class classification setup, the model is trained to distinguish between a healthy engine and four specific fault types in gasoline engines. Equal representation for each class ensures balanced training and fair evaluation.

Class	Training	Validation	Test
Healthy Engine	600	150	150
Bougies d'allumage	420	105	105
Consommation d'huile	420	105	105
Ratés d'allumage	420	105	105
Ralenti instable	420	105	105

Table 3.3: Gasoline engine dataset for multi-class classification

This structure enables the system to effectively learn the acoustic signatures associated with distinct mechanical issues in gasoline engines.

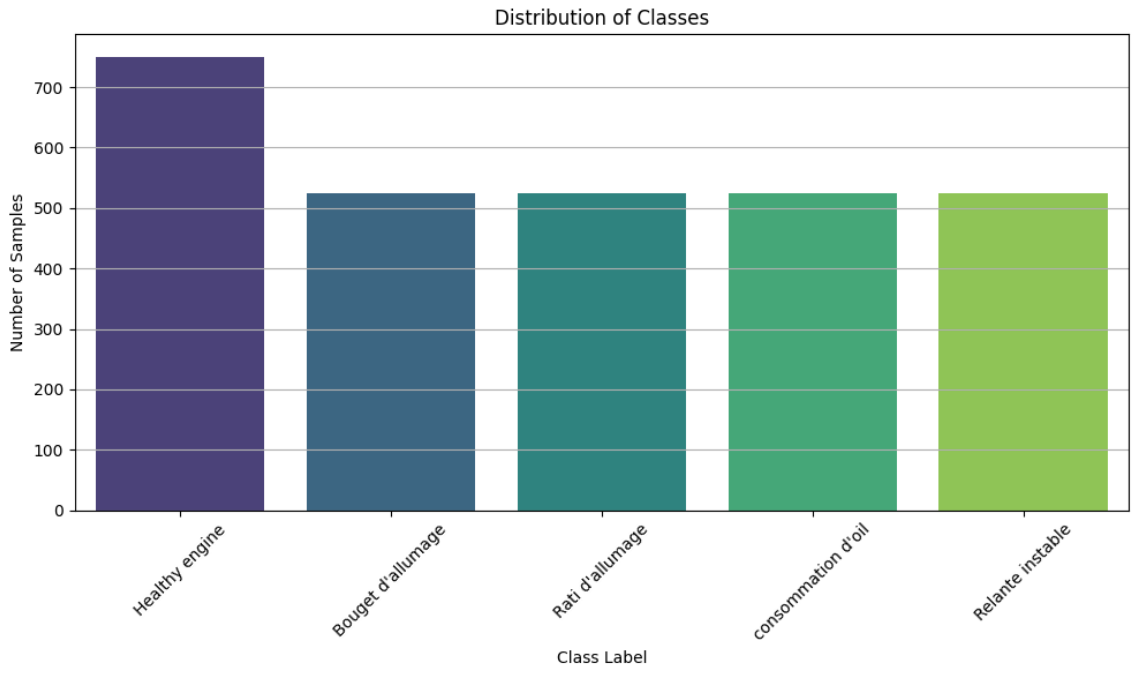

Figure 3.10: Distribution of samples to categories

Table 3.3 and Figure 3.10 provide a detailed overview of the sample distribution across all classes in the multi-class classification setup. This helps to visually and numerically confirm the dataset balance, which is essential for ensuring fair model training and evaluation.

3.5 Conclusion

This chapter presented a comprehensive overview of the data collection and preprocessing pipeline used to construct a reliable sound-based engine fault detection system. Several challenges were encountered during the recording phase, such as minimizing background noise and ensuring accurate fault labeling, which were addressed through controlled recording environments and expert-guided diagnostics.

A detailed explanation of preprocessing techniques, including sample segmentation, data augmentation, MFCC and spectrogram extraction, and CSV file generation, laid the groundwork for training robust machine learning models. Additionally, the datasets were carefully structured for both binary and multiclass classification scenarios, ensuring balanced class distribution and realistic evaluation setups.

These foundations form the basis for the next stage of the project, where the collected and processed data are used to train and evaluate a range of machine learning and deep learning models. The upcoming chapter delves into these experiments, performance metrics, and comparative analyses, highlighting the system's diagnostic potential.

Model Training, Evaluation, and Comparative Analysis

CHAPTER 4

MODEL TRAINING, EVALUATION, AND COMPARATIVE ANALYSIS

4.1 Introduction

To build a reliable fault diagnosis system based on engine sound, a systematic training and evaluation process was conducted. This chapter outlines the development stages of various models, from early tests on small datasets to more robust training on diverse and validated audio samples. Both traditional classifiers (e.g., K-NN, SVM) and deep learning models (e.g., CNNs, LSTMs) were explored.

Experiments included binary and multiclass classification tasks. Initial tests with single recordings per class showed limitations like overfitting, prompting the collection of more diverse data. Further experiments were conducted separately for gasoline and diesel engines using balanced datasets.

This chapter presents dataset configurations, model performances, confusion matrices, and metrics like accuracy and F1-score. The findings help identify the most suitable models for real-world diagnostic applications.

4.2 Single Audio Experiment

4.2.1 Objective and Motivation

This preliminary experiment aimed to assess the feasibility of fault classification using a minimal dataset—one audio sample per class. Five categories were used, and MFCC (13 coefficients and 20) served as the primary feature to evaluate model capability with limited data.

4.2.2 Experimental Setup

The experimental setup involved five fault classes, including healthy and faulty engine conditions such as spark plug issues and water contamination. The dataset was augmented and split into 70% for training, 15% for validation, and 15% for testing. Feature extraction was based on MFCCs, using both 13 and 20 coefficients in different experiments. A range of models were evaluated, including traditional machine learning algorithms (K-NN, Random Forest, XGBoost), neural networks (ANN, LSTM), and a CNN.

4.2.3 Results and Observations

Using both 13 and 20 MFCC features, all models—including CNN—achieved perfect accuracy (100%) across all metrics, indicating strong overfitting due to the limited and homogeneous

dataset.

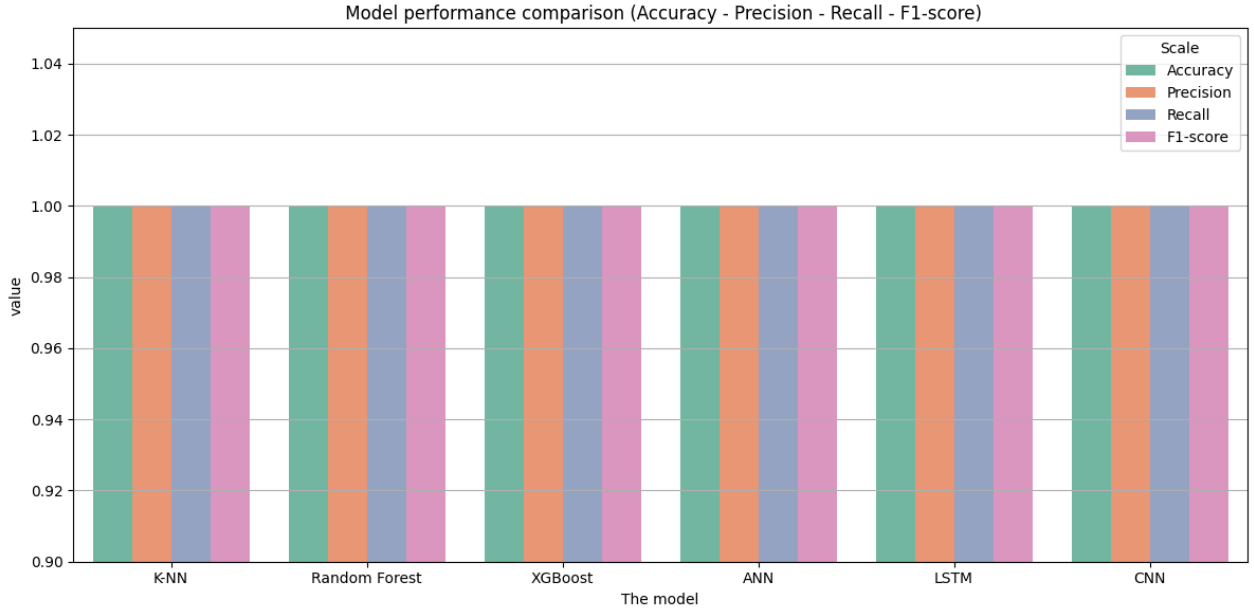

Figure 4.1: comparison (Accuracy - Precision - Recall - F1-score)

Observation:

Despite the excellent performance metrics, this experiment highlighted a critical flaw: the dataset lacked intra-class variability. Since only one recording per class was used, all augmented data shared nearly identical characteristics, leading to memorization rather than generalization, as evidenced in Table 4.1 and Figure 4.1. These visual and statistical representations clearly illustrate the uniformity within each class, which hindered the model's ability to generalize to unseen examples.

4.2.4 Comparative Results Table

Model	MFCCs Used	Accuracy	F1 Score	Observations
K-NN	13 / 20	1.00	1.00	Overfitting due to data similarity
RF	13 / 20	1.00	1.00	High accuracy on all classes
XGBoost	13 / 20	1.00	1.00	Similar pattern as RF and K-NN
ANN	13 / 20	1.00	1.00	Generalized well on the limited set
LSTM	13 / 20	1.00	1.00	Suitable for time-series features
CNN	Spectrogram	1.00	1.00	Applied only with Spectrogram

Table 4.1: Performance comparison for the single-audio experiment

4.2.5 Conclusion and Next Steps

Although all models demonstrated exceptional performance in this controlled scenario, the results are not generalizable to real-world applications due to the lack of diversity in the training data. Consequently, it was necessary to record new audio samples for each fault type, ensuring varied engine conditions and environments. This transition marks the beginning of more realistic training scenarios explored in the subsequent sections of this chapter.

4.3 Multi-Class Classification – Gasoline Engines

To avoid the poor results encountered previously, as discussed in subsubsection 4.2.3 where the main issue was that the test set contained the same samples used during training and validation — the dataset was restructured. In this phase, it was split into two distinct subsets: 80% for training and validation, and 20% for testing. Furthermore, the training/validation set was internally divided to monitor the models' performance during training while ensuring that the test set remained completely unseen throughout the process.

4.3.1 Training with Five Fault Categories

Class Distribution in the Training Set (`train_val.csv`) 3.9:

Label	Samples
Healthy engine	750
Bougie d'allumage	525
Rati d'allumage	525
Consommation d'oil	525
Relante instable	525

Table 4.2: Training set class distribution

After splitting the original dataset into training and validation sets, data augmentation was applied exclusively to the training set, as described in Subsection 3.3.2. For each original image in the training set, three augmented versions were generated, resulting in a more diverse and representative dataset for model training.

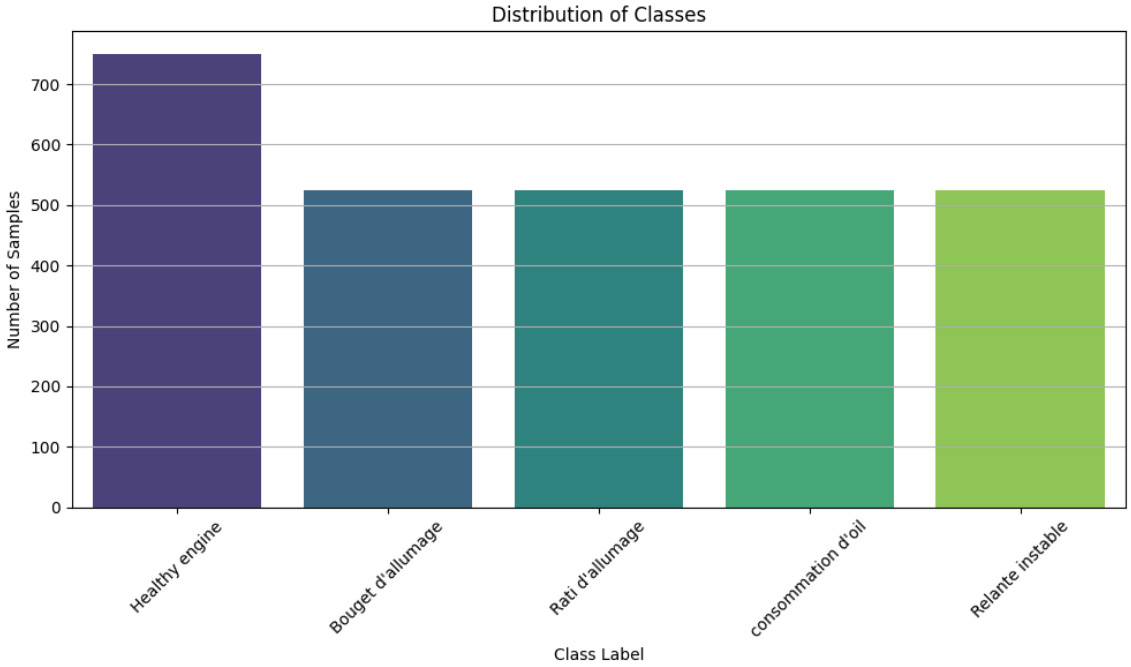

Figure 4.2: istribution of samples to categories (train_val)

Table 4.3 presents the class distribution within the training set after applying data augmentation. Each original image was expanded with three augmented variants to enhance the model's generalization ability across all fault categories.

Label	Healthy engine	Bougie d'allumage	Raté d'allumage	Consommation d'huile	Ralenti instable
Samples	2400	1680	1680	1680	1680

Table 4.3: Training set with data augmentation

Class Distribution in the Test Set (`test.csv`) 3.9:

Label	**Samples**
Healthy engine	150
Bougie d'allumage	105
Rati d'allumage	105
Consommation d'oil	105
Relante instable	105

Table 4.4: Test set class distribution

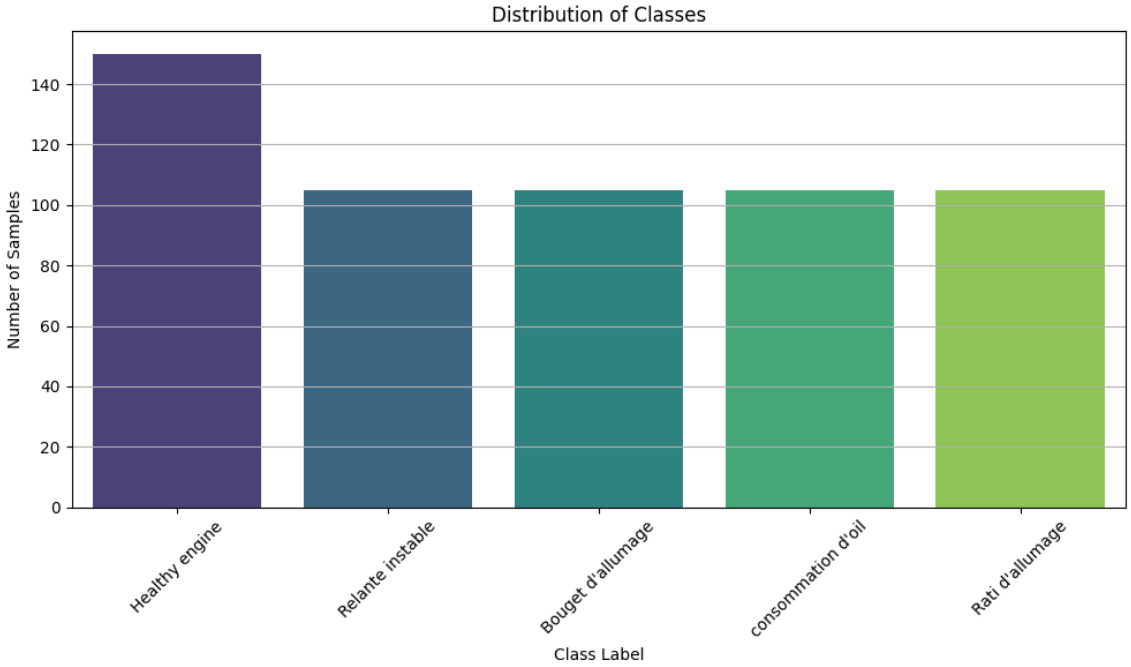

Figure 4.3: istribution of samples to categories (test)

Results and performance comparison of models:

K-NN Model

Figure 4.4 shows the confusion matrix for the K-NN model, while Table 4.5 displays the performance metrics of K-NN in classifying the five fault categories. The table includes important metrics such as precision, recall, F1 score, and overall accuracy.

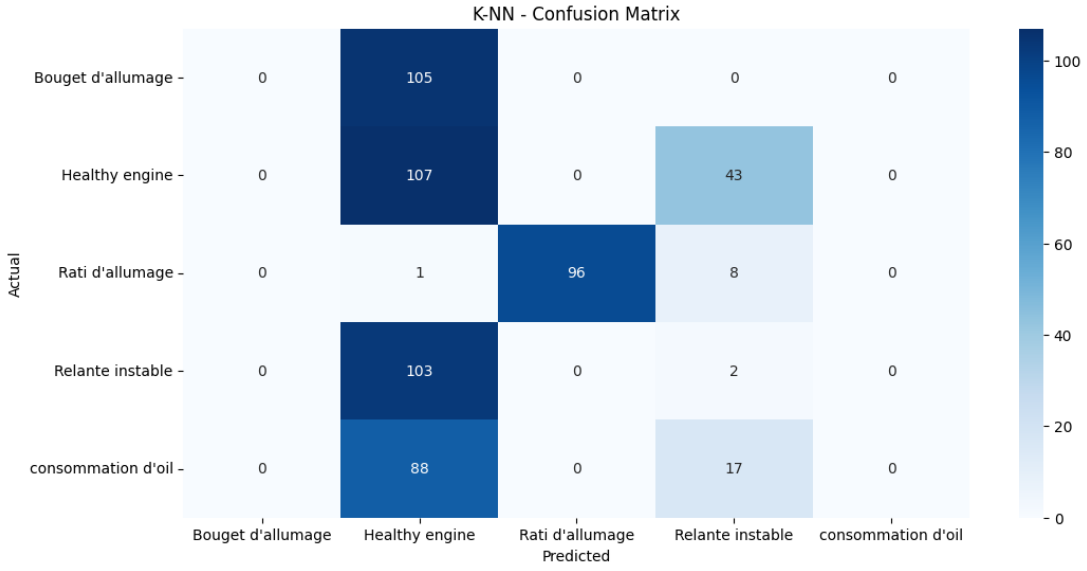

Figure 4.4: Confusion Matrix KNN

Class	Precision	Recall	F1-Score	Support
Bougie d'allumage	0.00	0.00	0.00	105
Healthy engine	0.26	0.71	0.39	150
Raté d'allumage	1.00	0.91	0.96	105
Ralenti instable	0.03	0.02	0.02	105
Consommation d'huile	0.00	0.00	0.00	105
Accuracy			**0.36**	570
Macro avg	0.26	0.33	0.27	570
Weighted avg	0.26	0.36	0.28	570

Table 4.5: KNN Classification report for multi-class classification

Analysis: The K-NN model achieved a relatively low overall accuracy of 36%, with high performance only for the class "Rati d'allumage". The model struggled to correctly classify most of the other fault categories, particularly "Bouget d'allumage" and "Consommation d'oil", which received zero scores across all metrics. This suggests that K-NN may not be suitable for this multi-class classification task due to its sensitivity to feature distribution and class overlap.

Random Forest Model

Figure 4.5 presents the confusion matrix for the Random Forest model, and Table 4.6 provides the classification report for Random Forest in relation to the five fault categories. It includes precision, recall, F1 score, and accuracy.

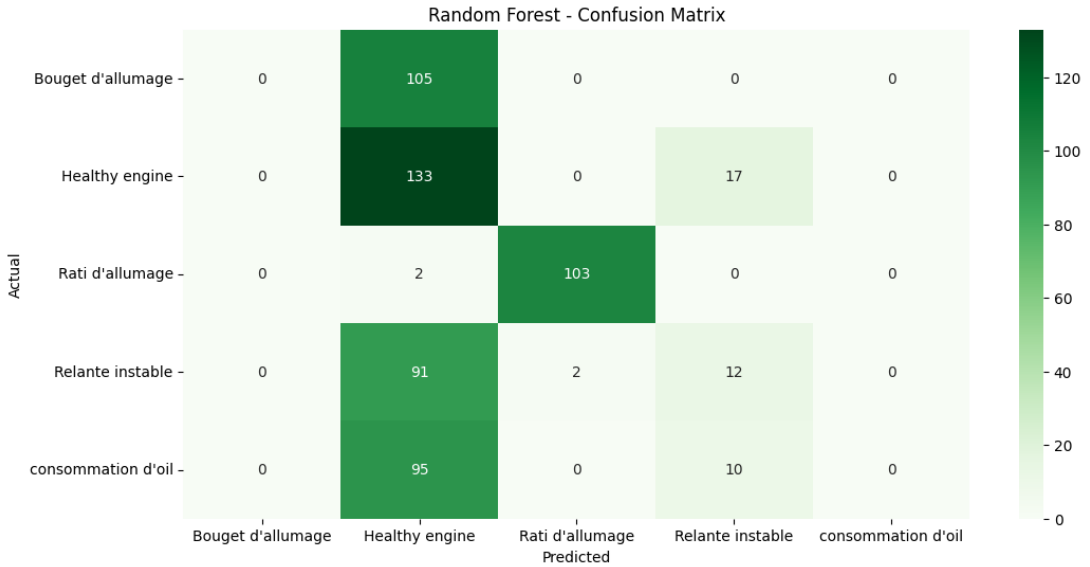

Figure 4.5: Confusion Matrix RF

Class	Precision	Recall	F1-Score	Support
Bougie d'allumage	0.00	0.00	0.00	105
Healthy engine	0.31	0.89	0.46	150
Raté d'allumage	0.98	0.98	0.98	105
Ralenti instable	0.31	0.11	0.17	105
Consommation d'huile	0.00	0.00	0.00	105
Accuracy			**0.44**	570
Macro avg	0.32	0.40	0.32	570
Weighted avg	0.32	0.44	0.33	570

Table 4.6: RF Classification report for multi-class classification

Analysis: The Random Forest model showed slight improvement over K-NN, reaching an accuracy of 44%. It performed particularly well on "Rati d'allumage" and reasonably well on "Healthy engine", but failed to classify the remaining categories effectively. This indicates some robustness in handling dominant patterns, yet the model still lacks generalization across all classes.

XGBoost Model

Figure 4.6 illustrates the confusion matrix for the XGBoost model, while Table 4.7 shows the performance of XGBoost in categorizing the five fault types. This table includes precision, recall, F1 score, and overall accuracy.

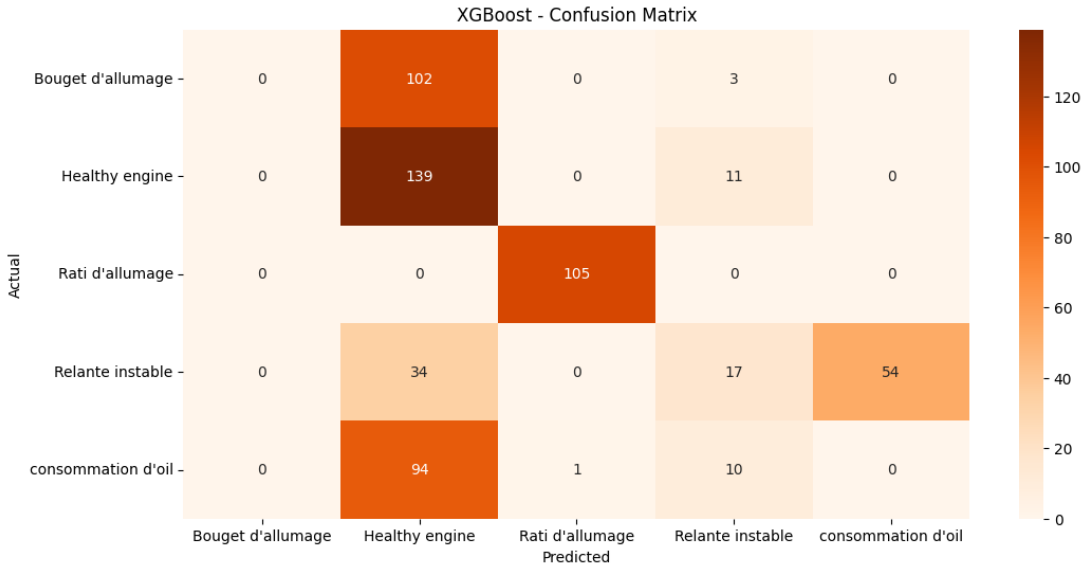

Figure 4.6: Confusion Matrix XGBoost

Class	Precision	Recall	F1-Score	Support
Bougie d'allumage	0.00	0.00	0.00	105
Healthy engine	0.38	0.93	0.54	150
Raté d'allumage	0.99	1.00	1.00	105
Ralenti instable	0.41	0.16	0.23	105
Consommation d'huile	0.00	0.00	0.00	105
Accuracy			**0.46**	570
Macro avg	0.36	0.42	0.35	570
Weighted avg	0.36	0.46	0.37	570

Table 4.7: XGBoost Classification report for the multi-class model

Analysis: XGBoost achieved the highest accuracy among the traditional models (46%), with very strong performance on "Rati d'allumage" and "Healthy engine". Despite this, it still completely failed to classify "Bouget d'allumage" and "Consommation d'oil". This highlights its strong learning capacity, but also a potential overfitting to dominant classes.

ANN Model

Figure 4.7 displays the confusion matrix for the ANN model, with Table 4.8 presenting the classification report for ANN in classifying the five fault categories. The table includes key metrics such as precision, recall, F1 score, and accuracy.

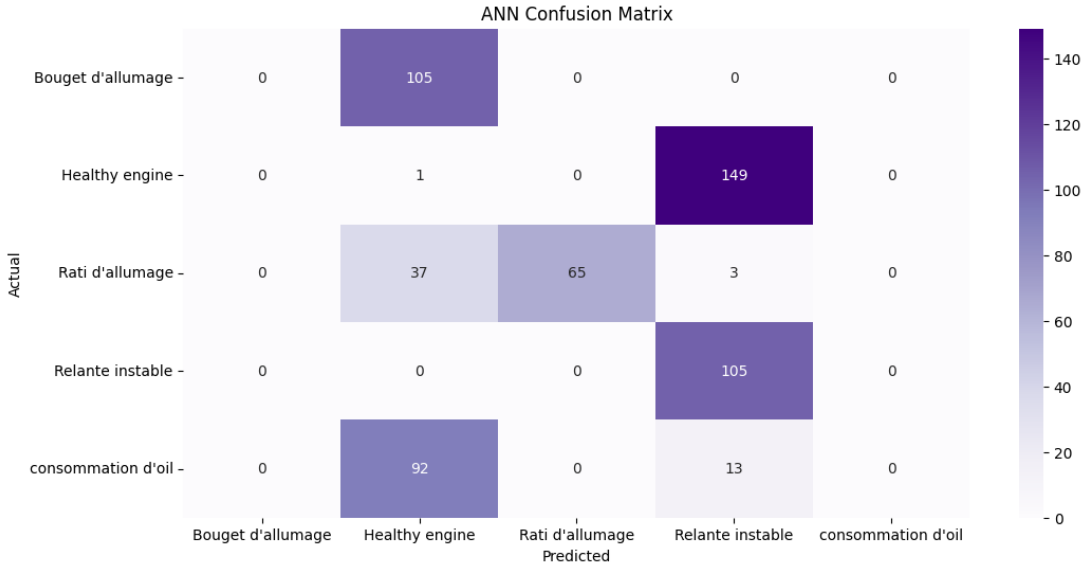

Figure 4.7: Confusion Matrix ANN

Class	Precision	Recall	F1-Score	Support
Bougie d'allumage	0.00	0.00	0.00	105
Healthy engine	0.00	0.01	0.01	150
Raté d'allumage	1.00	0.62	0.76	105
Ralenti instable	0.39	1.00	0.56	105
Consommation d'huile	0.00	0.00	0.00	105
Accuracy			**0.30**	570
Macro avg	0.28	0.33	0.27	570
Weighted avg	0.26	0.30	0.25	570

Table 4.8: ANN Classification report for multi-class classification

Figure 4.8 represents the accuracy and loss curve of the ANN model.

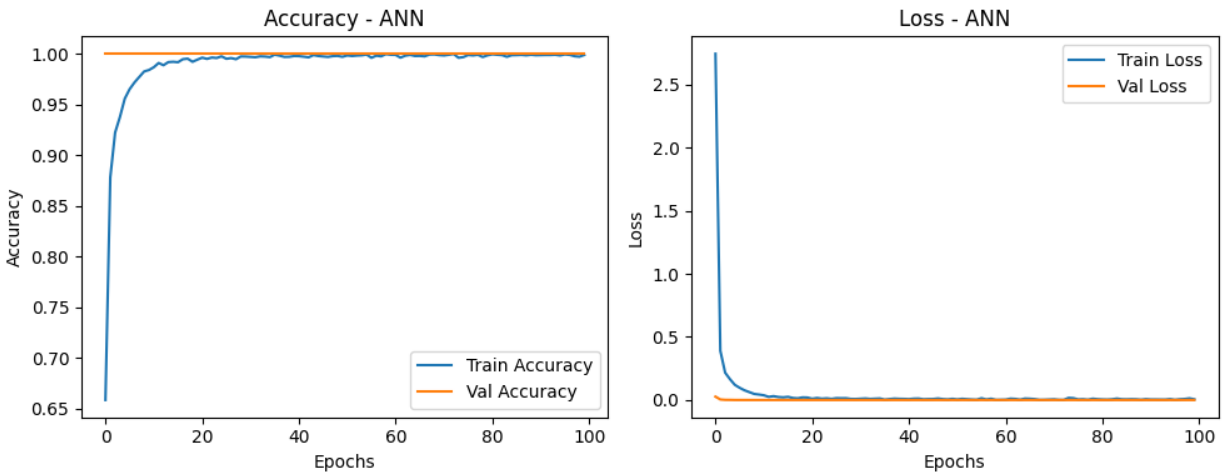

Figure 4.8: Accuracy and Loss (ANN)

Analysis: The Artificial Neural Network (ANN) model achieved 30% accuracy, with effective performance for only two classes: "Rati d'allumage" and "Relante instable". The remaining classes had zero impact on performance, indicating that the model may not have been sufficiently complex or well-regularized to handle this classification task.

LSTM Model

Figure 4.9 shows the confusion matrix for the LSTM model, and Table 4.9 presents the performance of LSTM in classifying the five fault categories. The table includes precision, recall, F1 score, and overall accuracy.

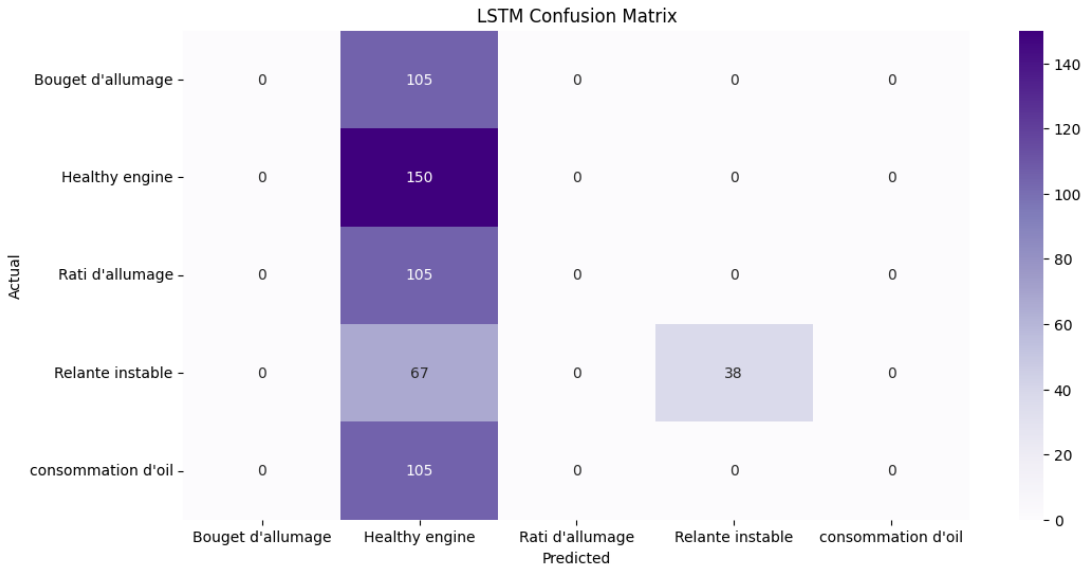

Figure 4.9: Confusion Matrix LSTM

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Bougie d'allumage	0.00	0.00	0.00	105
Healthy engine	0.28	1.00	0.44	150
Raté d'allumage	0.00	0.00	0.00	105
Ralenti instable	1.00	0.36	0.53	105
Consommation d'huile	0.00	0.00	0.00	105
Accuracy			**0.33**	570
Macro avg	0.26	0.27	0.19	570
Weighted avg	0.26	0.33	0.21	570

Table 4.9: LSTM Classification report for multi-class classification

Figure 4.10 represents the accuracy and loss curve of the LSTM model.

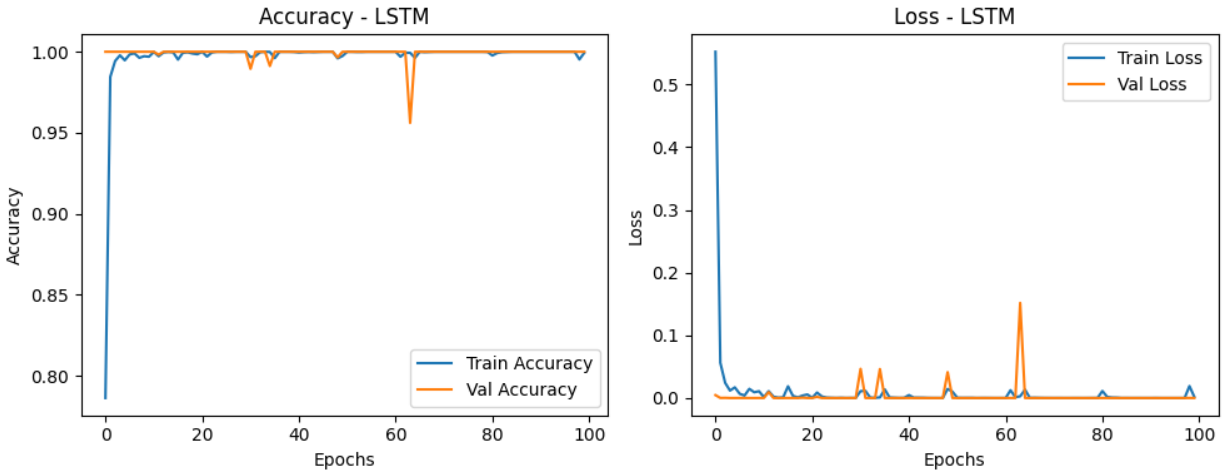

Figure 4.10: Accuracy and Loss (LSTM)

Analysis: LSTM reached a 33% accuracy, showing decent classification of "Healthy engine" and "Relante instable". However, it failed to detect three out of five classes, indicating that its temporal learning capabilities did not translate well into distinguishing between subtle sound patterns across multiple engine fault types.

CNN Model

Figure 4.11 illustrates the confusion matrix for the CNN model, and Table 4.10 presents the classification report for CNN, showing its performance in categorizing the five fault categories. It includes precision, recall, F1 score, and accuracy.

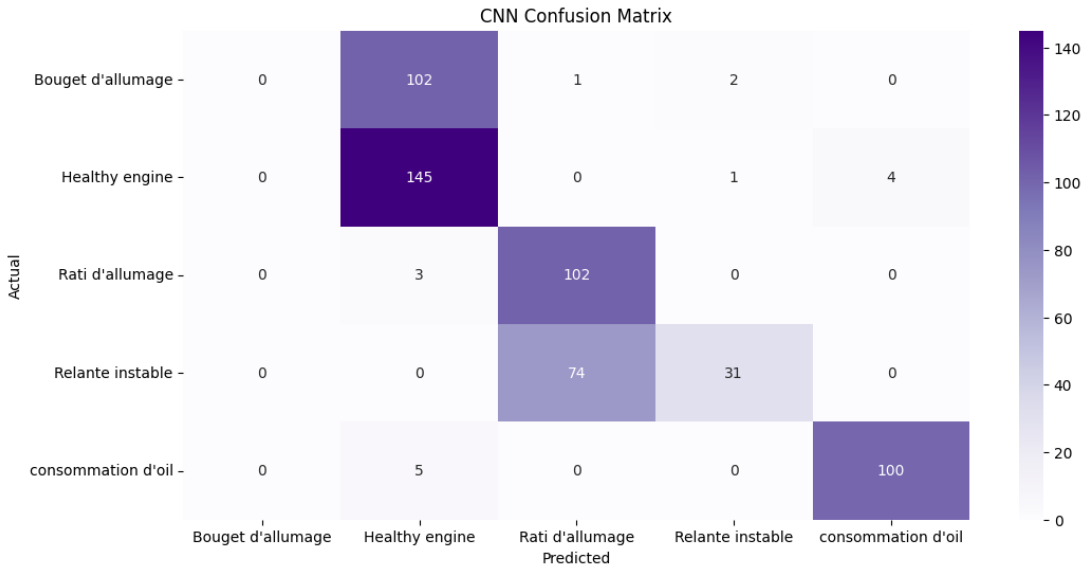

Figure 4.11: Confusion Matrix CNN

Class	Precision	Recall	F1-Score	Support
Bougie d'allumage	0.00	0.00	0.00	105
Healthy engine	0.57	0.97	0.72	150
Raté d'allumage	0.58	0.97	0.72	105
Ralenti instable	0.91	0.30	0.45	105
Consommation d'huile	0.96	0.95	0.96	105
Accuracy			**0.66**	570
Macro avg	0.60	0.64	0.57	570
Weighted avg	0.60	0.66	0.58	570

Table 4.10: CNN Classification report for multi-class classification

Figure 4.12 represents the accuracy and loss curve of the CNN model.

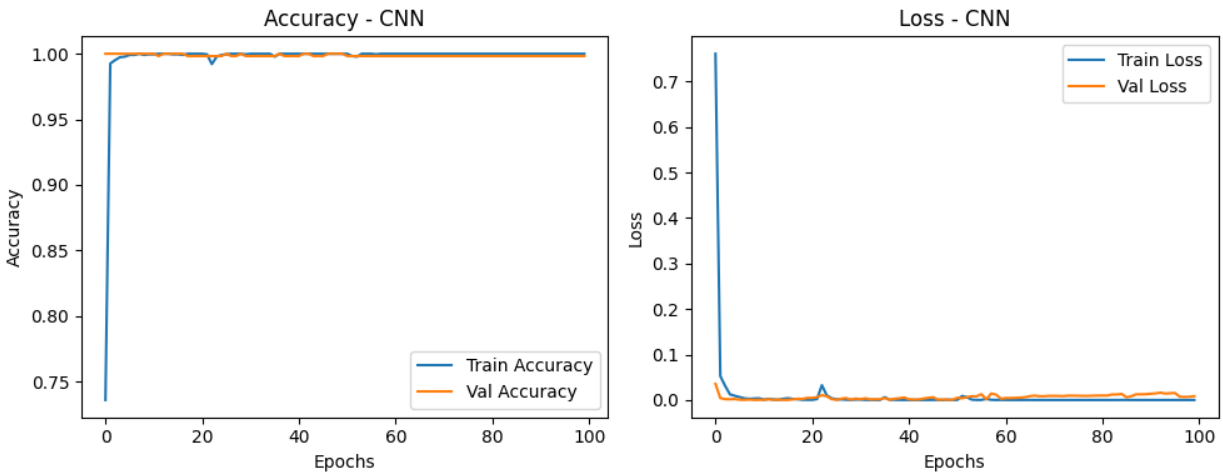

Figure 4.12: Accuracy and Loss (CNN)

Analysis: The CNN model significantly outperformed all other models with an accuracy of 66%. It demonstrated strong generalization across most fault categories, especially "Consommation d'oil" and "Healthy engine". Despite the challenge with "Bouget d'allumage", the results suggest that convolutional architectures are better suited for learning from spectrogram-based audio data.

Figure 4.13 shows the performance of the models (CNN LSTM ANN) in terms of accuracy and loss. Figure 4.14 shows the performance of all models in terms of accuracy and loss.

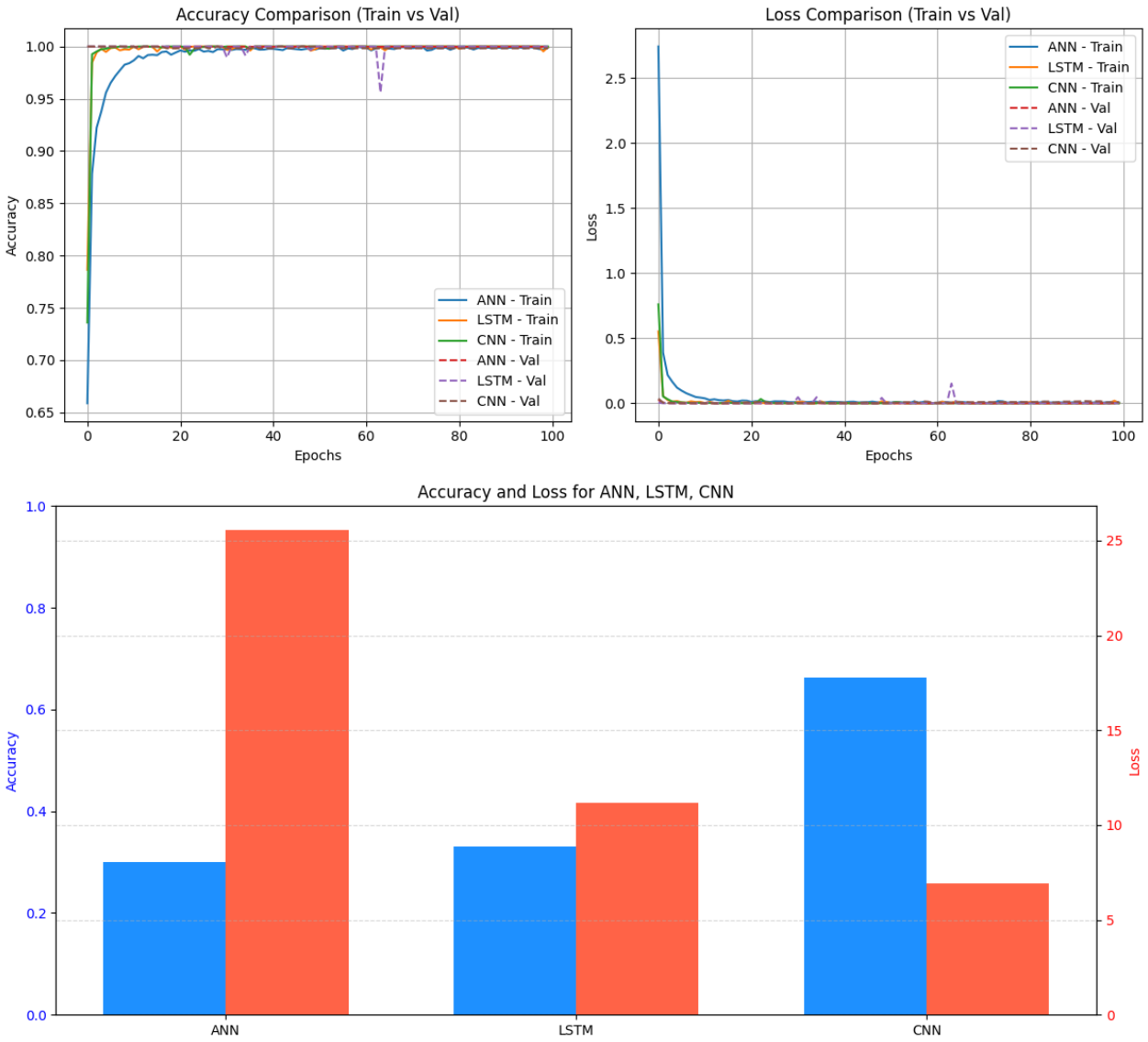

Figure 4.13: Accuracy and Loss

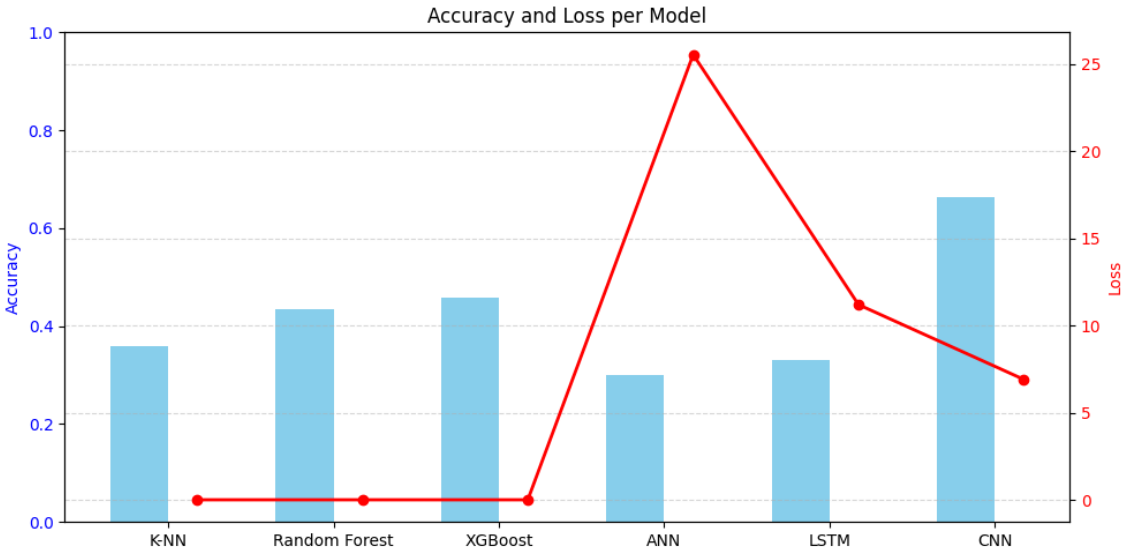

Figure 4.14: Accuracy and Loss all models

Table 4.11 shows the performance of the models in classifying engine faults into five categories. Figure 4.15 also shows the performance of all models on (accuracy Precision Recall F1-Score).

Figure 4.16 also shows the performance of all models on the test set.

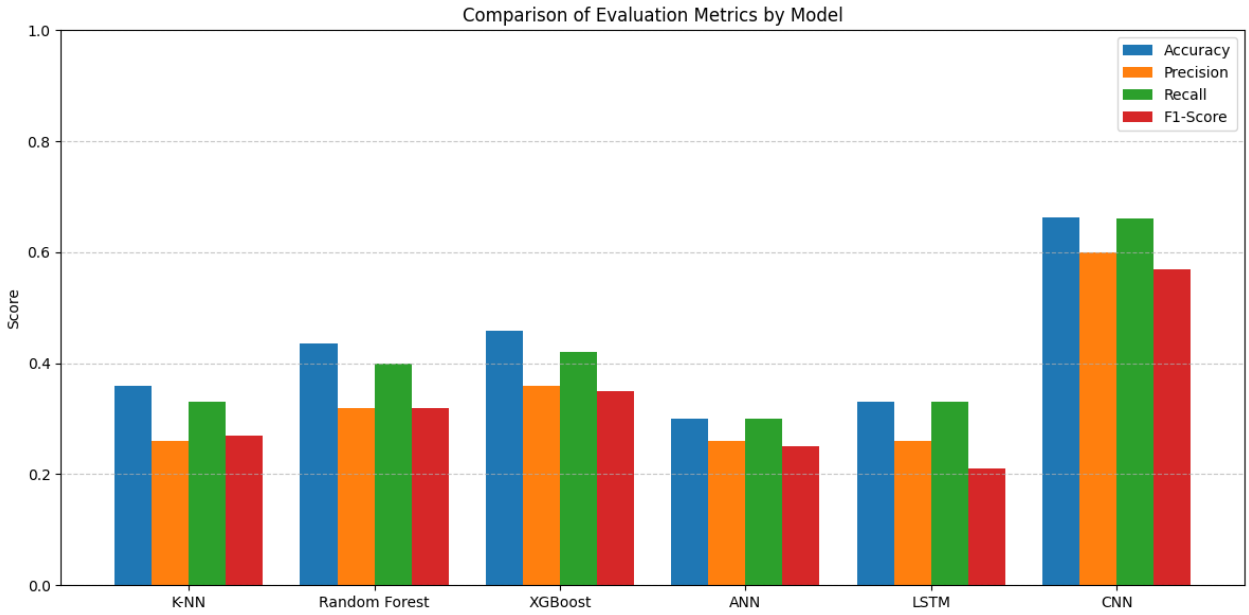

Figure 4.15: Comparison of Evaluation Metrics by Model

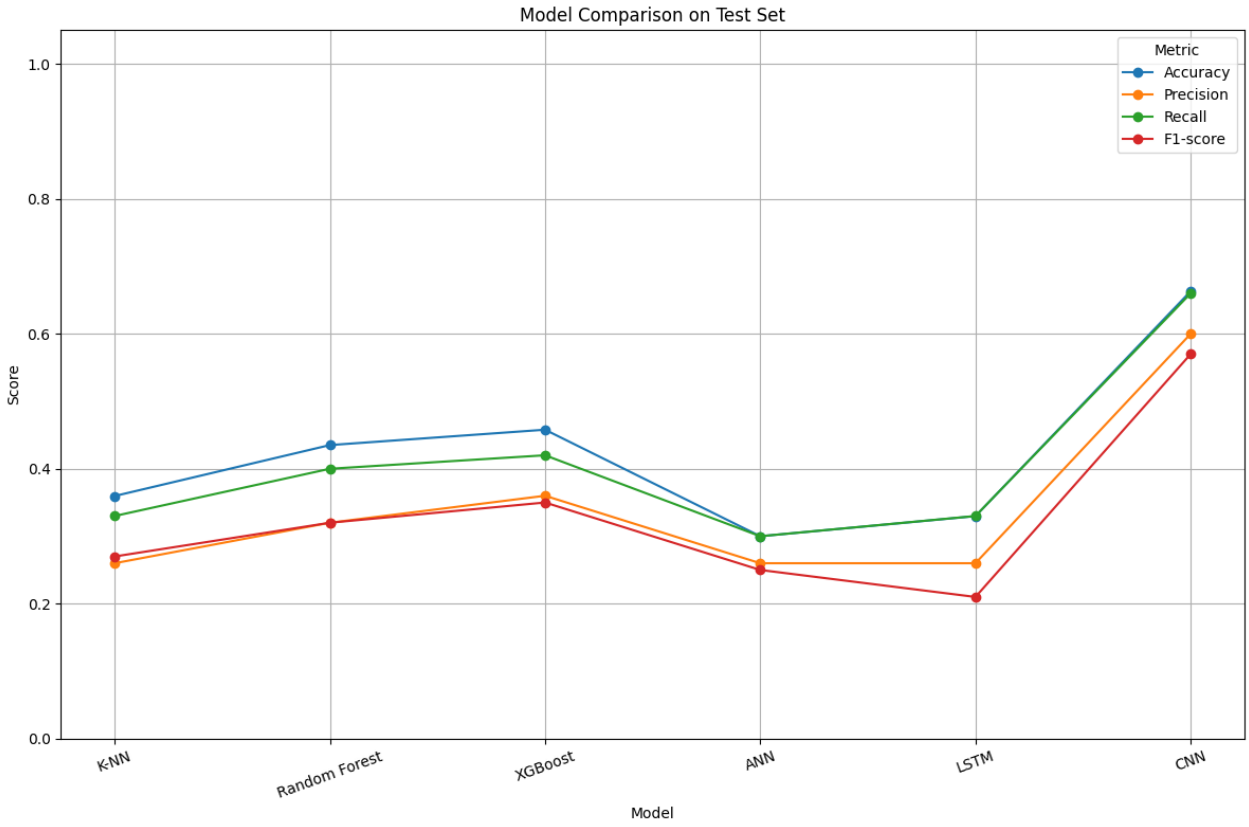

Figure 4.16: Model Comparison on Test set

Model	**Accuracy**	**Loss**	**Observations**
K-NN	36%	N/A	Achieved high recall for *Raté d'allumage* (0.91), but performed poorly on other classes; struggled with minority class separation.
Random Forest	44%	N/A	Good detection of *Raté d'allumage* (recall: 0.98), high precision for *Healthy engine*; failed to generalize on some fault classes.
XGBoost	46%	N/A	Best macro performance among classical models; excellent on *Raté d'allumage* and *Healthy engine*, but failed with minority faults.
ANN	30%	25.54	Moderate recall for *Ralenti instable* (1.00), but almost no performance on other classes; underfitting likely.
LSTM	33%	11.20	Strong on *Healthy engine* and *Ralenti instable*, but failed to recognize other classes; temporal modeling not fully effective.
CNN	**66.3%**	6.90	Outperformed all other models; strong precision and recall for *Healthy engine*, *Raté d'allumage*, and *Consommation d'huile*. Struggled only with *Bougie d'allumage*.

Table 4.11: Performance comparison

Analysis and Discussion

The results indicate that among the six tested models, the Convolutional Neural Network (CNN) achieved the highest overall accuracy (66.3%) and demonstrated robust performance across most engine fault categories. In contrast, traditional models such as K-NN, Random Forest, and XGBoost achieved moderate accuracies (36–46%) and primarily excelled in detecting the more represented classes like Healthy engine and Raté d'allumage. However, their performance on minority classes remained weak due to class imbalance and limited feature abstraction capabilities.

The Artificial Neural Network (ANN) and LSTM models showed subpar performance, likely due to insufficient depth or temporal dependency misalignment with the audio features. Despite

the LSTM's theoretical advantage in sequence modeling, its inability to generalize beyond dominant patterns suggests that the MFCC features alone may not capture enough temporal detail.

Overall, the CNN benefited from the spectrogram-based image representation, which captured richer patterns in time-frequency space. Its strong results support the notion that deep learning with appropriate feature representation can significantly improve fault classification accuracy, especially in multi-class audio diagnosis tasks.

4.3.2 Training After Removing the Problematic Class (Bougie d'allumage)

After observing that several models consistently failed to classify "Bougie d'allumage", this class was removed and training was repeated using the remaining four classes.

Updated class distribution (`train_val2.csv` and (`test2.csv`) 3.9:

Class	train_val	test
Healthy engine	750	150
Rati d'allumage	525	105
Consommation d'oil	525	105
Relante instable	525	105

Table 4.12: Distribution of samples by category in the training/validation and test sets

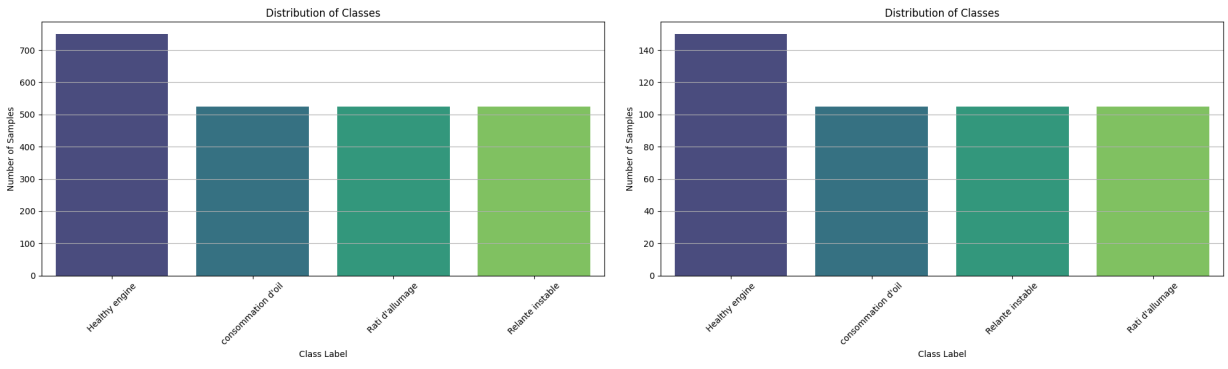

Figure 4.17: Distribution of samples by category

As previously mentioned in subsubsection 4.3.1, data augmentation was performed on the training set after the dataset was split.

Results and performance comparison of models:

K-NN Model

Figure 4.18 presents the confusion matrix for the K-NN model after removing the problematic class, while Table 4.13 displays the updated classification metrics for the remaining four fault categories. The table includes precision, recall, F1 score, and accuracy.

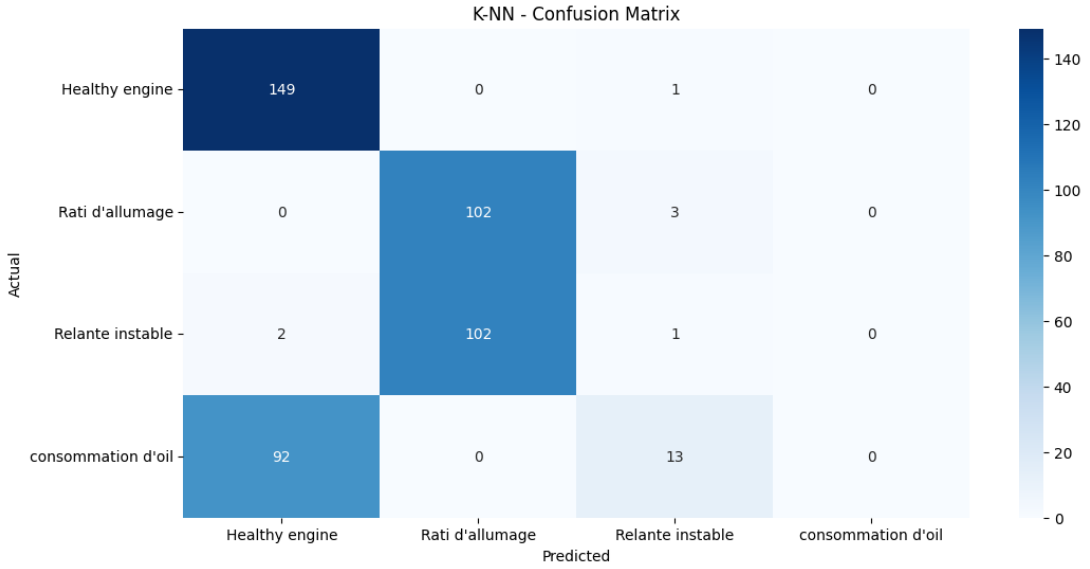

Figure 4.18: Confusion Matrix KNN

Classe	Precision	Recall	F1-score	Support
Healthy engine	0.61	0.99	0.76	150
Raté d'allumage	0.50	0.97	0.66	105
Ralenti instable	0.06	0.01	0.02	105
Consommation d'huile	0.00	0.00	0.00	105
Accuracy			**0.54**	465
Macro Average	0.29	0.49	0.36	465
Weighted Average	0.32	0.54	0.40	465

Table 4.13: Classification results for each class (model KNN)

Analysis: The K-NN model showed a noticeable improvement in accuracy (54%) compared to the previous setting. It performed well in classifying "Healthy engine" and "Rati d'allumage" but failed to recognize "consommation d'oil" and "Relante instable," as indicated by their very low recall and precision scores. This suggests that K-NN struggles with more complex or overlapping audio features in certain fault categories.

Random Forest Model

Figure 4.19 shows the confusion matrix for the Random Forest model, and Table 4.14 illustrates its performance in detecting the remaining four fault types. The table summarizes precision, recall, F1 score, and accuracy.

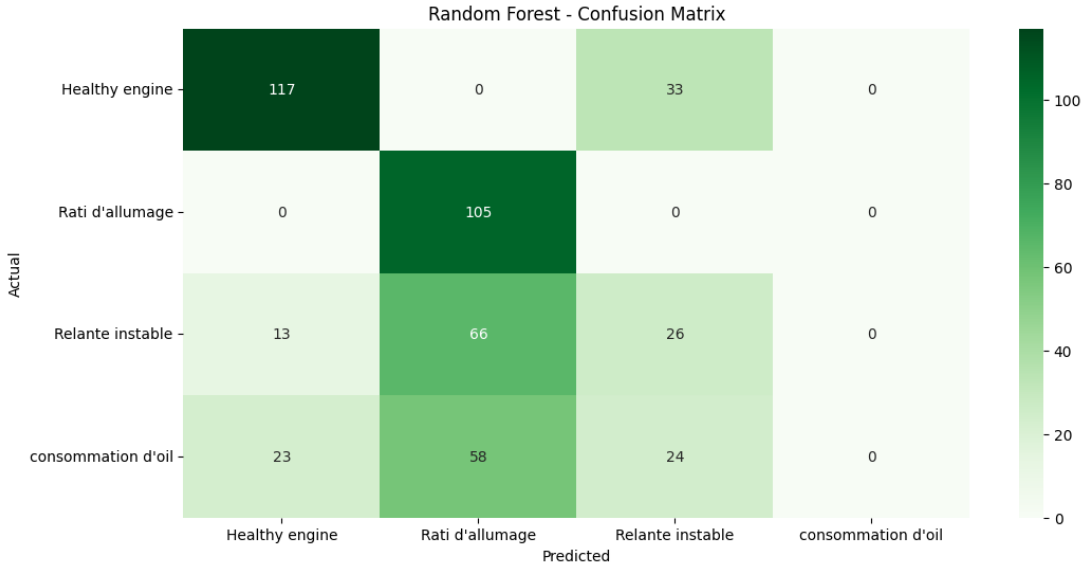

Figure 4.19: Confusion Matrix RF

Classe	Precision	Recall	F1-score	Support
Healthy engine	0.76	0.78	0.77	150
Raté d'allumage	0.46	1.00	0.63	105
Ralenti instable	0.31	0.25	0.28	105
Consommation d'huile	0.00	0.00	0.00	105
Accuracy			**0.53**	465
Macro Average	0.38	0.51	0.42	465
Weighted Average	0.42	0.53	0.45	465

Table 4.14: Classification results for each class (model RF)

Analysis: Random Forest achieved a moderate overall accuracy of 53%. While it classified "Healthy engine" and "Rati d'allumage" with decent performance, its performance for "Relante instable" remained weak, and it completely failed to detect "consommation d'oil." These results indicate the model's limitations in distinguishing less prominent classes.

XGBoost Model

Figure 4.20 displays the confusion matrix for the XGBoost model, while Table 4.15 shows the model's classification performance after excluding the problematic class. Metrics include precision, recall, F1 score, and accuracy.

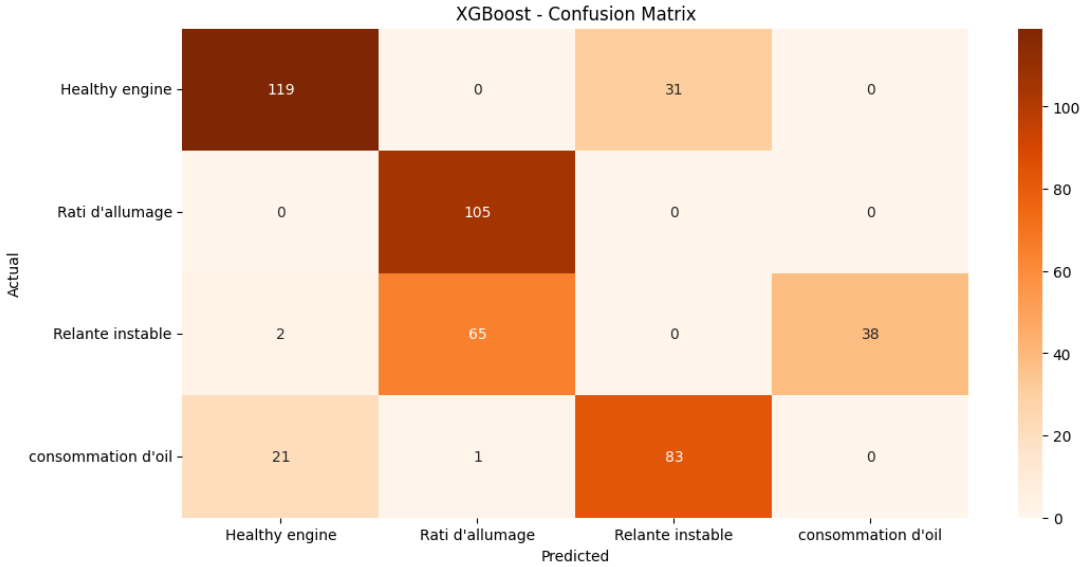

Figure 4.20: Confusion Matrix XGBOOST

Classe	Precision	Recall	F1-score	Support
Healthy engine	0.84	0.79	0.82	150
Raté d'allumage	0.61	1.00	0.76	105
Ralenti instable	0.00	0.00	0.00	105
Consommation d'huile	0.00	0.00	0.00	105
Accuracy			**0.48**	465
Macro Average	0.36	0.45	0.39	465
Weighted Average	0.41	0.48	0.43	465

Table 4.15: Classification results for each class (model XGBOOST)

Analysis: Although XGBoost achieved high precision and recall for "Healthy engine" and "Rati d'allumage," it failed completely for the other two classes. The macro and weighted averages dropped due to the model's inability to generalize across all categories, indicating an imbalance in its prediction strength.

ANN Model

Figure 4.21 shows the confusion matrix of the Artificial Neural Network model, and Table 4.16 provides the model's classification performance for the four fault types. Metrics such as accuracy, precision, recall, and F1 score are presented.

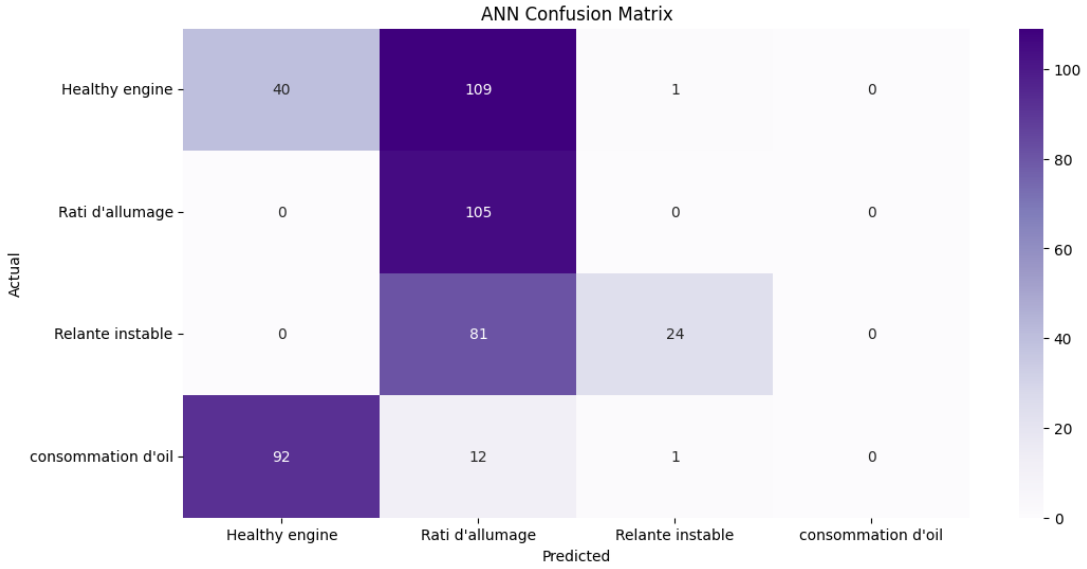

Figure 4.21: Confusion Matrix ANN

Classe	**Precision**	**Recall**	**F1-score**	**Support**
Healthy engine	0.30	0.27	0.28	150
Raté d'allumage	0.34	1.00	0.51	105
Ralenti instable	0.92	0.23	0.37	105
Consommation d'huile	0.00	0.00	0.00	105
Accuracy			**0.36**	465
Macro Average	0.39	0.37	0.29	465
Weighted Average	0.38	0.36	0.29	465

Table 4.16: Classification results for each class (model ANN)

Figure 4.22 represents the accuracy and loss curve of the ANN model.

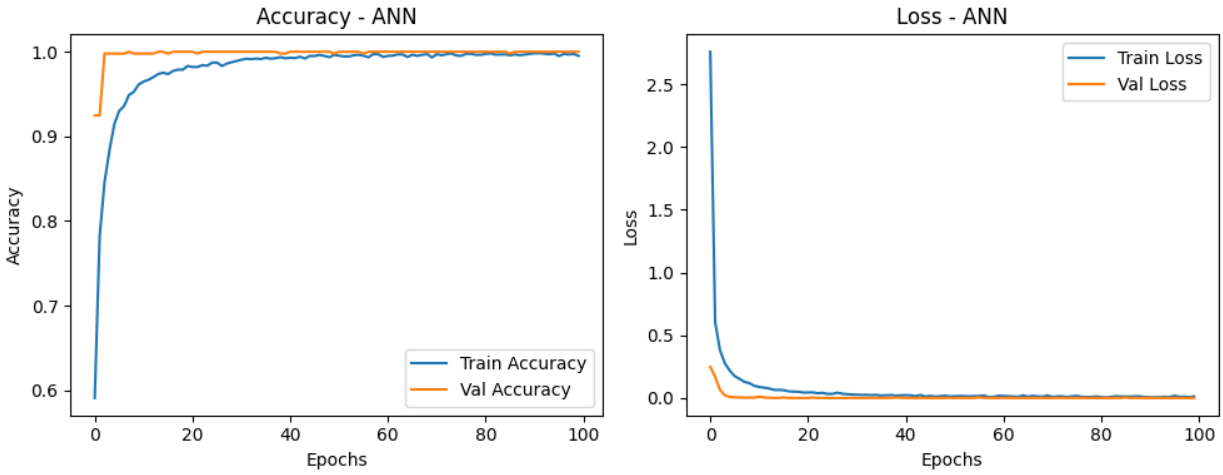

Figure 4.22: ccuracy and Loss (ANN)

Analysis: With an accuracy of 36%, the ANN model showed weak performance overall. It was only effective in classifying "Rati d'allumage" with high recall, while the other classes, especially "consommation d'oil," were not detected. This reflects poor generalization and suggests insufficient learning or capacity.

LSTM Model

Figure 4.23 presents the confusion matrix of the LSTM model, and Table 4.17 summarizes its performance metrics for the four fault categories, including accuracy, precision, recall, and F1 score.

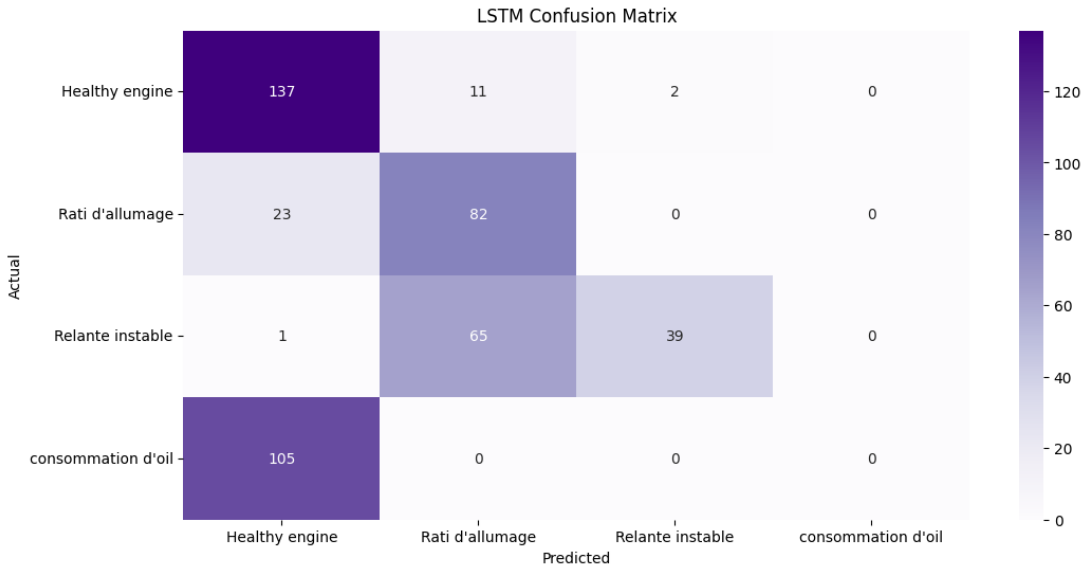

Figure 4.23: Confusion Matrix LSTM

Classe	Precision	Recall	F1-score	Support
Healthy engine	0.52	0.91	0.66	150
Raté d'allumage	0.52	0.78	0.62	105
Ralenti instable	0.95	0.37	0.53	105
Consommation d'huile	0.00	0.00	0.00	105
Accuracy			**0.55**	465
Macro Average	0.50	0.52	0.45	465
Weighted Average	0.50	0.55	0.47	465

Table 4.17: Classification results for each class (model LSTM)

Figure 4.24 represents the accuracy and loss curve of the ANN model.

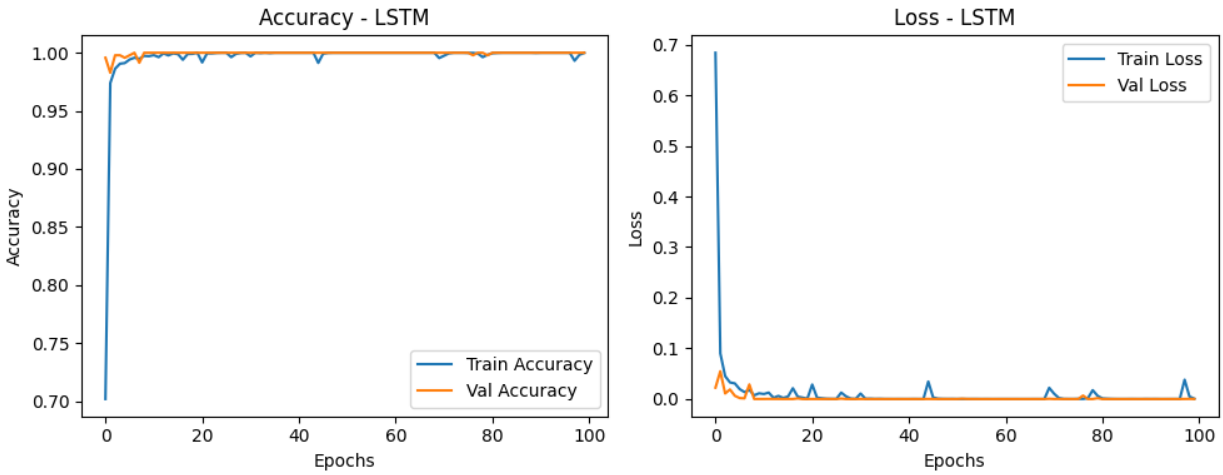

Figure 4.24: ccuracy and Loss (LSTM)

Analysis: The LSTM model reached 55% accuracy and showed relatively balanced performance across three of the four classes. It performed particularly well on "Healthy engine" and "Relante instable," indicating its potential to capture sequential patterns. However, its failure to detect "consommation d'oil" suggests further refinement is needed.

CNN Model

Figure 4.25 displays the confusion matrix of the CNN model, while Table 4.18 highlights the classification metrics after the exclusion of the problematic class. The key performance indicators include accuracy, precision, recall, and F1 score.

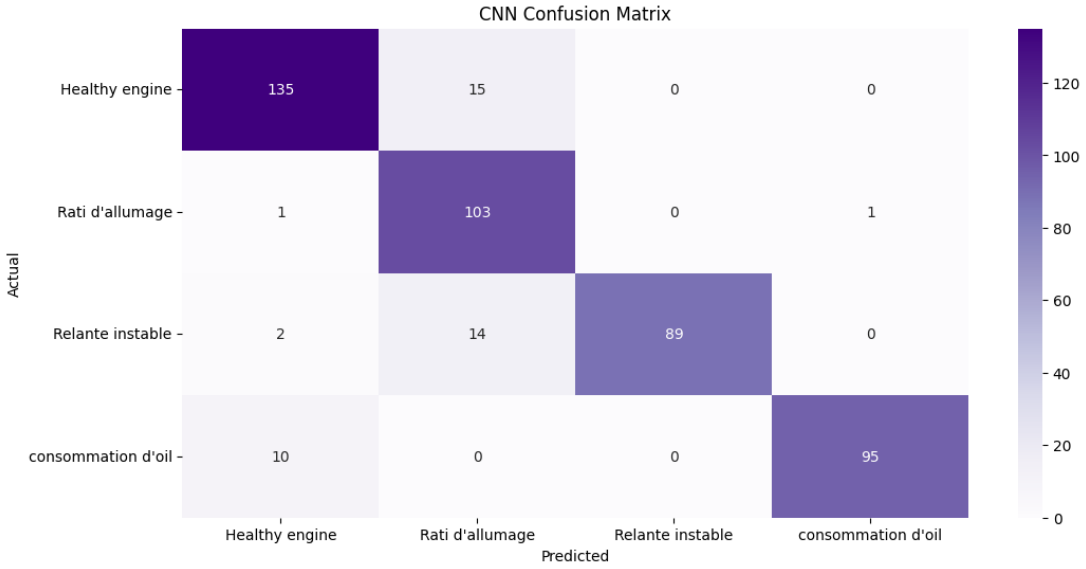

Figure 4.25: Confusion Matrix CNN

Classe	**Precision**	**Recall**	**F1-score**	**Support**
Healthy engine	0.91	0.90	0.91	150
Raté d'allumage	0.78	0.98	0.87	105
Ralenti instable	1.00	0.85	0.92	105
Consommation d'huile	0.99	0.90	0.95	105
Accuracy			**0.91**	465
Macro Average	0.92	0.91	0.91	465
Weighted Average	0.92	0.91	0.91	465

Table 4.18: Classification results for each class (model CNN)

Figure 4.26 represents the accuracy and loss curve of the ANN model.

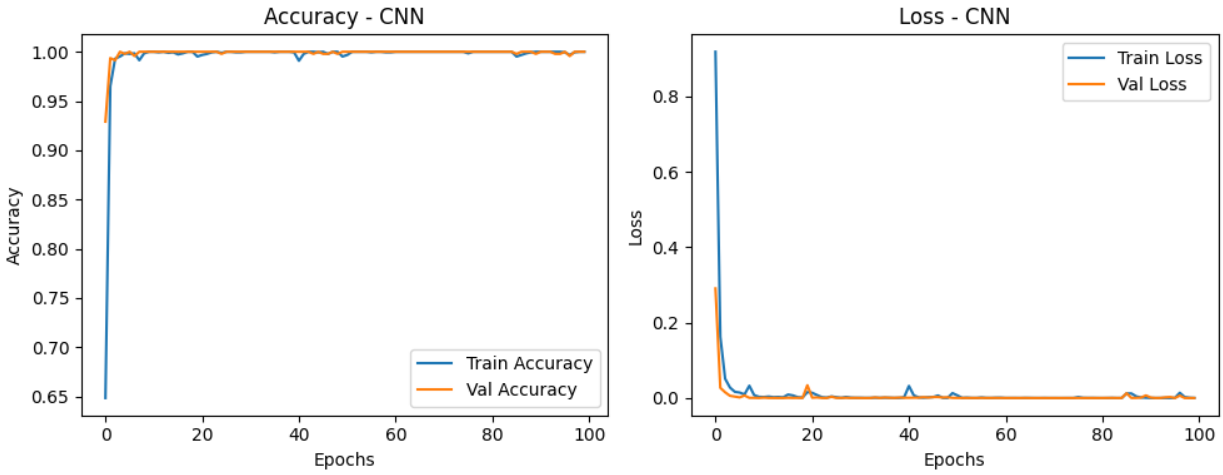

Figure 4.26: ccuracy and Loss (CNN)

Analysis: The CNN model achieved excellent results, with 91% accuracy and strong performance across all four categories. It showed high precision and recall values, indicating robust learning of distinctive audio patterns. This suggests CNN is well-suited for multi-class fault classification when clean and balanced data is used.

Figure 4.27 shows the performance of the models (CNN LSTM ANN) in terms of accuracy and loss. Figure 4.28 shows the performance of all models in terms of accuracy and loss.

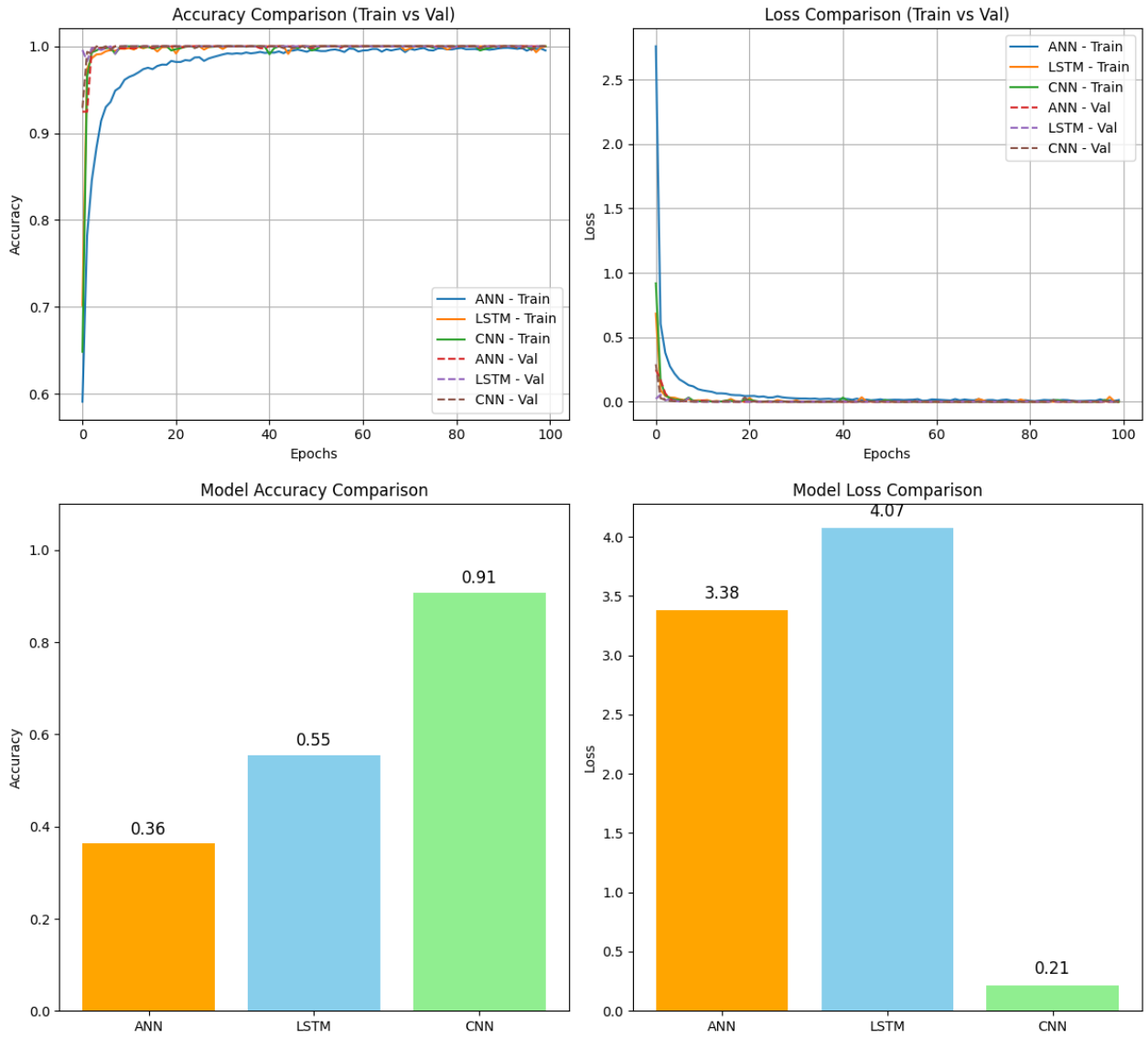

Figure 4.27: Accuracy and Loss

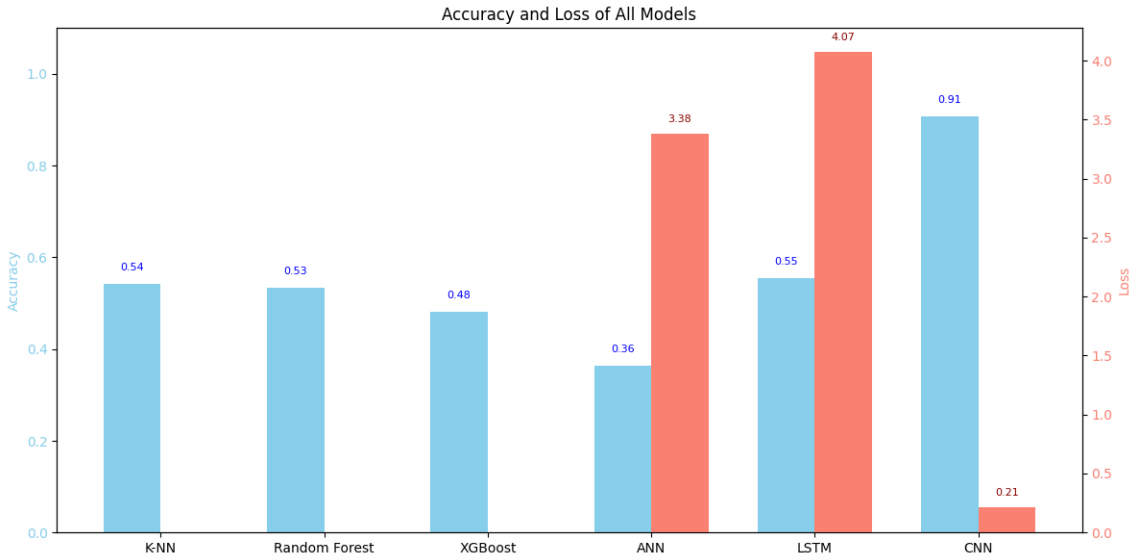

Figure 4.28: Accuracy and Loss all models

Table 4.19 shows the performance of the models in classifying engine faults into four categories after removing one of the categories. Figure 4.29 also shows the performance of all models in terms of (precision, recall, and F1 score).

Figure 4.30 also shows the performance of all models on the test set.

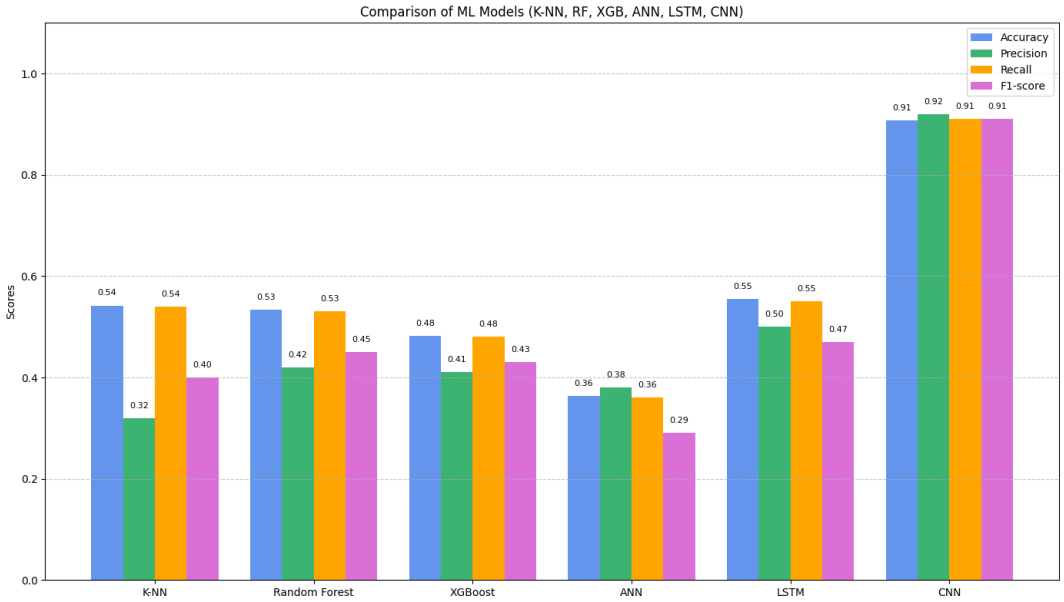

Figure 4.29: Comparison of Evaluation Metrics by Model

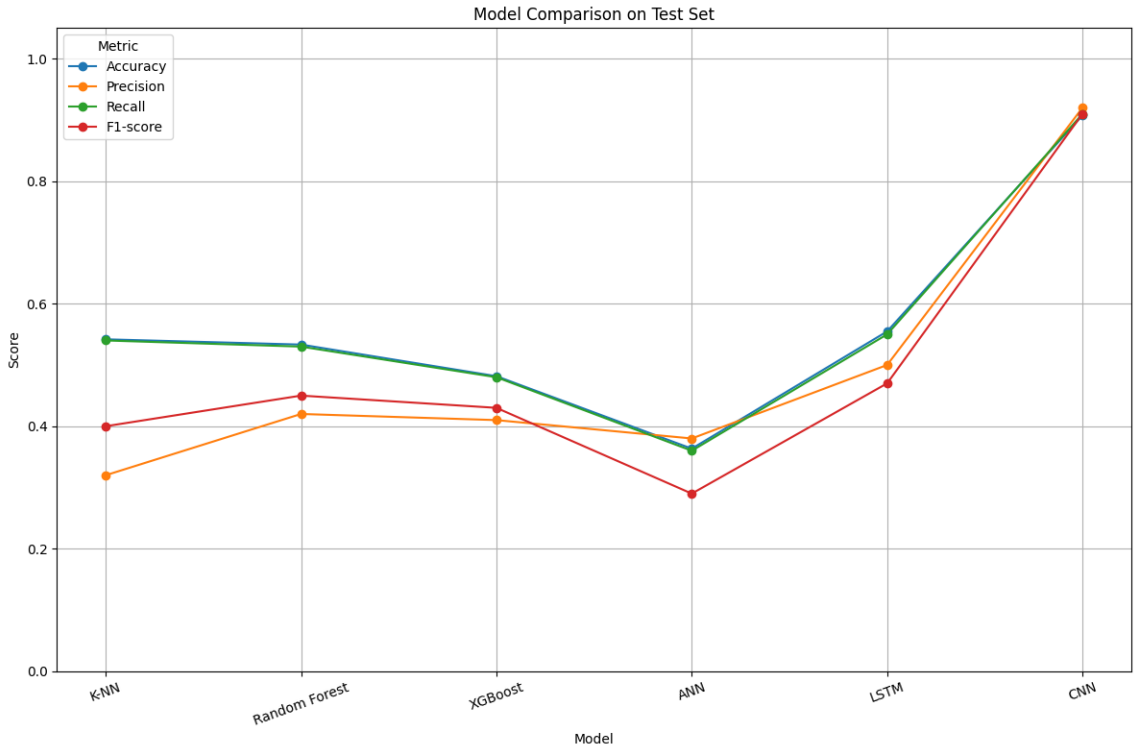

Figure 4.30: Model Comparison on Test set

Model	Accuracy	Loss	Observations
K-NN	54%	N/A	High recall for *Raté d'allumage* (0.97), but extremely poor performance on minority classes (*Relante instable* and *Consommation d'huile*); suffered from imbalance sensitivity.
Random Forest	53%	N/A	Strong performance on *Raté d'allumage* (recall: 1.00), moderate on *Healthy engine*, but failed completely on *Consommation d'huile*; lacks generalization.
XGBoost	48%	N/A	Best overall classical model in terms of precision for *Healthy engine* and *Raté d'allumage*; failed to detect any samples from minority classes.
ANN	36%	3.3786	Moderate recall for *Raté d'allumage* (1.00), but poor performance elsewhere, especially on *Consommation d'huile*; underfitting likely due to insufficient model depth or data noise.
LSTM	55%	4.0713	Good on *Healthy engine* and *Raté d'allumage*; surprisingly high precision on *Relante instable* but no detection of *Consommation d'huile*; temporal patterns partially captured.
CNN	91%	0.2107	Significantly outperformed all models; high precision and recall across all classes including *Consommation d'huile*; robust spatial feature extraction; likely benefited from image-based spectrogram input.

Table 4.19: Comparison of model performance based on accuracy, loss, and qualitative observations

Analysis and Discussion

The comparative analysis of the six models highlights the superiority of deep learning techniques—particularly the Convolutional Neural Network (CNN)—for the task of sound-based engine fault classification. While traditional models such as K-NN, Random Forest, and XGBoost showed relatively moderate performance on majority classes like Healthy engine and

Raté d'allumage, they systematically failed to recognize minority fault classes (Consommation d'huile, Relante instable). This confirms their sensitivity to class imbalance and limited ability to generalize across subtle audio features.

Artificial Neural Network (ANN) and Long Short-Term Memory (LSTM) models, though more adaptable, underperformed due to likely underfitting or suboptimal temporal feature extraction. LSTM showed some strength in identifying temporal patterns for recurrent faults but remained inconsistent.

In stark contrast, the CNN model achieved an impressive accuracy of 91%, maintaining high precision and recall across all four classes, including the previously challenging Consommation d'huile. Its ability to leverage spatial representations of sound through spectrogram images proved to be a critical factor in capturing discriminative features. Furthermore, the low loss value of 0.2107 confirms strong model convergence.

Overall, CNN not only offers the best quantitative performance but also demonstrates robust generalization capabilities, making it the most suitable choice for deployment in real-time engine diagnostics systems.

4.3.3 Performance Comparison of CNN Model Before and After Removing the Problematic Class

The primary objective of the first experiment 4.3.1 was to evaluate the models' ability to classify all five engine fault categories, including the problematic class "Bougie d'allumage." For example, the CNN model exhibited inconsistent performance, particularly struggling to distinguish this class from others due to its ambiguous characteristics. To address this issue, a second experiment 4.3.2 was conducted using the same model but after removing the problematic class. The goal was to observe potential improvements in classification accuracy and generalization capability.

After excluding the "Bougie d'allumage" class, the CNN model demonstrated a significant improvement. The test accuracy increased from 66.3% to 91%, with notable enhancements in all key metrics including precision, recall, and F1-score. This highlights that removing overlapping or ambiguous classes can lead to more stable and accurate classification performance.

Misclassification of engine faults can lead to incorrect diagnoses, resulting in unnecessary repairs, increased maintenance costs, and even safety risks if critical issues are overlooked. Therefore, improving classification accuracy is essential for developing reliable diagnostic systems, especially in real-time or embedded automotive applications.

Metric	**Before Removing "Bougie d'allumage"**	**After Removing "Bougie d'allumage"**
Accuracy	66.3%	**91.0%**
Loss	6.90	**0.21**
Precision (avg)	0.67	**0.92**
Recall (avg)	0.66	**0.91**
F1-Score (avg)	0.66	**0.91**
Observations	Low performance on "Bougie d'allumage" significantly impacted overall accuracy	Strong and balanced performance across all remaining classes

Table 4.20: Impact of removing the "Bougie d'allumage" class on CNN model performance

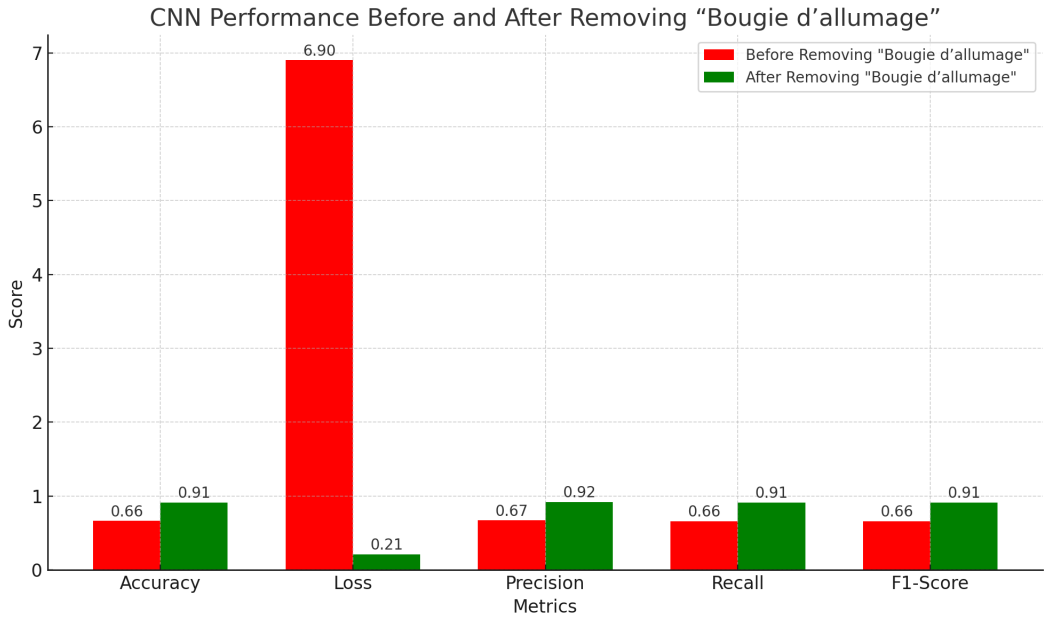

Figure 4.31: CNN Model Comparison

This Figure 4.31 shows a close comparison of the CNN model's performance before and after removing the "Bougie d'allumage" class. We observe significant improvements across all metrics, particularly in accuracy, which increased from 66.3% to 91%, and a significant decrease in loss, from 6.90 to 0.21, indicating that the model learned more effectively after removing the problematic class.

4.4 Binary Classification of Engine Sounds

In addition to the multi-class experiments 4.3 a binary classification task was conducted separately for diesel and gasoline engines. The objective was to differentiate between healthy and damaged engine conditions, which simplifies the classification problem and aligns with real-world needs for rapid diagnostic systems.

4.4.1 Dataset Overview

- **Diesel Engines:**
  - Training/Validation Set: 625 healthy, 625 damaged.
  - Test Set: 125 healthy, 125 damaged.
- Gasoline Engines:
  - Training/Validation Set: 750 healthy, 750 damaged.
  - Test Set: 150 healthy, 150 damaged.

Each dataset was balanced to avoid bias and ensure a fair evaluation of model performance.

4.4.2 Results and performance comparison of models

- **Diesel Engines:**

  Figure 4.32 displays the confusion matrices for Logistic Regression, K-Nearest Neighbors, Support Vector Machine, Random Forest, and Neural Network (MLP) on the binary engine classification task. Figure 4.33 illustrates the CNN model's confusion matrix and its training accuracy and loss curves, indicating superior performance and robust learning from spectrogram features.

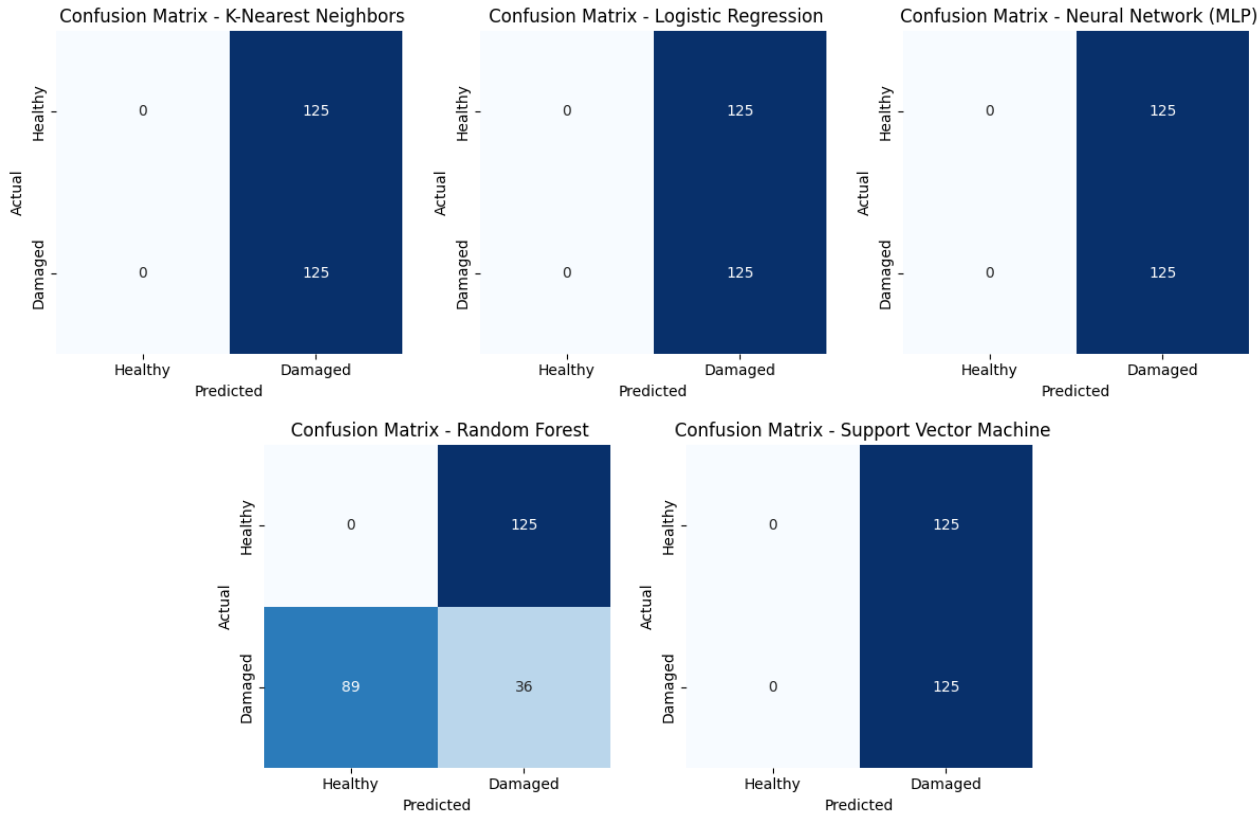

Figure 4.32: Diesel Engines Confusion Matrix

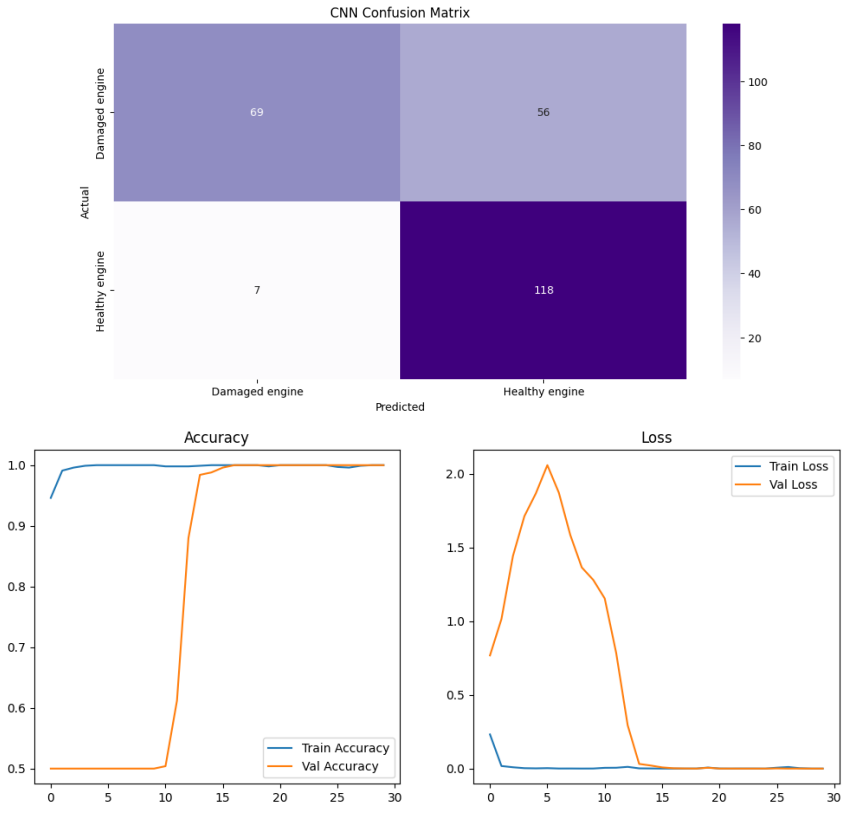

Figure 4.33: Confusion Matrix and Accuracy CNN

- **Gasoline Engines:**

Figure 4.34 presents the confusion matrices for five traditional models: Logistic Regression, K-Nearest Neighbors, Support Vector Machine, Random Forest, and a Multi-Layer Perceptron (MLP), applied to the binary classification between healthy and damaged engines. In contrast, Figure 4.35 shows the CNN model's confusion matrix along with its accuracy and loss curves, highlighting its superior ability to learn from spectrogram representations.

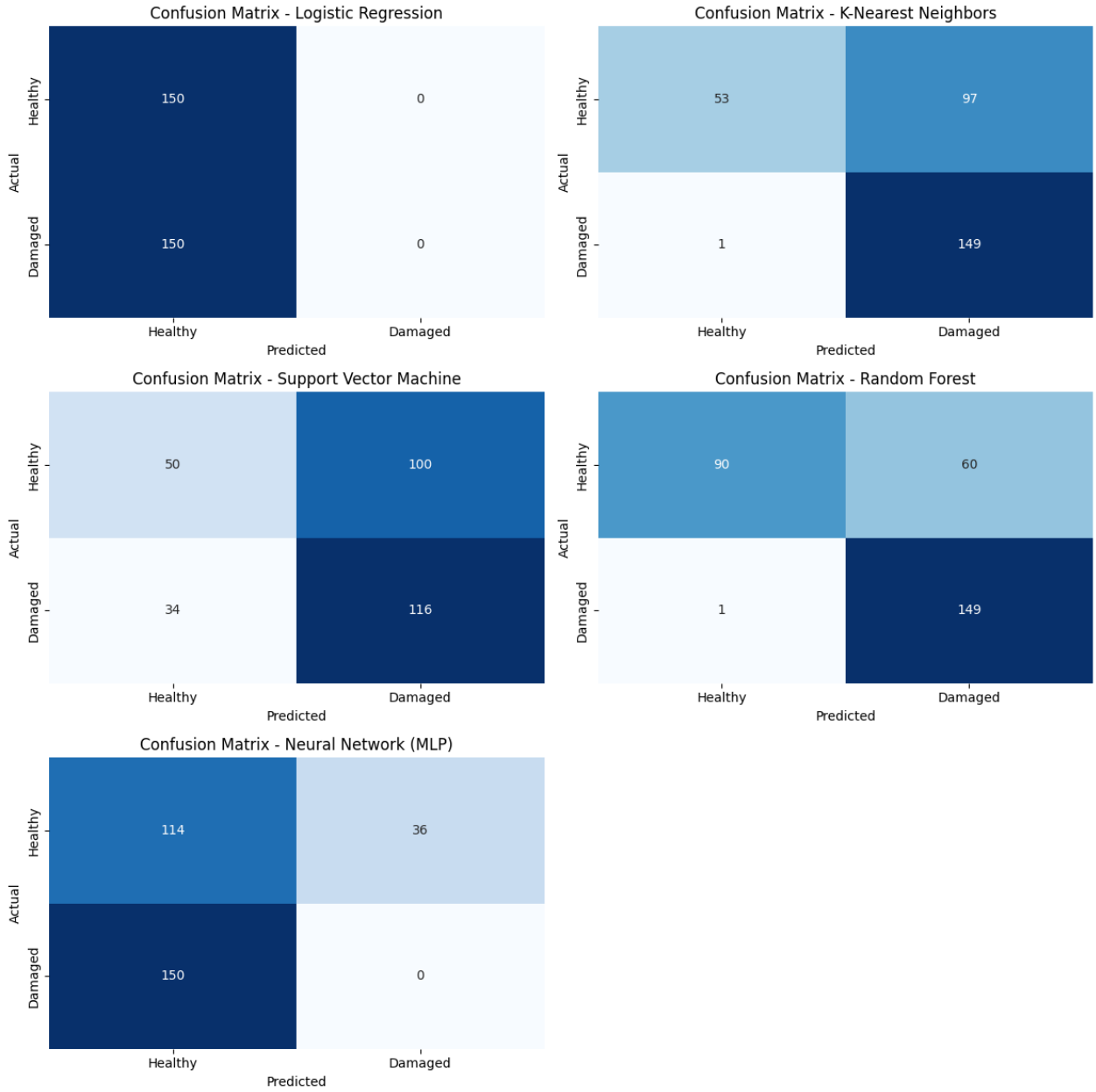

Figure 4.34: Gasoline Engines Confusion Matrix

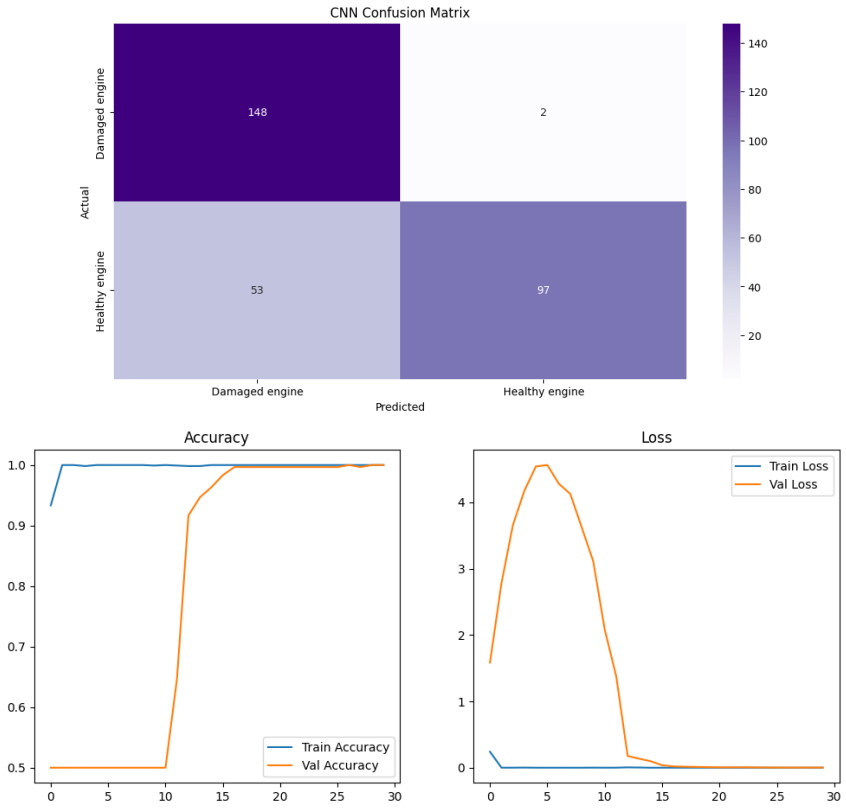

Figure 4.35: Confusion Matrix and Accuracy CNN

4.4.3 Performance on Diesel Engine Dataset

Model	Accuracy (Validation)	Accuracy (Test)	Precision (Test)	Recall (Test)	F1-score (Test)
Logistic Regression	100%	50.0%	0.5000	1.000	0.6667
K-Nearest Neighbors	100%	50.0%	0.5000	1.000	0.6667
Support Vector Machine	98.8%	50.0%	0.5000	1.000	0.6667
Random Forest	99.6%	14.4%	0.2236	0.288	0.2517
Neural Network (MLP)	100%	50.0%	0.5000	1.000	0.6667
CNN	—	**74.8%**	0.91 (Damaged)	0.55 (Damaged)	0.69 (Damaged)

Table 4.21: Performance comparison between models (Diesel)

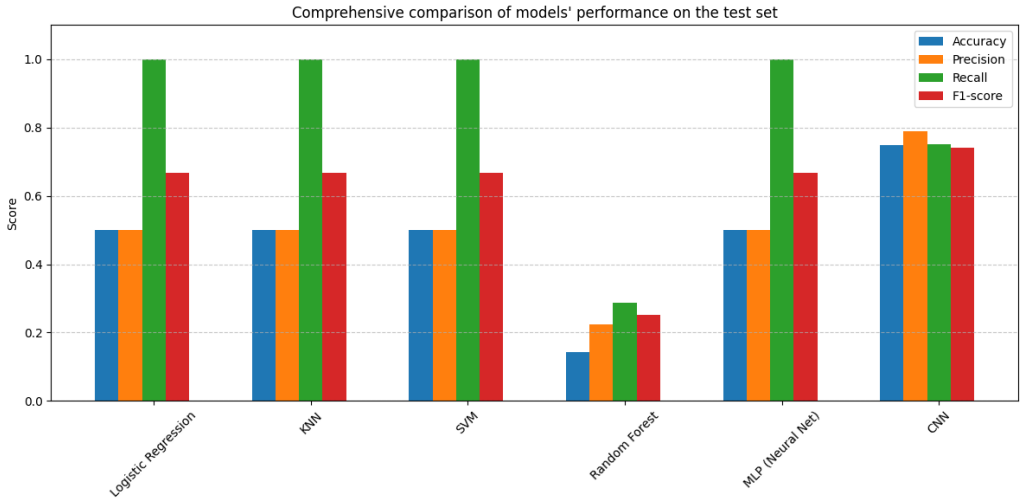

Figure 4.36: Comparison of Evaluation Metrics by Model (Diesel)

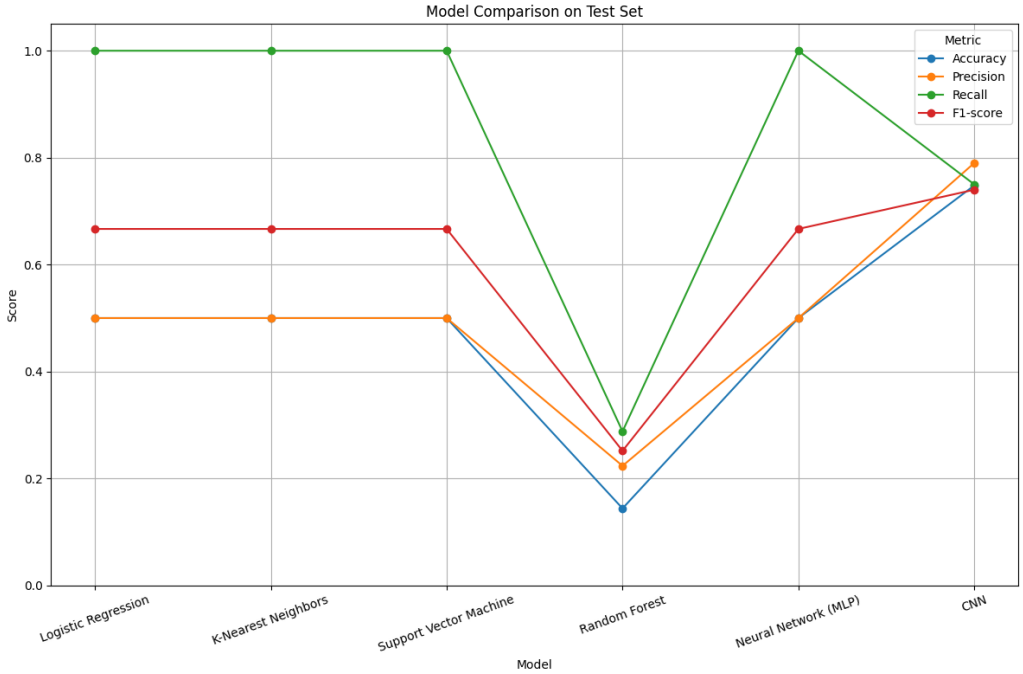

Figure 4.37: Model Comparison on Test set (Diesel)

Table 4.21 shows that CNN outperformed all classical models on the diesel test set, demonstrating superior generalization. As illustrated in Figure 4.37 and Figure 4.36 CNN achieved the highest accuracy and strong evaluation metrics. However, the confusion matrix reveals some false positives for the healthy class, suggesting a need for improved precision in that category.

4.4.4 Performance on Gasoline Engine Dataset

Model	Accuracy (Validation)	Accuracy (Test)	Precision (Test)	Recall (Test)	F1-score (Test)
Logistic Regression	100%	50.0%	0.0000	0.000	0.0000
K-Nearest Neighbors	100%	67.3%	0.6057	0.9933	0.7525
Support Vector Machine	100%	55.3%	0.5370	0.7733	0.6339
Random Forest	100%	**79.7%**	0.7129	0.9933	0.8301
Neural Network (MLP)	100%	38.0%	0.0000	0.000	0.0000
CNN	—	**81.7%**	0.74 (Damaged)	0.99 (Damaged)	0.84 (Damaged)

Table 4.22: Performance comparison between models (Gasoline)

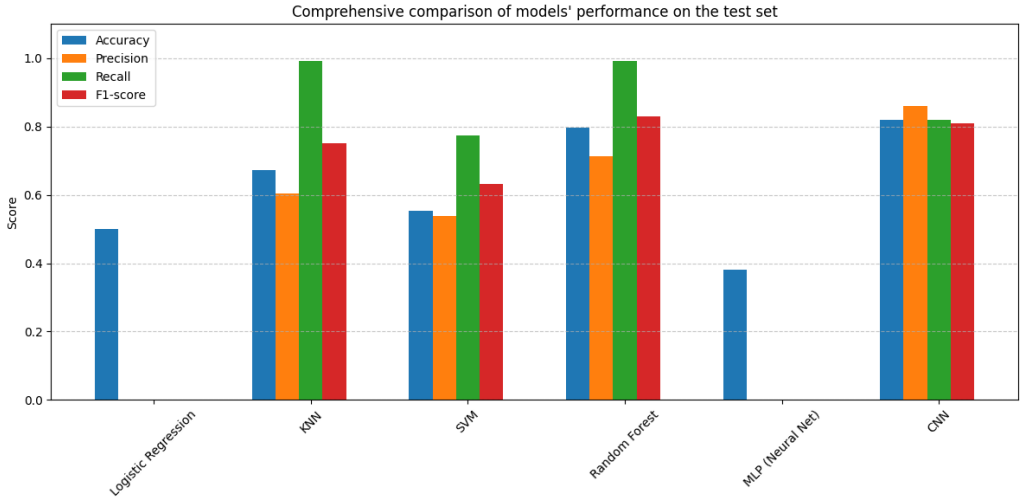

Figure 4.38: Comparison of Evaluation Metrics by Model (Gasoline)

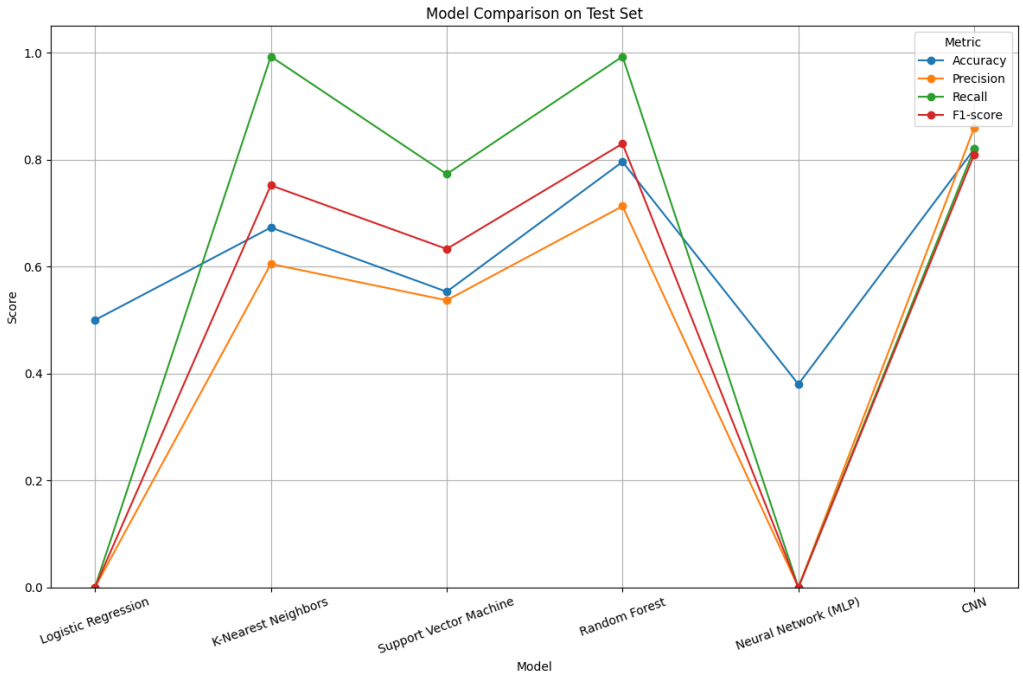

Figure 4.39: Model Comparison on Test set (Gasoline)

Table 4.22 shows that both CNN and Random Forest achieved the best performance on the gasoline test set. As illustrated in Figure 4.39 and Figure 4.38 CNN demonstrated high sensitivity in detecting damaged engines, outperforming classical models in most evaluation metrics. However, it showed reduced precision when identifying healthy engines, indicating a tendency toward false positives in that class.

4.4.5 Key Observations

- Classical models showed excellent performance on the validation set but generally failed to generalize to unseen test data, especially for diesel engines.
- CNN demonstrated stronger robustness and higher generalization capabilities, especially in imbalanced or noisy conditions.
- Test results reinforce the importance of deep models in sound-based fault detection, particularly when data distribution and acoustic features are complex.

4.4.6 Conclusion

Binary classification significantly simplified the problem space and led to higher accuracies, especially for CNN-based models. This supports their integration into real-time diagnostic systems for embedded automotive applications.

4.5 Comparison Between Multi-Class and Binary Classification

The experimental results reveal significant differences between multi-class and binary classification approaches:

- Multi-class classification 4.3 allows the system to distinguish between various types of faults. However, it introduces complexity, especially when different classes have overlapping acoustic features. Misclassification is more likely, particularly when the dataset is limited or unbalanced.
- Binary classification 4.4, on the other hand, simplifies the problem into a healthy vs. faulty decision. This setting often yields higher accuracy and better generalization, as demonstrated by the CNN models, particularly for diesel and gasoline engines.

Binary models achieved higher performance due to reduced class confusion and clearer decision boundaries. However, they lack the granularity needed for identifying specific faults, which limits their diagnostic capability.

4.6 Model Size Optimization for Embedded Deployment

Since one of the project goals is to integrate the trained models into a mobile application, model size and memory efficiency are critical.

The table 4.23 below compares the sizes of the CNN models before and after quantization:

Model	**Type**	**Size**
cnn_model_multiclass.keras	Multi-class CNN	84.64 MB
cnn_model_multiclass_quant.tflite	Quantized CNN	7.06 MB
cnn_model_diesel.keras	Binary CNN (Diesel)	37.88 MB
cnn_model_diesel.tflite	Quantized CNN	3.16 MB
cnn_model_essence_binary.keras	Binary CNN (Essence)	37.88 MB
cnn_model_essence_binary.tflite	Quantized CNN	3.16 MB

Table 4.23: Size comparison of CNN models before and after quantization

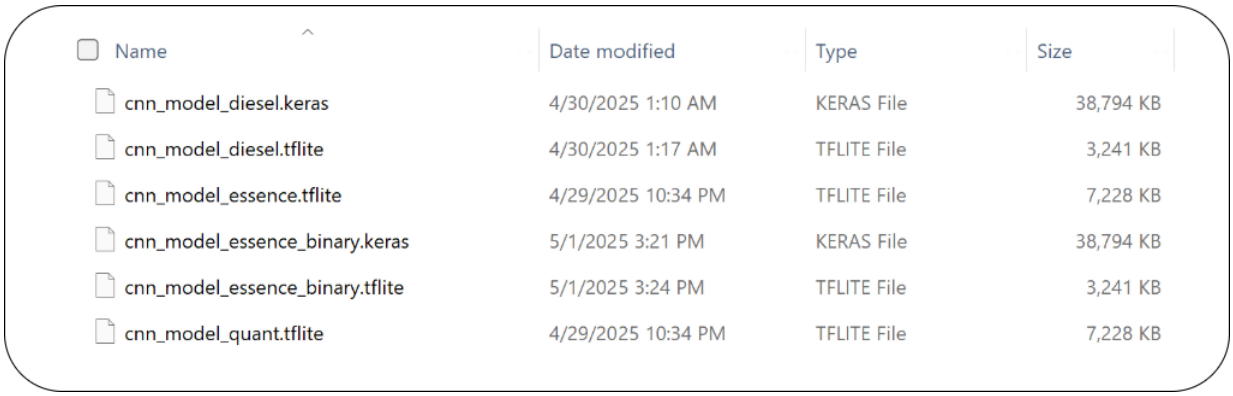

Figure 4.40: Models

Quantization significantly reduces model size—up to 90% in some cases—making the models suitable for on-device inference with minimal performance loss. These optimizations are essential for real-time fault detection in mobile or embedded environments.

4.7 Conclusion

This chapter highlighted the practical and experimental aspects of engine sound classification. A key takeaway is that misclassification can lead to severe consequences, such as unnecessary repairs, delayed interventions, or failure to detect critical issues on time. This was clearly evident in our experiments—before and after removing ambiguous classes from the dataset.

Moreover, sound-based diagnosis has inherent limitations. In many cases, multiple faults can produce acoustically similar patterns, making it difficult—even for advanced models—to determine the exact nature of the problem based solely on audio. This underscores the importance of complementing sound analysis with other sensor data (e.g., vibration, temperature) for more reliable diagnosis.

To mitigate these challenges and ensure practical usability, the trained models have been optimized and embedded into a mobile application designed for real-time fault detection. This integration not only enables users to record and analyze engine sounds directly from their smartphones, but also provides a user-friendly interface to guide the diagnostic process. Combining model predictions with interactive features and visual feedback enhances both the accuracy and the user experience. Therefore, this integration bridges the gap between theoretical machine learning models and real-world automotive applications.

System Architecture and Implementation

CHAPTER 5

SYSTEM ARCHITECTURE AND IMPLEMENTATION

5.1 Introduction

This chapter presents the overall system architecture and the development tools used to implement the intelligent engine fault diagnosis application. It describes the backend infrastructure, mobile application design, and the communication protocols used for efficient interaction between system components. Furthermore, various Unified Modeling Language (UML) diagrams are provided to illustrate the functional and structural perspectives of the system. Finally, the development environment, software stack, and deployment processes are described to support the reproducibility of the implementation.

5.2 Architecture

This section presents the core architectural design of the intelligent fault detection system, including multiple UML diagrams to capture its dynamic and structural behavior.

5.2.1 Use Case Diagram

he use case diagram models the high-level interactions between different actors (e.g., user, backend server, Firebase) and the system. It helps identify the main system functionalities such as user authentication, audio recording, fault analysis, and result retrieval.

Figure 5.1 illustrates how the user interacts with system components through specific interfaces.

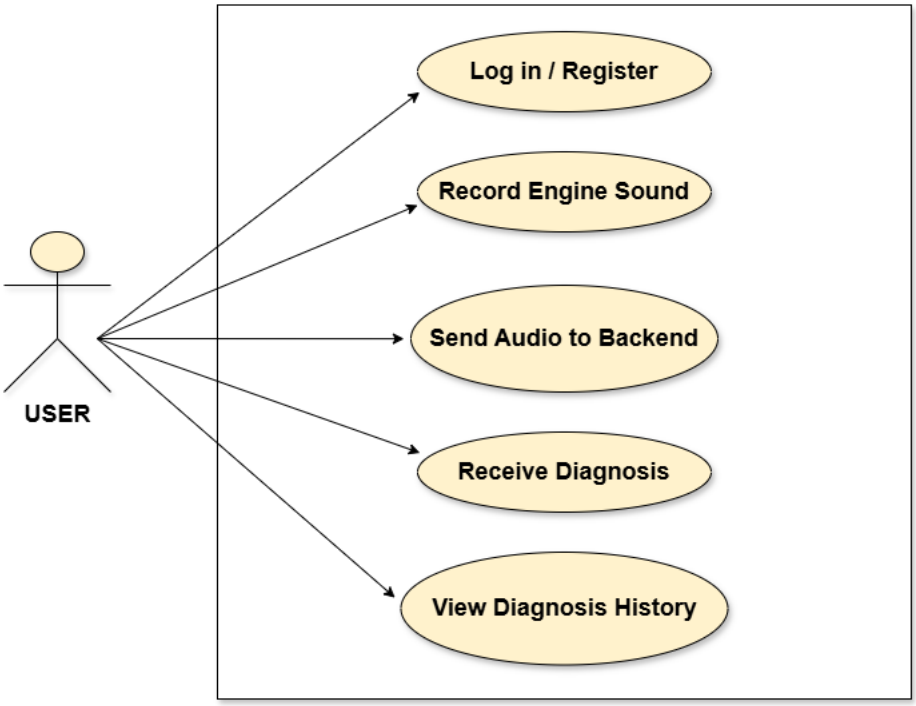

Figure 5.1: Use Case Diagram of the Engine Diagnosis System

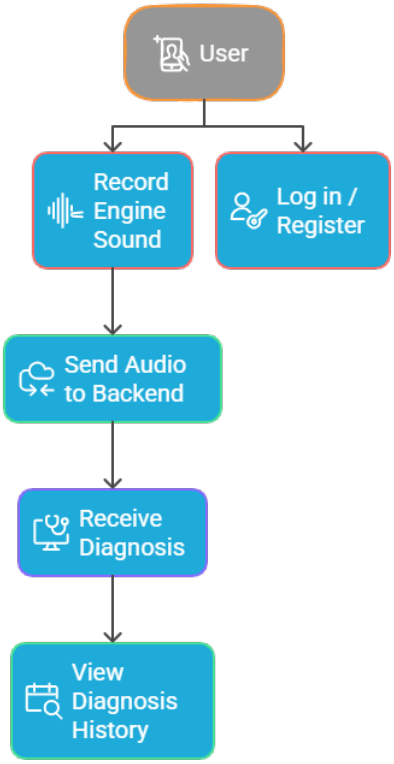

Figure 5.2: Engine Sound Diagnosis Process

5.2.2 Sequence Diagram

The sequence diagram captures the order of message exchanges between the components of the system during a typical diagnostic session. It outlines how audio data is processed from recording to model inference and response delivery.

Figure 5.3 shows the sequence of processes that occur from the moment the user enters the voice to the display of the diagnostic result.

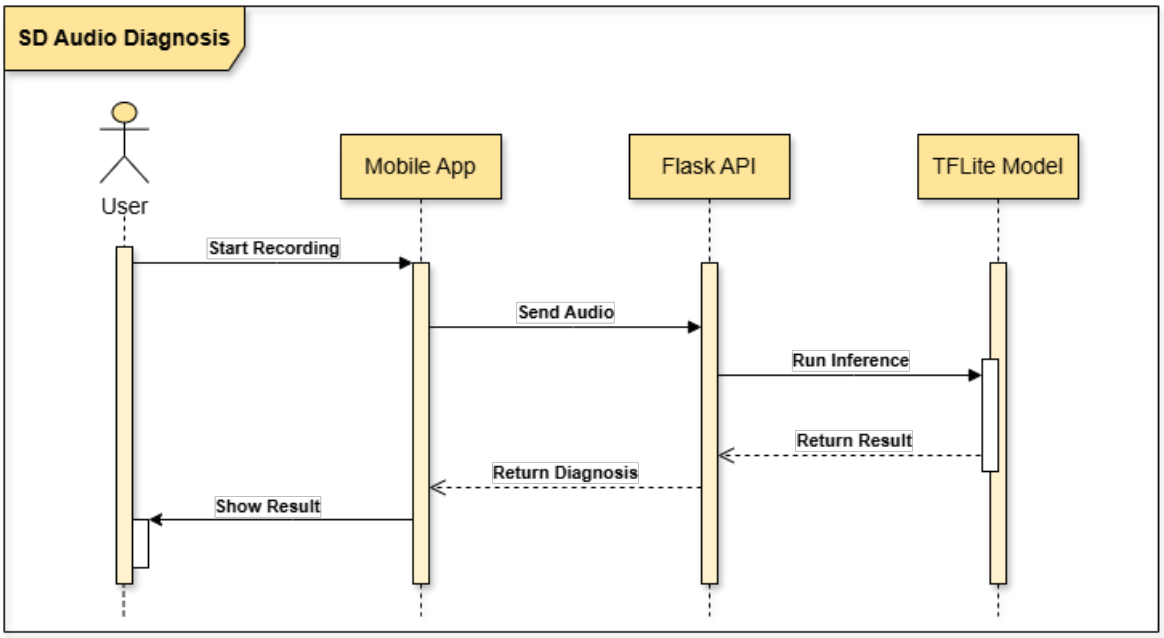

Figure 5.3: Sequence Diagram for the Sound-Based Diagnosis Process

5.2.3 Component Diagram

The component diagram shows the software components of the system (e.g., mobile app, REST API, TFLite models, Firebase services) and how they are connected. This provides a clear view of system modularity and deployment structure.

Figure 5.4 illustrates the system's structural architecture and shows how the different software modules interact with each other.

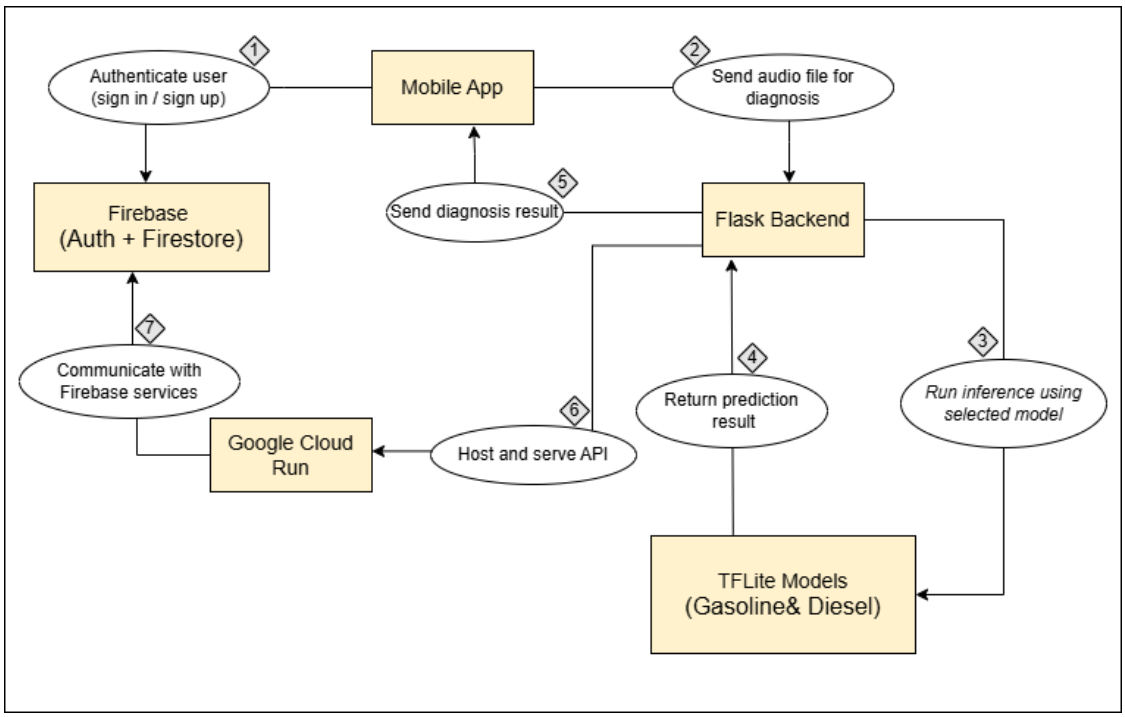

Figure 5.4: Component Diagram of the Overall System Architecture

5.3 Backend and Deployment

Flask

Flask is a micro web framework for Python. It was used to build the REST API that handles the interaction between the mobile app and the classification models [20].

Before deployment, the Flask API was thoroughly tested using Postman A as shown in Figure 5.6 to validate its response accuracy and ensure smooth communication with both the gasoline and diesel TensorFlow Lite models 5.3. These tests confirmed the readiness and robustness of the backend services under realistic usage scenarios.

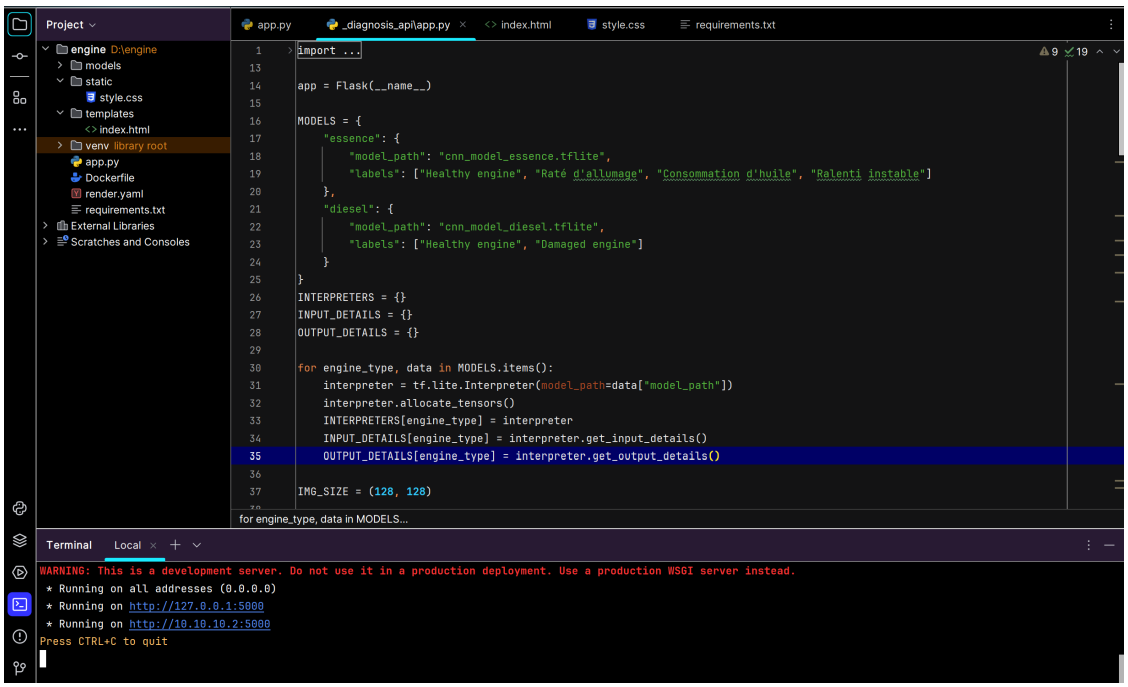

Figure 5.5: Code python (Flask)

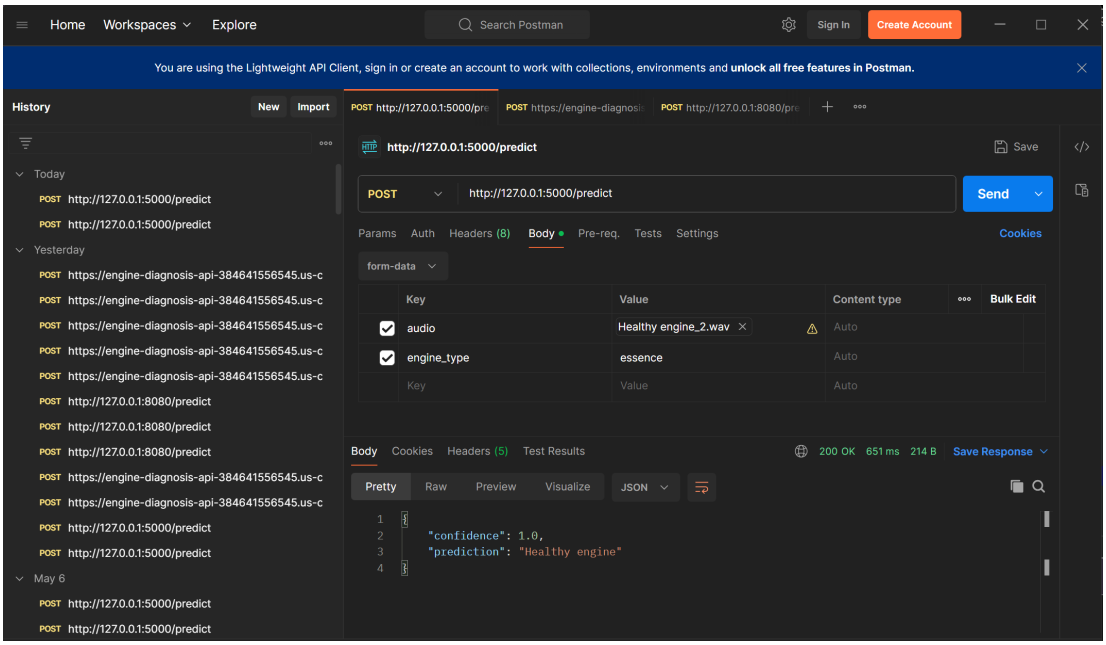

Figure 5.6: Verified by postman

TensorFlow Lite

TensorFlow Lite is a lightweight solution for deploying TensorFlow models on mobile and embedded devices. It supports model quantization for size and performance optimization [21].

Google Cloud Run

Google Cloud Run is a fully managed platform that automatically deploys and scales containerized applications. It was chosen to host the backend API securely and reliably [22].

Figures 5.7 and 5.8 illustrate the successful deployment of the project on Google Cloud Run using both Gunicorn A and Docker A. This deployment method enables the application to run in a scalable, serverless environment that abstracts infrastructure management. As a result, a dedicated and publicly accessible URL 5.11 was generated, which will be integrated into the mobile application to enable real-time communication with the machine learning model hosted on the cloud.

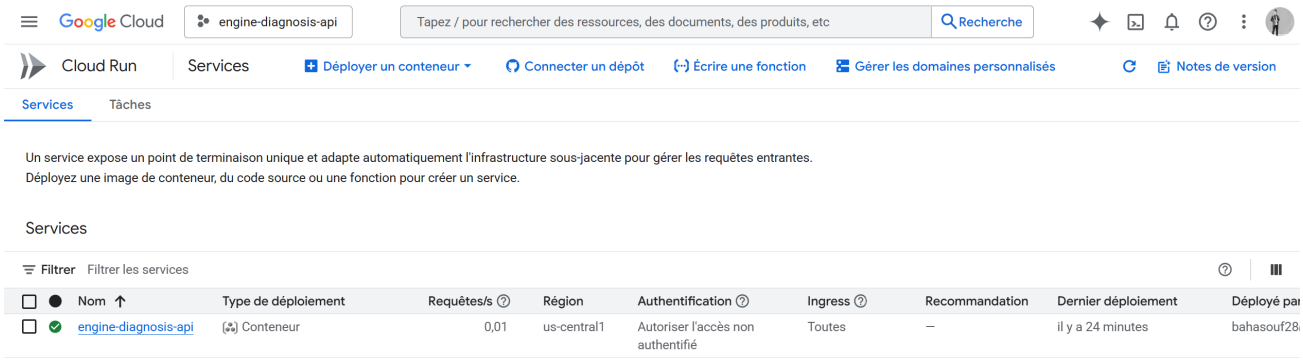

Figure 5.7: Cloud Run

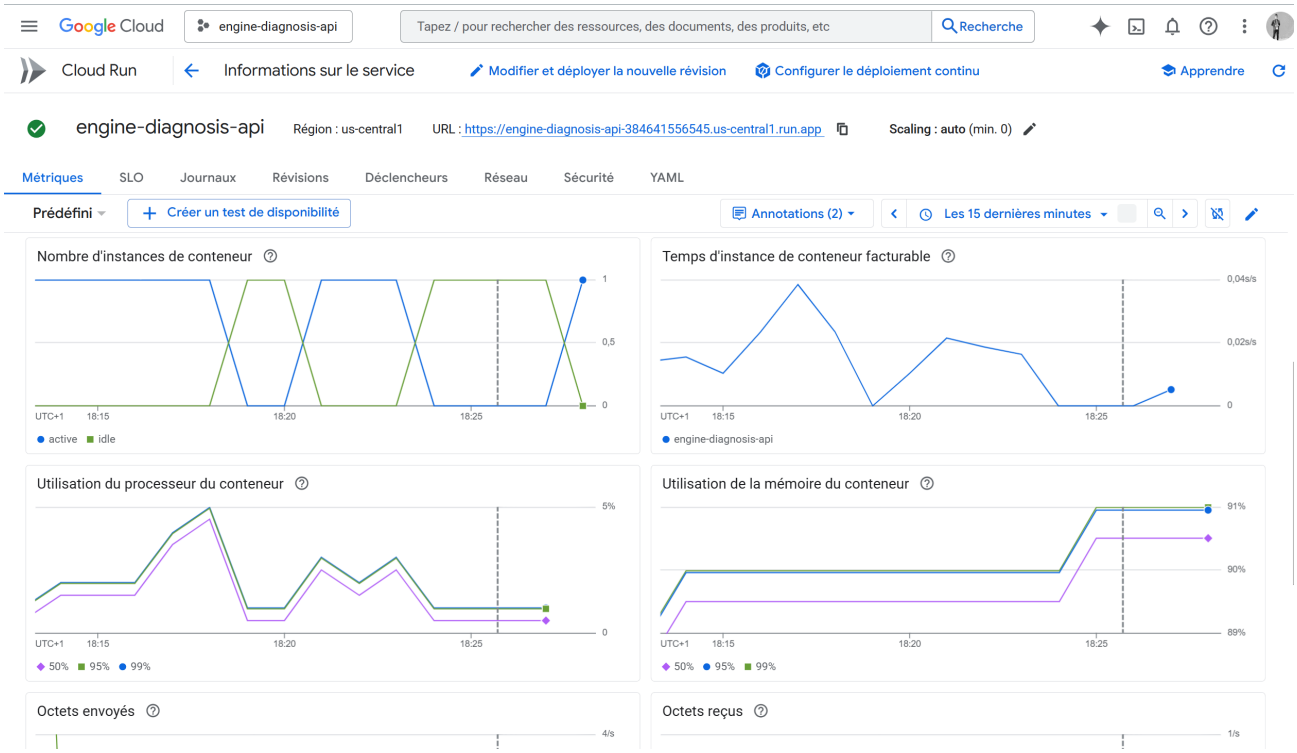

Figure 5.8: Cloud Run

5.4 Database and Authentication

Firebase Authentication

Firebase Authentication provides secure and easy-to-integrate user login and identity management. It supports email, password, and third-party authentication.

Figure 5.9 presents the number of users who have successfully registered within the application and are able to access it through the Sign In page 5.14. This functionality is a core component of the user authentication system, which ensures that only authorized users can log in and interact with the app's features. The registration and login process is managed through Firebase Authentication, providing a secure and scalable identity management solution.

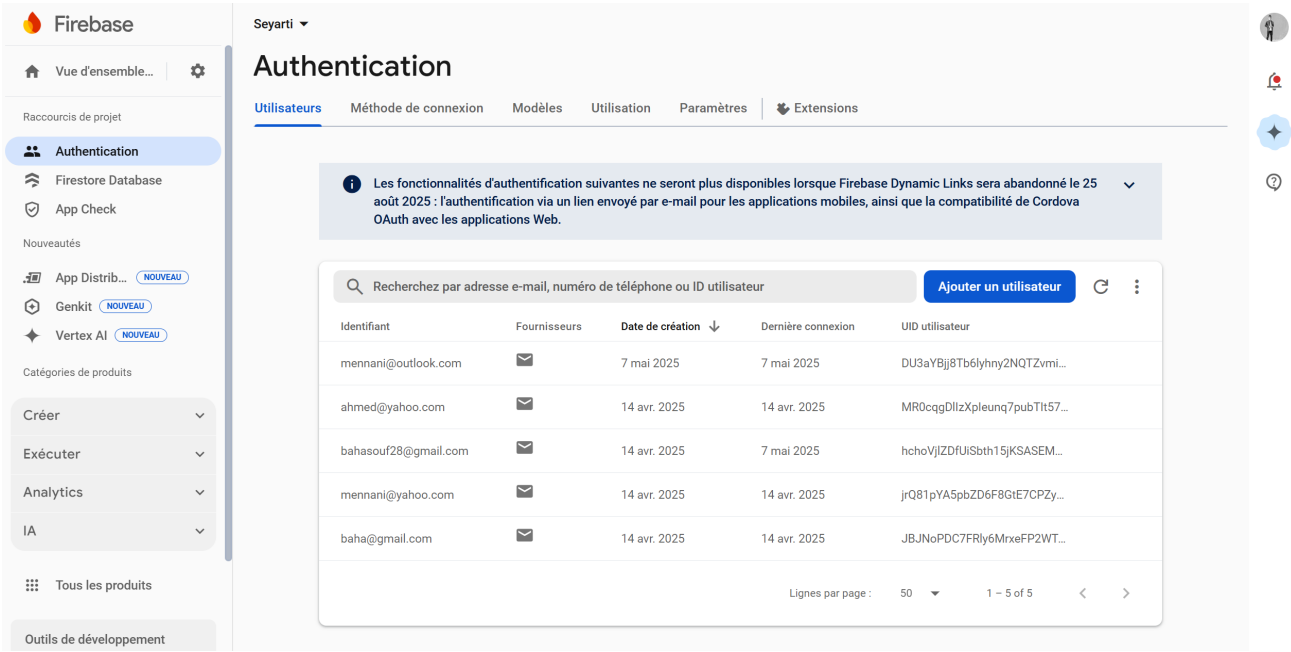

Figure 5.9: Firebase

Firestore

Firestore is a NoSQL cloud database by Firebase, used to store user-related data such as diagnostic history and preferences. It offers real-time synchronization with mobile apps.

When a new user navigates to the Sign Up page 5.13 and submits their registration details, the information is securely stored in Cloud Firestore, as illustrated in Figure 5.10. This cloud-based NoSQL database enables real-time synchronization and scalability. Subsequently, the user can access the application through the Sign In page 5.14, as shown in Figure 5.9. Together, these steps form the core of the authentication workflow, ensuring both data persistence and secure access management.

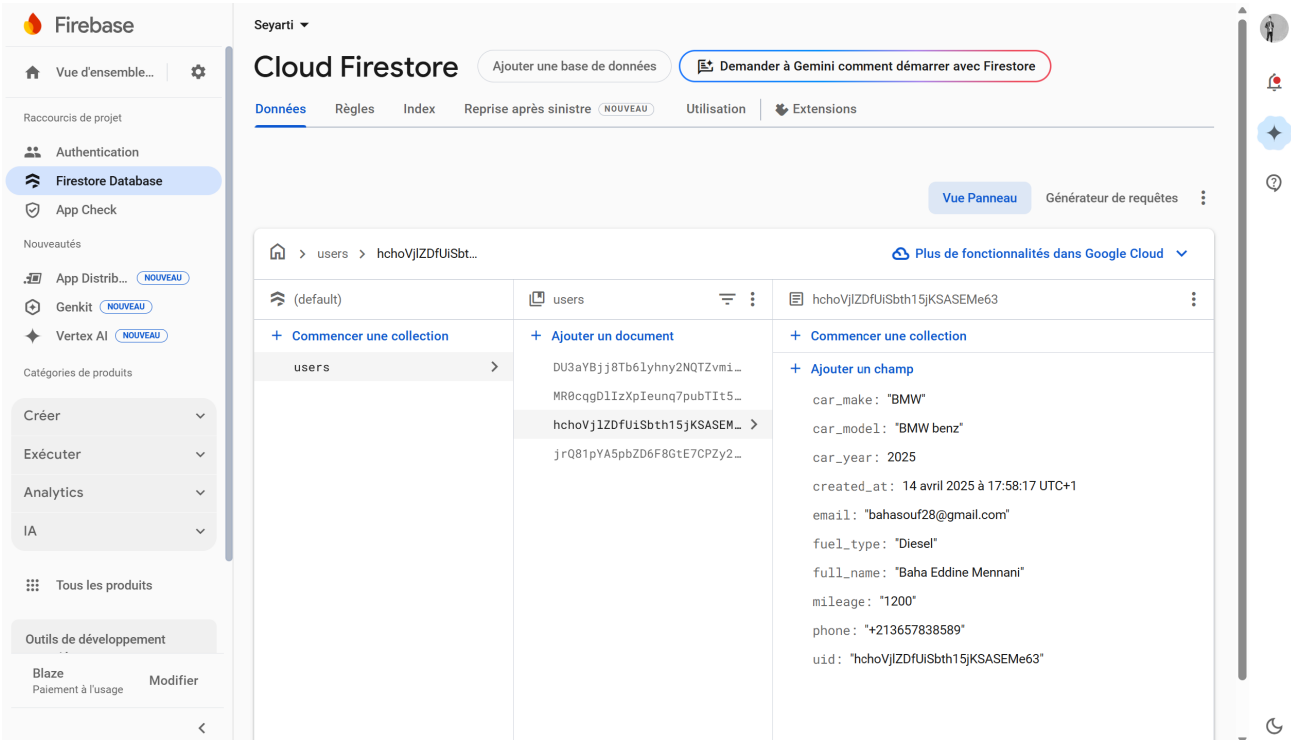

Figure 5.10: Cloud Firestore

5.5 Communication Protocols

HTTP / HTTPS

HTTP and its secure variant HTTPS were used to communicate between the mobile application and the cloud-based REST API endpoints.

REST API

RESTful APIs were implemented to expose the backend services for audio upload, model inference, and results retrieval. These endpoints serve as the bridge between the frontend and backend systems [23].

Figure 5.11 represents the unique URL generated after successfully deploying the project on Google Cloud Run 5.3. This web-based interface allows users to upload engine sound recordings and receive a diagnostic result indicating the detected fault. It serves as the bridge between the trained machine learning model and end-users, enabling remote and real-time inference through a user-friendly platform. Additionally, this URL was seamlessly integrated into the mobile application, allowing users to directly access the cloud service from within the app, ensuring a smooth and efficient user experience.

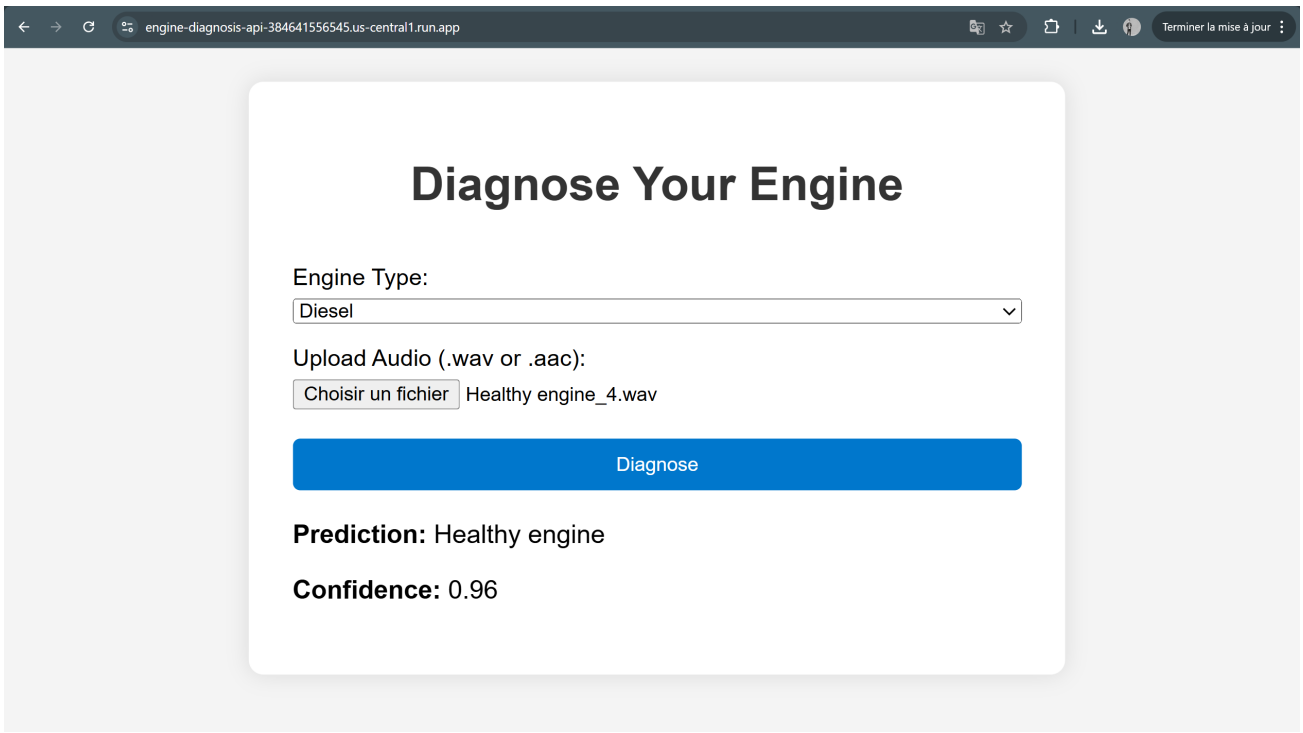

Figure 5.11: URL

5.6 Mobile Application Development

The mobile application was designed to provide an intuitive and efficient interface that allows users to interact seamlessly with the engine fault detection system. It supports both gasoline and diesel engines using binary and multi-class classification models.

Key Features

The application enables users to:

- Record engine sounds in real-time using the device's microphone 5.16 5.17.
- Send the recordings to the cloud for analysis and receive diagnostic feedback instantly 5.16 5.17.
- Identify various engine faults effortlessly, thanks to the integration of multiple trained models 5.16 5.17.
- Search for nearby workshops based on location or availability B.3.
- Browse automotive spare parts and related products B.4.
- Track their vehicle's maintenance history with periodic reminders B.5.

- Save diagnostic results and associated sounds for future reference via a dedicated favorites section B.1.

Audio Capture

The application features a real-time audio recording function that uses the mobile device's microphone. This ensures high compatibility and ease of use without requiring external hardware.

User Interface and Navigation

The user interface (UI) was designed with simplicity and clarity in mind. It offers smooth navigation between core functions, including sound recording, fault result display, engine type selection, and access to additional tools such as favorites and service tracking. The goal was to deliver a user-friendly experience suitable for both technical and non-technical users.

Figure 5.12 illustrates the application's welcome screen, where users are presented with three options: sign in, create a new account if they do not have one, or skip the login process to access the main features of the application directly.

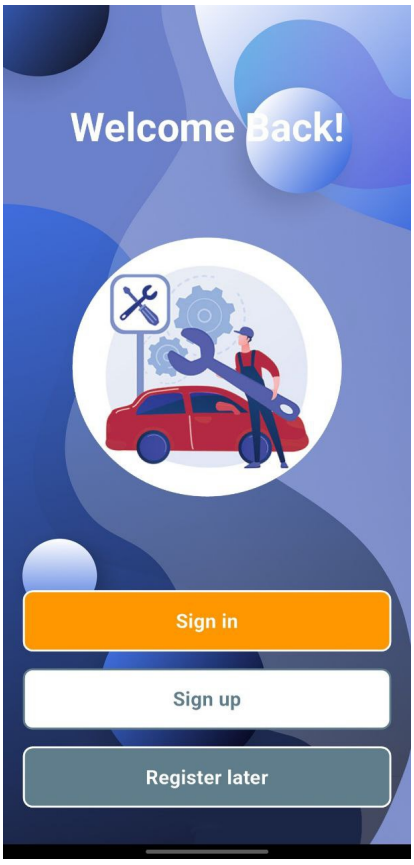

Figure 5.12: Welcome Page of the Application

Figure 5.13 shows the Sign Up page, where new users can create an account by providing their personal information. Upon successful registration, the data is stored securely in Firebase Firestore 5.4, and the user is automatically redirected to the Sign In page 5.14.

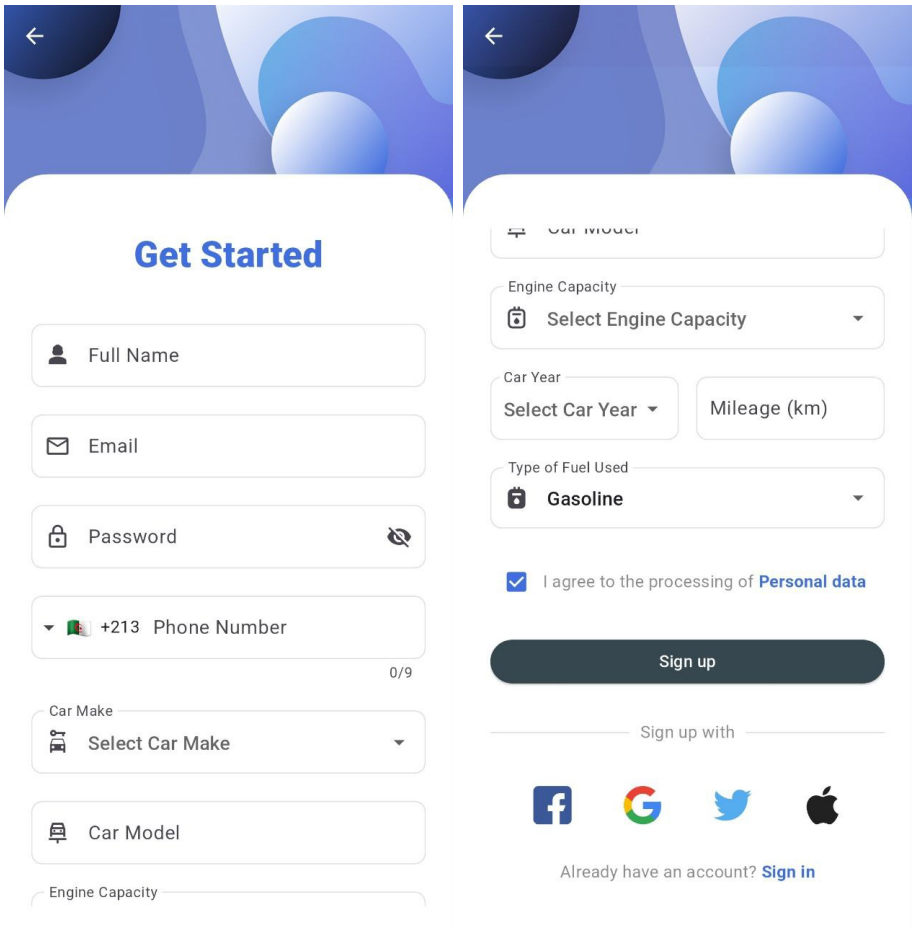

Figure 5.13: User Registration Page (Sign Up)

Figure 5.14 represents the Sign In screen, where user authentication is performed via Firebase Authentication 5.4. Upon successful login, the user is redirected to the application's main homepage.

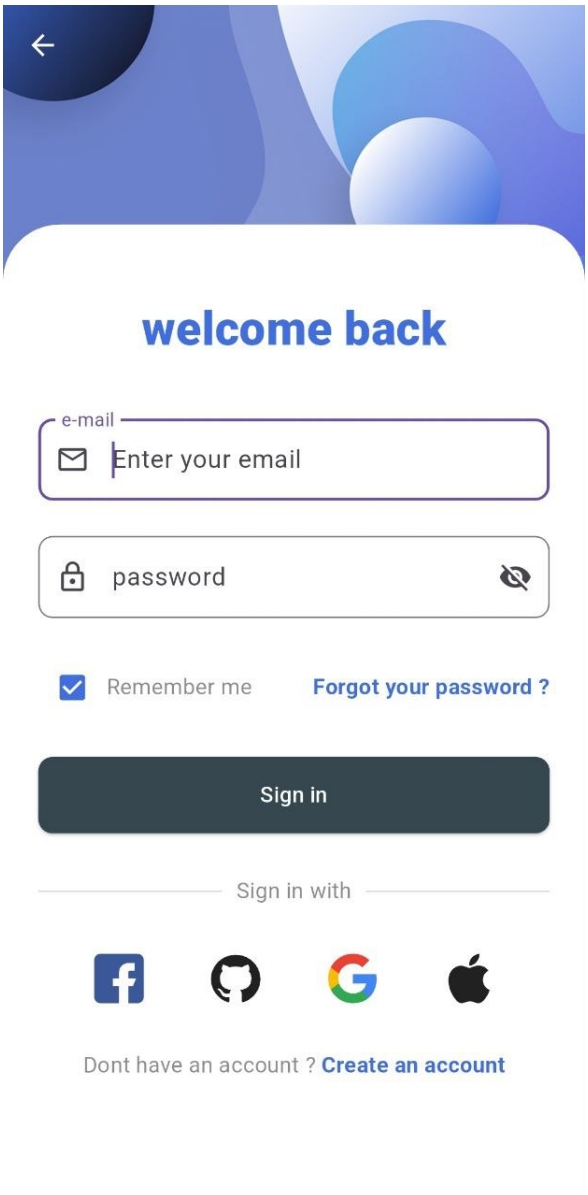

Figure 5.14: User Sign In Page

Figure 5.15 represents the main interface of the application, where users can see and access the essential icons and navigation options needed to use the app's features.

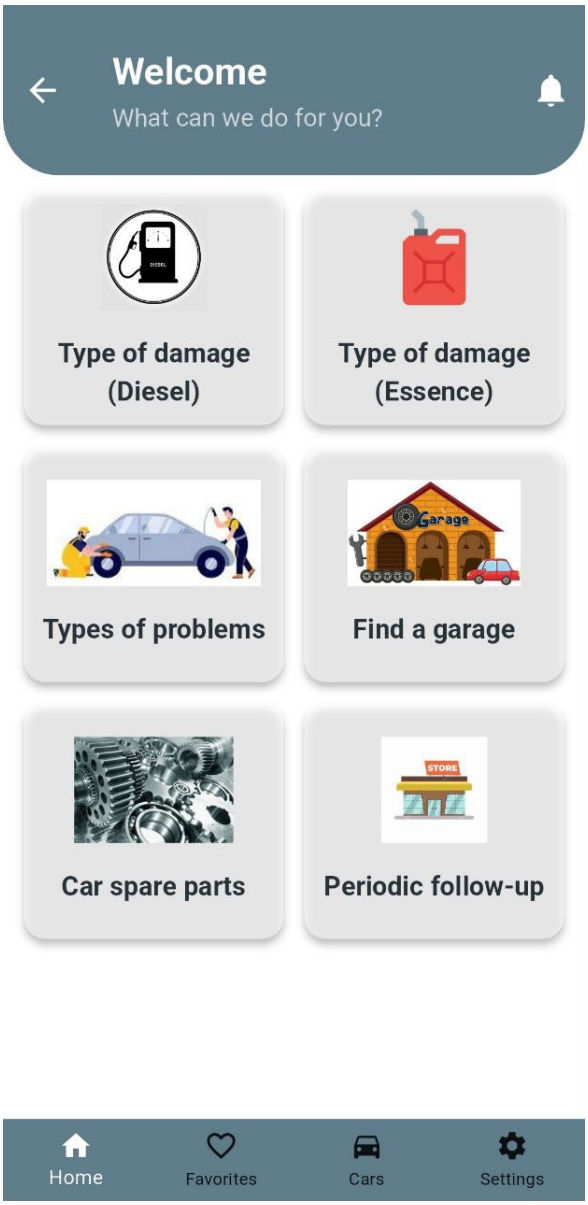

Figure 5.15: Home Page of the Mobile Application Interface

Figure 5.16 represents a diesel engine sound recording page, where the user can record the engine sound for a specified period of time, and then the recording is sent to the server for analysis and fault diagnosis based on the trained model.

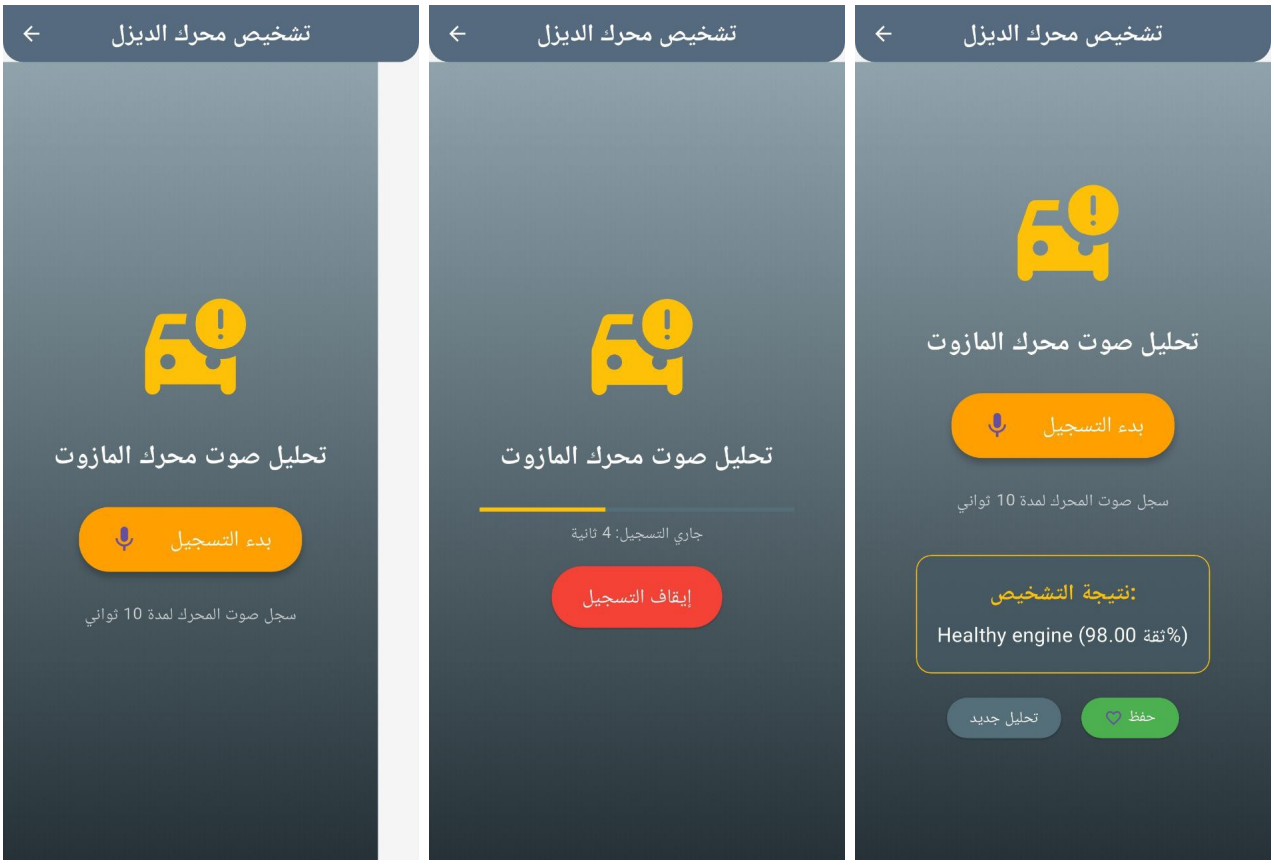

Figure 5.16: Diesel Engine Sound Recording and Fault Diagnosis Interface

Figure 5.17 represents a sound recording page for a gasoline engine, as seen in a diesel engine 5.16. The user can record the engine sound for a specific period of time, and then the recording is sent to the server for analysis and fault diagnosis based on the trained model.

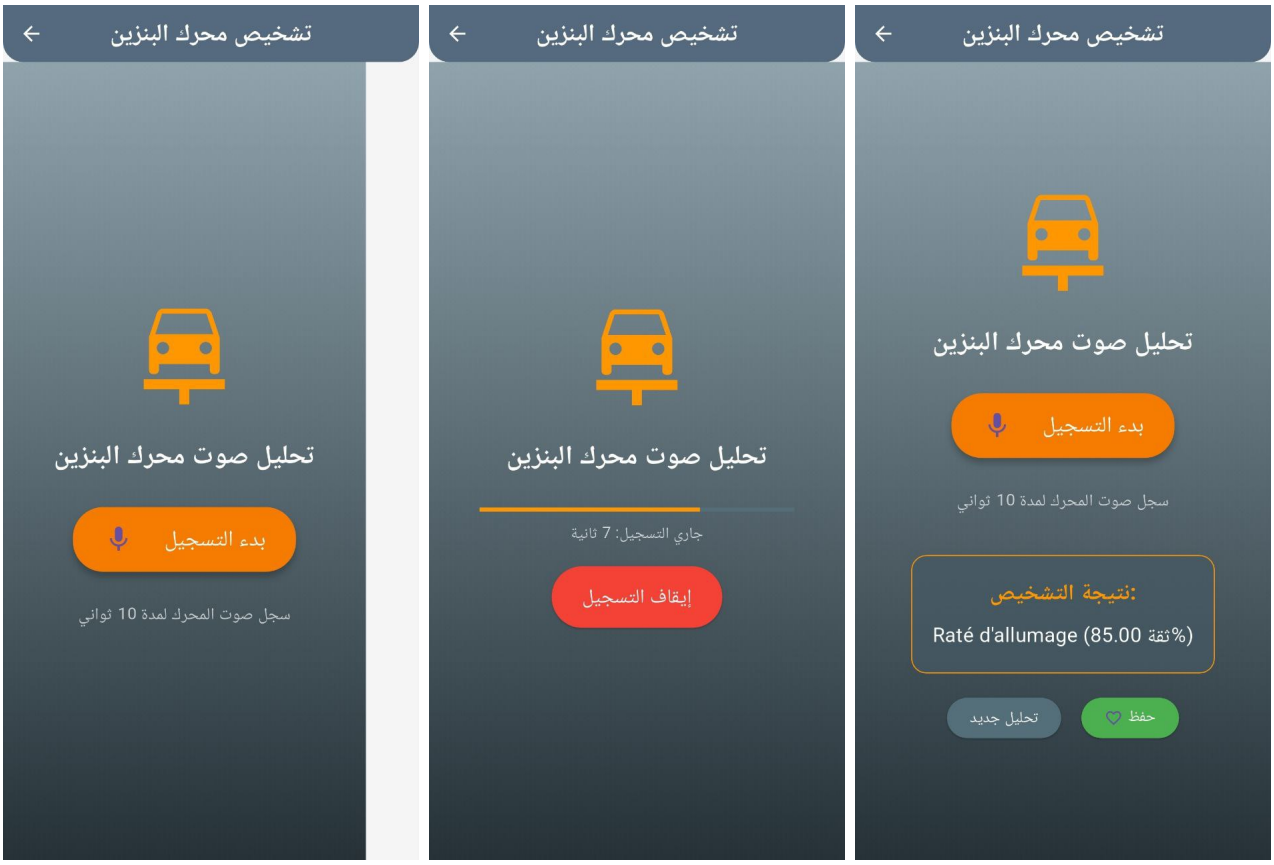

Figure 5.17: Gasoline Engine Sound Recording and Fault Diagnosis Interface

5.7 System Architecture for Real-Time Engine Fault Diagnosis

Figure 5.18 presents a two-part system architecture for sound-based engine fault diagnosis. Part (a) 5.19 illustrates the Development and Training Layer, where engine sound data is collected from both gasoline and diesel engines. This data is preprocessed by extracting audio features such as MFCCs and spectrograms, which are then used to train a variety of machine learning and deep learning models (e.g., KNN, RF, CNN). The most effective models are converted into TFLite format for deployment.

Part (b) 5.20 shows the Live Diagnostic System, consisting of four interconnected layers. First, the User Device Layer enables users to record engine sounds through a mobile app developed in Flutter. The sound is then sent to a Flask-based backend server (Backend Server

Layer), where it is processed in-memory and transformed into a spectrogram. The Model Inference Layer runs the spectrogram through the appropriate TFLite model (gasoline or diesel), returning the predicted fault and confidence level. Finally, the Frontend Application Layer displays the results in the app and optionally stores them in Firebase. This modular architecture ensures efficient model training and smooth real-time fault diagnosis.

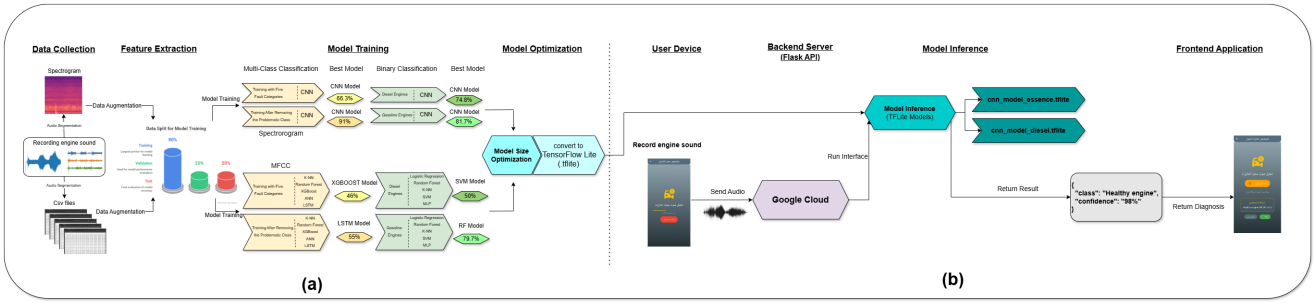

Figure 5.18: System Architecture for Real-Time Engine Fault Diagnosis via Sound Analysis

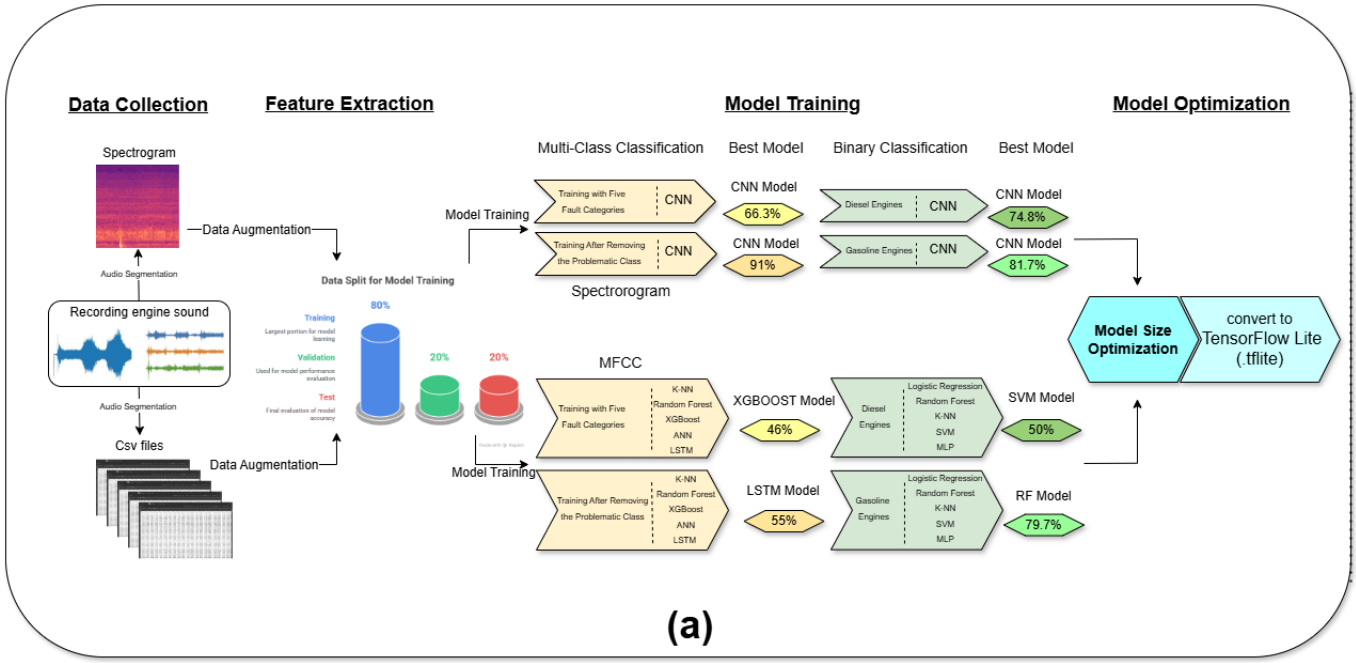

Figure 5.19: (a) Offline Development and Model Training

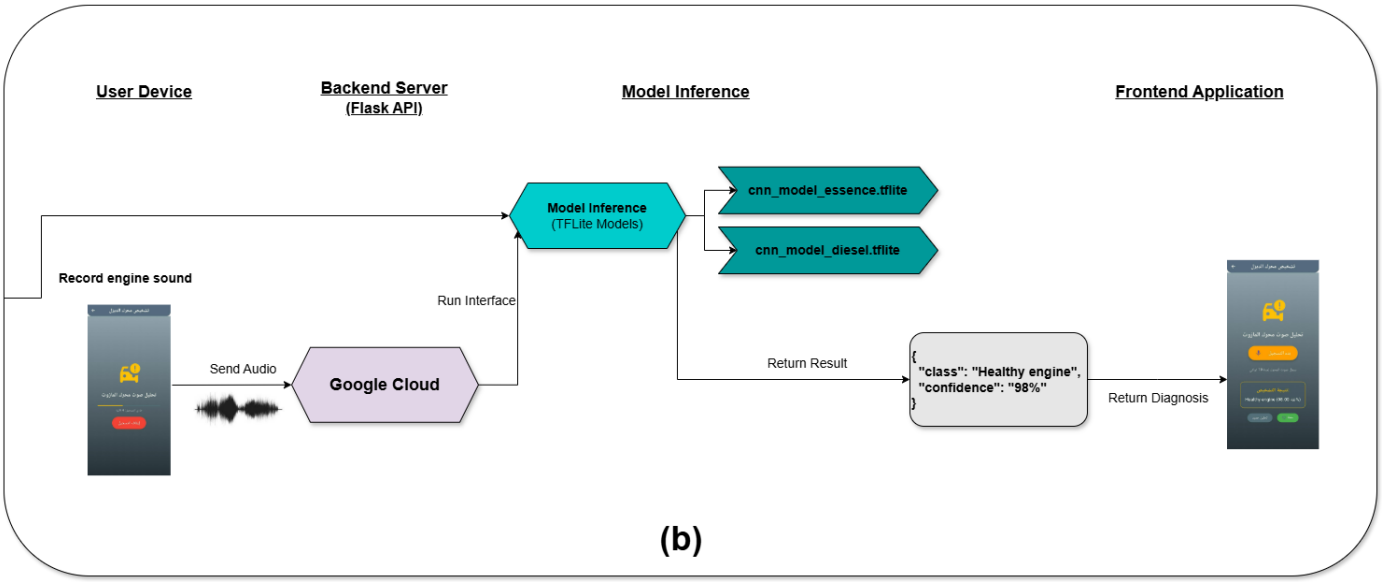

Figure 5.20: (b) Real-Time Fault Detection via Mobile Application

5.8 Challenges and Future Work Recommendations

Despite the successful deployment of the proposed system, the development process encountered several challenges that highlight areas for improvement and further research:

Development Challenges

— **Collaboration with Workshops:** One of the most significant challenges was coordinating with car workshops to collect real engine sound samples under various fault conditions. This required considerable effort and time to gain access to diverse and reliable data sources.

— **Audio Data Variability:** The classification accuracy of models was occasionally affected by background noise and inconsistent recording conditions, which led to variations in extracted features.

— **Distorted Diagnosis Due to External Noise:** In some cases, faulty or loose components unrelated to the engine itself (such as a damaged exhaust or loose shield) produced additional noise. This unintended interference negatively impacted the clarity of the engine sound recordings and led to inaccurate diagnoses by the model.

— **Limited Dataset Diversity:** Although the collected dataset was sufficient for initial training, it does not fully represent the wide range of real-world engine conditions, vehicle

types, and environmental settings.

— **Real-Time Inference Constraints:** Deploying TFLite models, especially spectrogram-based CNNs, on mobile devices introduced performance limitations, particularly on older smartphones with restricted processing and memory capacity.

— **Lack of Audio Preprocessing Tools in Flutter:** The absence of native libraries for converting raw audio into MFCC or spectrogram features in Flutter necessitated cloud-based inference, adding latency and dependency on internet connectivity.

— **Scalability of Backend Services:** While Google Cloud Run offers automatic scaling, handling high-frequency user requests and larger model sizes may require advanced strategies such as caching and load balancing to maintain responsiveness.

— **Absence of User Feedback Loop:** The current system lacks a mechanism for users to confirm or correct predictions. Such a feature could significantly enhance the model's long-term performance through continual learning.

Future Work Recommendations

To address the aforementioned challenges and enhance the system's capabilities, the following future directions are proposed:

— **Multimodal Fault Detection:** Integrating other sensor inputs—such as vibration data—can improve fault detection robustness and overall diagnostic accuracy.

— **Advanced Data Augmentation and Noise Filtering:** Real-time data augmentation and noise reduction techniques can enhance the model's generalization to diverse and noisy environments.

— **Offline Functionality:** Enabling offline inference within the application would improve usability in remote or low-connectivity areas. This can be achieved by embedding and optimizing the models directly within the app.

— **User-Centric Enhancements:** Features like multi-language support, real-time feedback submission, and user correction of predictions can improve accessibility and facilitate the collection of high-quality labeled data for retraining.

— **Expanded Dataset Collection:** Collaborating with automotive service centers or open data communities to build a more diverse and extensive dataset will enhance model reliability across varied scenarios.

— **Continuous Learning System:** Implementing a semi-supervised or federated learning framework will allow the model to evolve over time by learning from new user-provided data while preserving privacy.

— **Smart Notification System:** Incorporating an intelligent alert mechanism to notify users of unusual engine sounds or potential issues could provide proactive maintenance support and enhance user experience.

Figure 5.21 shows the proposed feedback and continuous learning flow, which aims to enhance the diagnostic system over time.

To ensure the continuous improvement of the engine fault diagnosis system, a feedback loop and adaptive learning pipeline are proposed. After each diagnosis, users are prompted to confirm or correct the model's prediction. This user feedback is securely stored in a dedicated Firebase Firestore 5.4 collection. Periodically, this feedback is used to enrich the training dataset. Through a semi-supervised or federated learning approach, the system retrains the TFLite models while maintaining user privacy. Updated models are then redeployed to the cloud, enabling the mobile application to benefit from enhanced diagnostic accuracy without requiring manual updates. This continuous learning cycle ensures that the system evolves and adapts to real-world usage over time.

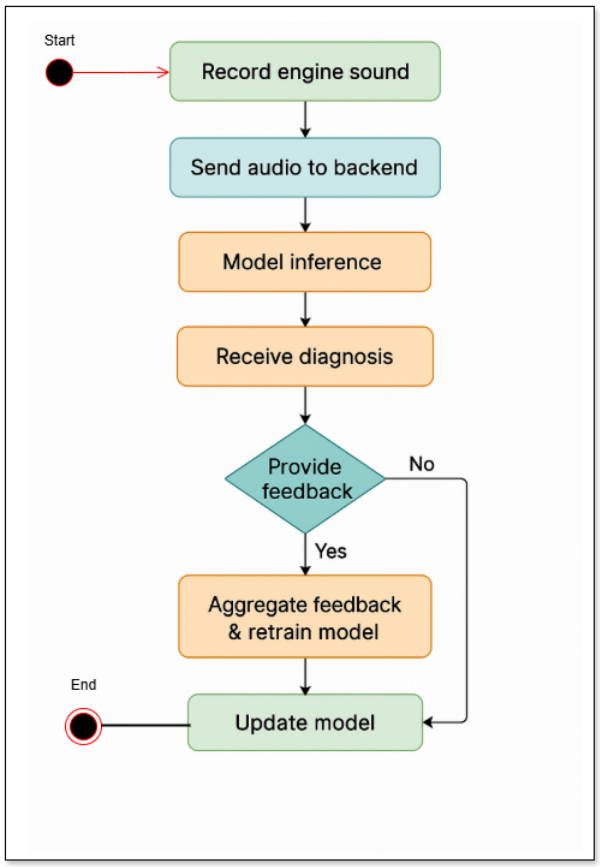

Figure 5.21: Home Page of the Mobile Application Interface

5.9 Conclusion

This chapter detailed the technical tools and development environment used throughout the project. A well-structured software stack, supported by robust frameworks and deployment platforms, enabled the successful implementation of an intelligent, real-time fault diagnosis system accessible through a mobile application.

In addition, the main features of the mobile application and its ability to detect engine faults were presented. The chapter also discussed the various challenges encountered during development, such as data variability, inference limitations, and the absence of native audio preprocessing tools in Flutter. Finally, several recommendations were proposed to guide future improvements, including enhanced data diversity, offline functionality, and user-centric features.

General Conclusion

GENERAL CONCLUSION

This thesis presented a comprehensive approach to the detection and classification of engine faults using acoustic signals and artificial intelligence techniques. By leveraging both traditional machine learning algorithms and deep learning models, the study demonstrated that sound-based analysis can serve as a reliable, non-invasive, and cost-effective method for identifying various engine anomalies. Real-world audio recordings were collected from petrol and diesel engines, processed through MFCC and spectrogram features, and used to train and evaluate multiple classification models, including CNNs, LSTMs, and ensemble methods such as XGBoost and Random Forest.

The results revealed that deep learning models, particularly Convolutional Neural Networks, outperformed traditional algorithms in terms of accuracy and robustness, especially when trained on spectrogram-based visual features. The project also integrated these models into a functional mobile application supported by a Flask backend, enabling real-time fault diagnosis based on user-recorded engine sounds.

Beyond the technical achievements, this work contributes to the field of AI-based predictive maintenance by offering a reproducible pipeline—from data acquisition and preprocessing to deployment. It also highlights challenges such as environmental noise and data diversity, while proposing directions for future work, including multimodal inputs and edge deployment.

In summary, this research lays the foundation for intelligent, sound-based engine diagnostics, bridging the gap between theoretical AI models and real-world applications, with the potential to enhance maintenance strategies and improve safety in transportation systems.

Bibliography

BIBLIOGRAPHY

[1] Spark ignition engine working. Accessed: 2024-04-30. [Online]. Available: https://www.mechanicaleducation.com/spark-ignition-engine-working/

[2] What are diesel engines and how do they work? Accessed: 2024-04-30. [Online]. Available: https://www.topone-power.com/info/what-are-diesel-engines-and-how-do-they-work/

[3] Gasoline vs diesel. Accessed: 2024-04-30. [Online]. Available: https://medium.com/@Breadarose/what-is-the-difference-between-a-diesel-engine-and-a-gasoline-engine-5b31b34789e7

[4] Gasoline vs diesel. Accessed: 2024-04-30. [Online]. Available: https://innovationdiscoveries.space/how-do-gasoline-engines-differ-from-diesel-engines/

[5] Méthode des k plus proches voisins. Accessed: 2024-04-30. [Online]. Available: https://fr.wikipedia.org/wiki/M%C3%A9thode_des_k_plus_proches_voisins

[6] Forêt d'arbres décisionnels. Accessed: 2024-04-30. [Online]. Available: https://fr.wikipedia.org/wiki/For%C3%AAt_d%27arbres_d%C3%A9cisionnels

[7] Xgboost - geeksforgeeks. Accessed: 2024-04-30. [Online]. Available: https://www.geeksforgeeks.org/xgboost/

[8] Classification decision boundary using logistic regression. Accessed: 2024-04-30. [Online]. Available: https://www.researchgate.net/figure/Classification-decision-boundary-using-logistic-regression-The-blue-area-corresponds-to_fig5_325813999

[9] Machine à vecteurs de support. Accessed: 2024-04-30. [Online]. Available: https://fr.wikipedia.org/wiki/Machine_%C3%A0_vecteurs_de_support

[10] Artificial neural network (ann) algorithm. Accessed: 2024-04-30. [Online]. Available: https://www.researchgate.net/figure/Artificial-Neural-Network-ANN-algorithm_fig1_351222802

[11] Convolutional neural network (cnn) algorithm. Accessed: 2024-04-30. [Online]. Available: https://goodguanvs.click/product_tag/85707993_.html

[12] Complete guide to learn lstm models. Accessed: 2024-04-30. [Online]. Available: https://blog.stackademic.com/complete-guide-to-learn-lstm-models-types-applications-and-when-to-use-which-model-f9b779f31714

[13] F. Nasim, S. Masood, A. Jaffar, U. Ahmad, and M. Rashid, "Intelligent sound-based early fault detection system for vehicles." *Computer Systems Science & Engineering*, vol. 46, no. 3, 2023.

[14] D. Fedorishin, L. Forte, P. Schneider, S. Setlur, and V. Govindaraju, "Fine-grained engine fault sound event detection using multimodal signals," in *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2024, pp. 1186–1190.

[15] R. J. Crawford, *Plastics engineering*. Elsevier, 1998.

[16] C. Rakopoulos, "Evaluation of a spark ignition engine cycle using first and second law analysis techniques," *Energy conversion and Management*, vol. 34, no. 12, pp. 1299–1314, 1993.

[17] T. Suzuki, T. Kakegawa, K. Hikino, and A. Obata, "Development of diesel combustion for commercial vehicles," *SAE transactions*, pp. 2114–2132, 1997.

[18] G. Kalghatgi, "Knock onset, knock intensity, superknock and preignition in spark ignition engines," *International Journal of Engine Research*, vol. 19, no. 1, pp. 7–20, 2018.

[19] A. Abid, M. T. Khan, and J. Iqbal, "A review on fault detection and diagnosis techniques: basics and beyond," *Artificial Intelligence Review*, vol. 54, no. 5, pp. 3639–3664, 2021.

[20] M. Lathkar, *Building Web Apps with Python and Flask: Learn to Develop and Deploy Responsive RESTful Web Applications Using Flask Framework (English Edition)*. BPB Publications, 2021.

[21] M. Rashidi, "Application of tensorflow lite on embedded devices: A hands-on practice of tensorflow model conversion to tensorflow lite model and its deployment on smartphone to compare model's performance," 2022.

[22] S. Containerized and S. Ifrah, "Getting started with containers in google cloud platform."

[23] P. Á. Álvarez Fernández, "Factor: web repository of transcribe online classes," Ph.D. dissertation, ETSI_Informatica, 2020.

[24] P. Barry, *Head first Python: A brain-friendly guide.* " O'Reilly Media, Inc.", 2016.

[25] D. Chopra and R. Khurana, *Flutter and Dart: Up and Running: Build native apps for both iOS and Android using a single codebase (English Edition).* Bpb Publications, 2023.

[26] K. Ramasubramanian, A. Singh, K. Ramasubramanian, and A. Singh, "Deep learning using keras and tensorflow," *Machine Learning Using R: With Time Series and Industry-Based Use Cases in R*, pp. 667–688, 2019.

[27] R. Lenain, J. Weston, A. Shivkumar, and E. Fristed, "Surfboard: Audio feature extraction for modern machine learning," *arXiv preprint arXiv:2005.08848*, 2020.

[28] W. McKinney, *Python for data analysis: Data wrangling with pandas, numpy, and jupyter.* " O'Reilly Media, Inc.", 2022.

[29] P. Gupta and A. Bagchi, "Data manipulation with pandas," in *Essentials of Python for Artificial Intelligence and Machine Learning.* Springer, 2024, pp. 197–235.

[30] S. Han and I.-Y. Kwak, "Mastering data visualization with python: practical tips for researchers," *Journal of Minimally Invasive Surgery*, vol. 26, no. 4, p. 167, 2023.

[31] F. J. Blanco-Silva, *Learning SciPy for numerical and scientific computing.* Packt Pub., 2013.

[32] Z. A. Al-Sharif, "A high level audio communications api for the unicon language," Ph.D. dissertation, Citeseer, 2005.

[33] W. R. Vasquez Rivas, "Securing media parsing code," Ph.D. dissertation, 2024.

[34] G. Vagale, M. N. Khan, R. Goutam, D. Nikhil, M. R. Kumar, and T. Narula, "Multi ai agents based black box audio transcriber and aviation incident report generator," in *2025 Fifth International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT).* IEEE, 2025, pp. 1–7.

[35] E. Caprioli, "A semi-supervised approach to bird song classification," Master's thesis, NTNU, 2022.

[36] L. Thomas, "Computer vision and data-analysis solutions for phenotypic screening of small model organisms," Ph.D. dissertation, 2021.

[37] R. Garreta and G. Moncecchi, *Learning scikit-learn: machine learning in python*. Packt Publishing Birmingham, 2013, vol. 2013.

[38] M. Beg, J. Taka, T. Kluyver, A. Konovalov, M. Ragan-Kelley, N. M. Thiéry, and H. Fangohr, "Using jupyter for reproducible scientific workflows," *Computing in Science & Engineering*, vol. 23, no. 2, pp. 36–46, 2021.

[39] A. Studio, "the official ide for android," *Android Studio. URL: https://developer. android. com/studio/index. html*, 2016.

[40] M. Kennedy and M. Harrison, *Effective PyCharm: Learn the PyCharm IDE with a Hands-on Approach*. Effectivepycharm. com., 2019.

[41] J. Loeliger and M. McCullough, *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc.", 2012.

[42] K. Relan, "Deploying flask applications," in *Building REST APIs with Flask: Create Python Web Services with MySQL*. Springer, 2019, pp. 159–182.

[43] A. Mouat, *Using Docker: Developing and deploying software with containers*. " O'Reilly Media, Inc.", 2015.

LIST OF ACRONYMS

AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
MFCC	Mel-Frequency Cepstral Coefficients
KNN	K-Nearest Neighbors
RF	Random Forest
XGBoost	Extreme Gradient Boosting
ANN	Artificial Neural Network
LSTM	Long Short-Term Memory
CNN	Convolutional Neural Network
SVM	Support Vector Machine
LR	Logistic Regression
NN	Neural Network
UML	Unified Modeling Language
API	Application Programming Interface
CSV	Comma-Separated Values
FFmpeg	Fast Forward MPEG (Multimedia Framework)
GCP	Google Cloud Platform
GUI	Graphical User Interface
REST	Representational State Transfer
TFLite	TensorFlow Lite
UI	User Interface

APPENDIX

APPENDIX A

TOOLS AND SOFTWARE

Tools and Software Stack

Programming Languages and Libraries

Python

Python is a high-level programming language that has a clear and readable syntax. It highlights simplicity and efficiency as key aspects, making it suitable for a wide range of applications, from web development and data analysis to artificial intelligence and automation [24].

Dart

Dart is a client-optimized language developed by Google, primarily used for building cross-platform mobile applications with Flutter. It offers fast compilation and a reactive programming model [25].

TensorFlow / Keras

TensorFlow and its high-level API, Keras, are open-source libraries designed for developing and training machine learning and deep learning models. They support scalable computation on CPUs, GPUs, and TPUs [26].

Librosa

Librosa is a Python library used for analyzing and processing audio signals, especially in music and voice-related projects. It supports feature extraction such as MFCCs and spectrograms [27].

NumPy

NumPy is a fundamental Python library for numerical computing, providing the creation and manipulation of large arrays and matrices efficiently. It offers a wide collection of mathematical functions, making it useful for data analysis and scientific computing tasks [28].

Pandas

Pandas is a data manipulation and analysis library that provides data structures like DataFrames. It is widely used for cleaning, transforming, and visualizing structured datasets [29].

Matplotlib

Matplotlib is a 2D plotting library for Python that enables the visualization of data in the form of graphs, charts, and plots. It is particularly useful for exploratory data analysis [30].

Scipy

SciPy extends NumPy by adding modules for optimization, signal processing, linear algebra, and more. It is a key tool for scientific and technical computing [31].

Soundfile

Soundfile is a library for reading and writing audio files in various formats, including WAV and FLAC. It provides a simple API for audio file I/O [32].

ffmpeg-python

This is a Python wrapper for FFmpeg, a multimedia framework that can decode, encode, transcode, and stream audio and video files. It enables the conversion and processing of audio files programmatically [33].

Pydub

Pydub is a high-level audio manipulation library that simplifies tasks such as slicing, concatenation, and format conversion of audio files [34].

Pillow

Pillow is a Python Imaging Library (PIL) fork, used for opening, manipulating, and saving many different image file formats. It is utilized for processing spectrogram images [35].

OpenCV (opencv-python-headless)

OpenCV is a powerful library for computer vision tasks. The headless version excludes GUI functionality, making it suitable for servers and deployment environments [36].

Scikit-learn

Scikit-learn is a machine learning library that includes tools for classification, regression, clustering, and model evaluation. It was used to implement traditional models and metrics [37].

Development Tools

Visual Studio Code

VS Code is a lightweight but powerful source code editor that supports debugging, syntax highlighting, and Git integration. It was used for general-purpose coding.

Jupyter Notebook

Jupyter provides an interactive environment for running Python code in cells. It was essential during the model development and experimentation phases [38].

Android Studio / Flutter

Android Studio is the official IDE for Android development. Combined with Flutter, it allowed building cross-platform mobile applications with a native feel [39].

PyCharm

PyCharm is an integrated development environment (IDE) designed for Python programming. It offers advanced features like code analysis, debugging, and intelligent code completion to enhance developer productivity [40].

Git

Git is a version control system used to track changes in source code during software development. It simplifies collaboration among multiple developers and provides efficient features like versioning and branching [41].

Postman

Postman is a collaboration platform for API development and testing. It provides a user-friendly interface to send HTTP requests, inspect responses, and automate testing of RESTful services. It was used extensively to verify the functionality and performance of the Flask-based backend before integrating it with the mobile application.

Gunicorn

Gunicorn is a Python WSGI HTTP server for UNIX systems. It is commonly used to deploy Flask applications in a production environment [42].

Docker

Docker is a platform that uses containerization to ensure consistent environments across development, testing, and deployment. It improves portability and scalability [43].

Overleaf

Overleaf is a collaborative LaTeX editor used to write and manage scientific documents. It facilitated structured writing and citation management for the thesis.

APPENDIX B

SCREENSHOTS OF THE APPLICATION

Application interfaces

Figure B.1 represents the "Favorites" page, where, as noted earlier, the user can save diagnostic results after each voice analysis. This page displays all recordings that have been saved for later review.

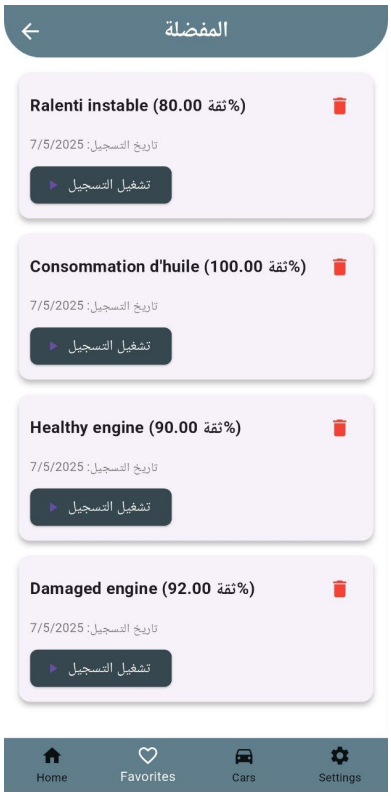

Figure B.1: Saved Diagnoses and Favorite Recordings Interface

Figure B.2 represents the page that displays when you click on the Type of Problems icon, where the user is presented with a list of some common car faults, with the aim of guiding and helping them identify the possible symptoms of the problem.

Figure B.2: Common Car Problems Information Page

Figure B.3 represents the page that displays when you click the Find a Garage icon, where the user can search for nearby repair shops and book a service appointment based on their needs and preferences.

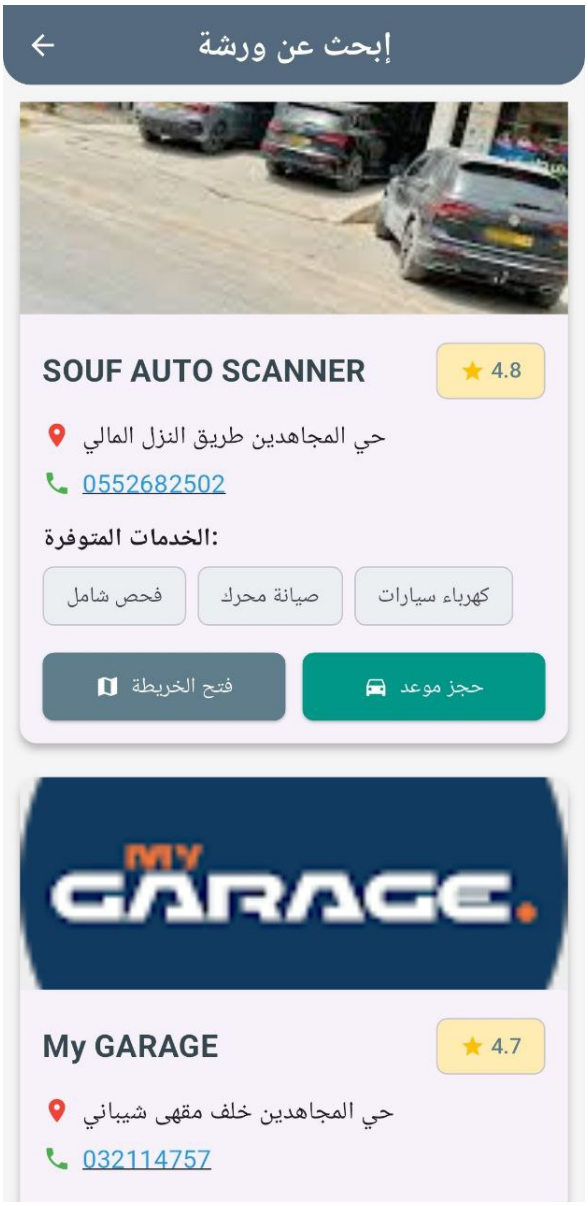

Figure B.3: Garage Finder and Appointment Booking Page

Figure B.4 represents the page that appears when you click on the Car Spare Parts icon, where different types of car spare parts are displayed, helping the user to search and find the parts he needs easily.

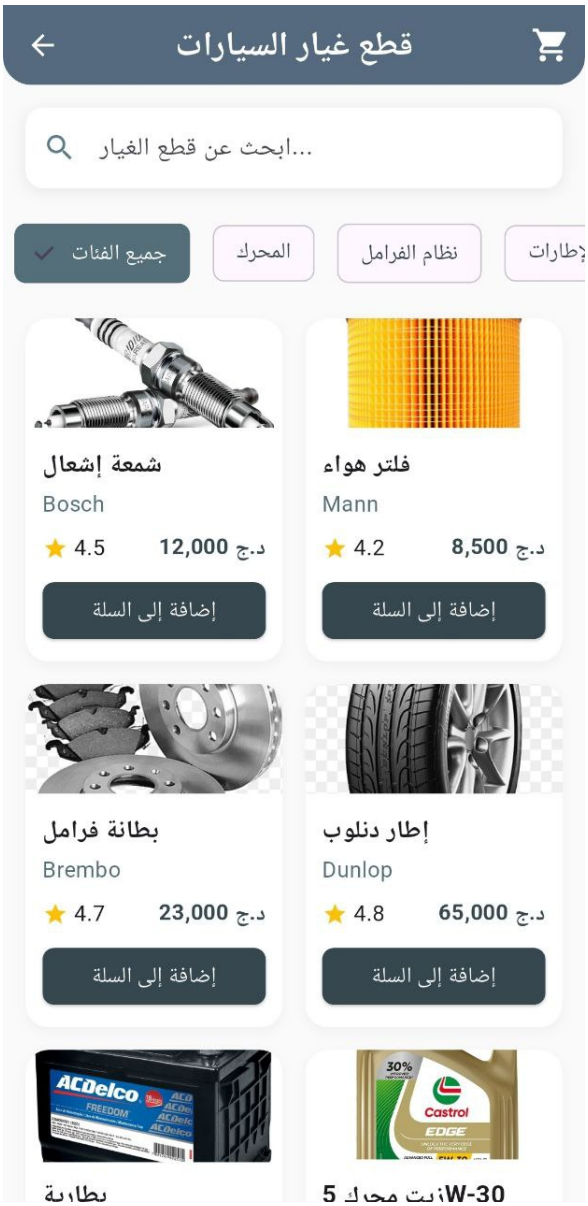

Figure B.4: Car Spare Parts Browsing Interface

Figure B.5 represents the page displayed when you click the Periodic Follow-up icon, providing an overview of the vehicle's periodic follow-up, including upcoming tasks and checks that should be performed to ensure regular maintenance.

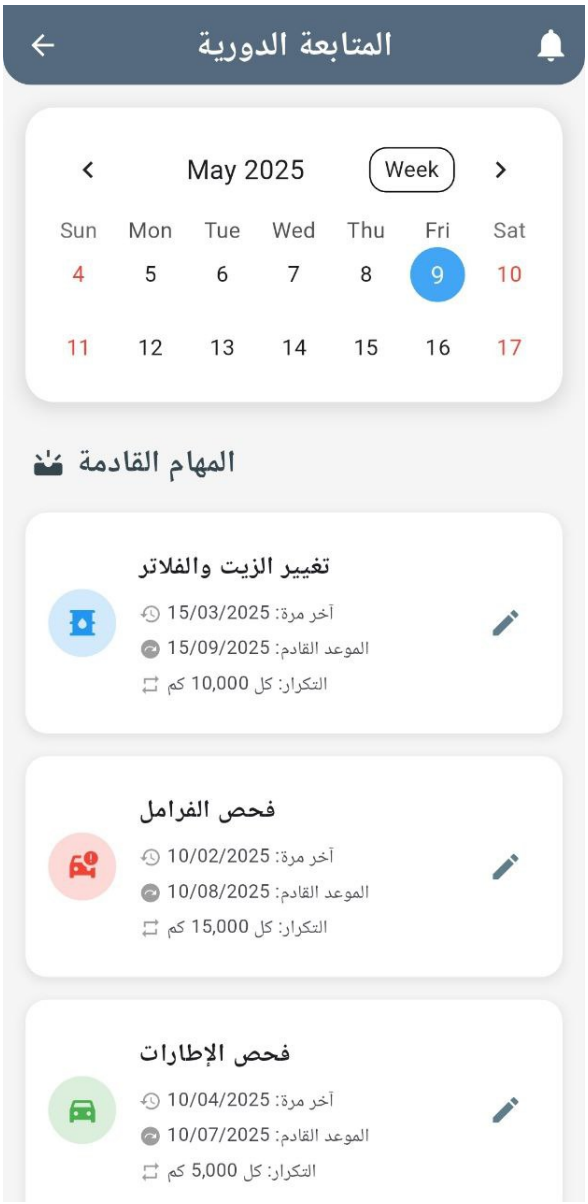

Figure B.5: Periodic Follow-up and Maintenance Tasks Page

APPENDIX C

SAMPLE CODE SNIPPETS

K-NN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score,
    confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

model_knn = KNeighborsClassifier(n_neighbors=5)
model_knn.fit(X_train_classic, y_train_classic)
```

Listing C.1: K-NN Model

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

model_rf = RandomForestClassifier(n_estimators=100, random_state=42)
model_rf.fit(X_train_classic, y_train_classic)
```

Listing C.2: Random Forest Model

XGBOOST

```
from xgboost import XGBClassifier

model_xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss',
    random_state=42)
model_xgb.fit(X_train_classic, y_train_classic)
```

Listing C.3: XGBOOST Model

ANN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

model_ann = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_ann.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(num_classes, activation='softmax')
])

model_ann.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model_ann.summary()
```

Listing C.4: ANN Model

CNN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
    Dropout
```

```
from tensorflow.keras.optimizers import Adam

model_cnn = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(IMG_SIZE[0], IMG_SIZE[1],
        1)),
    MaxPooling2D((2,2)),

    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(y_train_ohe_cnn.shape[1], activation='softmax')
])

model_cnn.compile(optimizer=Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

model_cnn.summary()
```

Listing C.5: CNN Model

LSTM

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam

model_lstm = Sequential([
    LSTM(64, input_shape=(X_train_lstm.shape[1], 1), return_sequences=False),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(y_train_ohe_lstm.shape[1], activation='softmax')
])

model_lstm.compile(optimizer=Adam(learning_rate=0.001),
                   loss='categorical_crossentropy',
```

```
                    metrics=['accuracy'])

model_lstm.summary()
```

Listing C.6: LSTM Model

SVM

```
from sklearn.svm import SVC

model_svm = SVC(kernel='rbf', random_state=42)

model_svm.fit(x_train,y_train)
```

Listing C.7: SVM Model

Logistic Reggression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    f1_score, confusion_matrix, classification_report

model_logreg = LogisticRegression(max_iter=1000, random_state=42)

model_logreg.fit(x_train,y_train)
```

Listing C.8: Logistic Reggression Model

MLP

```
from sklearn.neural_network import MLPClassifier

model_nn = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500, random_state
    =42)

model_nn.fit(x_train,y_train)
```

Listing C.9: Neural Network Model