

COMPARATIVE ANALYSIS OF TinyML MODELS FOR AUTOMOTIVE ECU FAULT DETECTION USING OBD-II DATA

MAJOR PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE OF

MASTER OF TECHNOLOGY IN

ARTIFICIAL INTELLIGENCE

Submitted by

Megha Bharti (24/AFI/20)

Under the supervision of

Prof. RAJNI JINDAL



DEPARTMENT OF COMPUTER SCIENCE

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering), Bawana Road, Delhi-110042

MAY, 2026

**DEPARTMENT OF COMPUTER SCIENCE
DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, Megha Bharti, Roll No – 24/AFI/20, student of M.Tech (Artificial Intelligence), hereby declare that the project dissertation titled "Comparative Analysis of TinyML Models for Automotive ECU Fault Detection Using OBD-II Data" which is submitted by me to the Department of Computer Science, Delhi Technological University, Delhi, in partial fulfilment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma, Associateship, Fellowship or other similar title or recognition.

Place: Delhi

Date: 31.05.2026

Megha Bharti

DEPARTMENT OF COMPUTER SCIENCE
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)

CERTIFICATE

I hereby certify that the Project Dissertation titled "Comparative Analysis of TinyML Models for Automotive ECU Fault Detection Using OBD-II Data" which is submitted by Megha Bharti, Roll No – 24/AFI/20, Artificial Intelligence (AFI), Delhi Technological University, Delhi, in partial fulfilment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge, the contents of this submission have never been submitted in whole or in part, in any form or language, for any degree or diploma to this university or any other institution.

Place: Delhi
Date: 31.05.2026

Prof. Rajni Jindal
SUPERVISOR

**DEPARTMENT OF COMPUTER SCIENCE
DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

ACKNOWLEDGEMENT

Firstly, I would like to thank Prof. Rajni Jindal with whom I have been working all through the thesis process with constant help and guidance. I appreciate her valuable feedback on the research and the extensive knowledge she has on machine learning techniques. Throughout, she guided me in taking this research forward without compromising on scientific rigor.

Next, I would like to thank the department of computer science at Delhi Technological University for providing me with a conducive atmosphere for performing research. The open environment helped me in many ways in conducting my experiments.

A special mention should be made of Robert Bosch India Limited where I am currently doing my internship. Being in Bosch has made me aware of how my problem formulation and the sensor data processing are related to actual scenarios, thanks to the exposure provided by my internship.

Moreover, I wish to thank my peers for their valuable criticisms and discussions that have helped shape some methodologies used in the work.

Lastly, I would like to thank my family members for bearing with me through this long academic journey and encouraging me all the way through.

Place: Delhi

Date: 31.05.2026

Megha Bharti

ABSTRACT

Modern cars are filled with electronic components called Electronic Control Units (ECUs) that play a crucial role in the management of most vehicle functionalities by processing huge amounts of sensor data required for the operation of functions such as fuel injection, timing of the combustion, and emissions control. Traditionally, any malfunction in ECUs was diagnosed via sending this data to cloud servers (that introduces unacceptable delay) or bringing the car to a shop equipped with expensive diagnostic equipment - a far cry from the real-world budget constraints of embedded automotive systems.

TinyML can help here by efficiently compressing neural networks, thus allowing running inference on low-cost microcontrollers onboard the car, even those with as little as 256 KB memory.

We perform a detailed comparison between four neural architectures designed for onboard ECU fault detection using OBD-II signals: Multi-Layer Perceptron (MLP), a one-dimensional Convolutional Neural Network (CNN-1D), a Long Short-Term Memory network (LSTM), and a hybrid CNN-LSTM network combining both convolution and sequence modeling. The five classes (Normal, Overheating, Misfire, Air Leak, Throttle Fault) are embedded in the KIT Automotive OBD-II Dataset and the models are fine-tuned using these data, with class labels assigned by our custom rule-based fault labelling approach which was validated by automotive engineers. After fine-tuning, all models are quantized to INT8 precision using TensorFlow Lite, and the changes in fault detection performance, memory consumption, and inference speed caused by the quantization are analyzed.

In all three metrics, CNN-LSTM performs best with 96.1% fault detection accuracy ($F1 = 0.960$), followed by CNN-1D with 94.2%, LSTM with 93.7%, and MLP with 91.4%. The main reason behind this performance is the ability of hybrid architecture to leverage convolution operations for detecting local transient features of faults while using LSTM layers for understanding their long-term dependencies. Quantized to INT8, the hybrid model needs 164 KB of flash storage and takes 31 ms for inference, thus becoming suitable for high-end automotive processors like STM32H7 and NXP i.MX RT series. For applications constrained by hardware, MLP can become an alternative providing 91.4% accuracy with 38 KB of flash consumed and 12 ms of inference speed.

The major contributions of this paper are as follows: (1) a reliable labeling protocol for unlabelled OBD-II dataset; (2) a consistent benchmarking procedure for automotive TinyML models including fault detection accuracy, memory consumption, and inference speed; (3) the first analysis of performance of a hybrid CNN-LSTM architecture under INT8 quantization constraints.

CONTENTS

Candidate's Declaration

Certificate

Acknowledgement

Abstract

Contents

List of Tables

List of Figures

List of Symbols / Abbreviations

1 INTRODUCTION	1
1.1 Overview	1
1.1.1 Background and Context	1
1.1.2 Problem Statement	2
1.1.3 Research Objectives	3
1.1.4 Thesis Organisation	3
2 LITERATURE REVIEW	4
2.1 Introduction	4
2.2 Background and Research Gaps	5
2.3 TinyML on Automotive Embedded Hardware	6
2.4 Deep Learning Architectures for Time-Series Fault Detection	7
2.5 Hybrid CNN-LSTM Models in Literature	8
2.6 AI Agents in Intelligent Vehicle Systems	9
2.7 Summary of Literature Findings	10
3 METHODOLOGY	11
3.1 Objective	11
3.2 Dataset Acquisition and Preprocessing	11
3.3 Fault Label Generation	13
3.4 Model Architecture Design	14
3.5 CNN-LSTM Hybrid Architecture	18
3.6 Model Training Configuration	19
3.7 INT8 Post-Training Quantization	20
3.8 Deployment Simulation	21
3.9 Evaluation Metrics	22
3.10 AI Agent-Assisted Development Pipeline	23
4 RESULTS AND DISCUSSION	23
4.1 Classification Performance	23
4.2 Per-Class Analysis	24
4.3 Deployment Metrics	26
4.4 Hybrid Model Advancement	27
4.5 AI Agent Contribution to Model Improvement	29

4.6	Key Observations	31
5	INDUSTRIAL CONTEXT: BOSCH INTERNSHIP AND REAL-WORLD RELEVANCE	35
5.1	My Bosch Internship Experience	35
5.2	How Bosch Utilises ECU Fault Detection in Practice	35
5.2.1	Bosch's ECU Ecosystem and the Diagnostic Challenge	35
5.2.2	Where This Thesis Fits in Bosch's AI Strategy	36
5.2.3	Empirical Insights Gained in the Bosch Workshops	37
5.3	Why These Results Are Important — Beyond the Numbers	38
5.3.1	The Human Impact of Earlier Fault Detection	38
5.3.2	What the 4.7% Accuracy Gap Means	39
6	AGENTIC AI: THE NEXT FRONTIER FOR ECU DIAGNOSTICS	40
6.1	Agentic AI: Definition and Importance in Relation to the Current Situation	40
6.2	How Agentic AI Improves This Specific Solution	41
6.2.1	Reducing False Positives Through Temporal Reasoning	41
6.2.2	Adding Human-Readable Explanations Through LLM Reasoning	41
6.2.3	Continuous Self-Calibration Over the Vehicle Lifecycle	42
6.2.4	Fleet Intelligence Through Multi-Agent Learning	43
7	CONCLUSION AND FUTURE SCOPE	44
7.1	Conclusion	44
7.2	Future Scope	47
	References	52

LIST OF TABLES

Table 2.1	Summary of Literature Review on TinyML and ECU Fault Detection	9
Table 3.1	OBD-II Dataset Fault Class Distribution and Labelling Rules	13
Table 3.2	Training Data Sets, Validation Data Sets, and Test Data Sets	14
Table 3.3	MLP Architecture Summary	15
Table 3.4	CNN-1D Architecture Summary	16
Table 3.5	LSTM Architecture Summary	17
Table 3.6	CNN-LSTM Hybrid Architecture Summary	19
Table 4.1	Classification Performance Comparison Across All Four Models	23
Table 4.2	Per-Class F1-Score for All Four Models	25
Table 4.3	Model Size and Inference Latency After INT8 Quantization	26
Table 4.4	AI Agent Contribution Summary Across Research Pipeline Stages	29

LIST OF FIGURES

Figure 3.1	Overall System Architecture for TinyML-Based ECU Fault Detection	12
Figure 3.2	OBD-II Data Preprocessing and Labelling Pipeline	13
Figure 3.3	MLP Architecture for ECU Fault Classification	15
Figure 3.4	CNN-1D Architecture for ECU Fault Classification	16
Figure 3.5	LSTM Architecture for ECU Fault Classification	17
Figure 3.6	CNN-LSTM Hybrid Architecture for ECU Fault Classification	18
Figure 3.7	INT8 Post-Training Quantization Pipeline	21
Figure 3.8	AI Agent Architecture for TinyML ECU Fault Detection System	23
Figure 3.9	AI Agent-Assisted Research and Development Workflow	24
Figure 4.1	Accuracy Comparison of All Four Models (Post-INT8)	25
Figure 4.2	TinyML Cloud vs Edge Inference — Core Motivation (Warden 2020)	25
Figure 4.3	OBD-II Sensor Data Streams with Fault Events (Flores 2022)	26
Figure 4.4	INT8 Quantization Effect on Accuracy and Model Size (Jacob 2018)	27
Figure 4.5	Confusion Matrix — CNN-LSTM Hybrid (Test Set 750 Samples)	28
Figure 4.6	Training and Validation Accuracy Curves — All Four Models	29
Figure 4.7	ROC Curves — Macro-Averaged Across Five Fault Classes	30
Figure 4.8	Per-Class F1-Score Comparison — All Four Models	31
Figure 4.9	Model Size vs. Model Accuracy: Trade-off Analysis (Scatter Graph)	32
Figure 4.10	Why These Results Matter — Practical Impact Summary	34
Figure 5.1	Bosch Industrial Integration Architecture	38
Figure 5.2	Agentic AI Closed-Loop System for ECU Fault Management	42

LIST OF SYMBOLS / ABBREVIATIONS

TinyML	Tiny Machine Learning
ECU	Electronic Control Unit
OBD-II	On-Board Diagnostics Version II
MLP	Multi-Layer Perceptron
CNN-1D	One-Dimensional Convolutional Neural Network
LSTM	Long Short-Term Memory
CNN-LSTM	Convolutional Neural Network – Long Short-Term Memory Hybrid
TF Lite	TensorFlow Lite
INT8	8-bit Integer Quantization
MCU	Microcontroller Unit
RPM	Revolutions Per Minute
MAF	Mass Air Flow
MAP	Manifold Absolute Pressure
BN	Batch Normalisation
ReLU	Rectified Linear Unit
DTC	Diagnostic Trouble Code
TOPS	Tera Operations Per Second
RAM	Random Access Memory
AUC-ROC	Area Under the Receiver Operating Characteristic Curve
ADAS	Advanced Driver Assistance Systems

Chapter 1

INTRODUCTION

1.1 Overview

1.1.1 Background and Context

On average, a modern passenger car features anywhere from 70 to 100 Electronic Control Units (ECUs), each designed to perform a particular role in the vehicle. In general, ECM controls combustion and fueling, whereas TCM selects gears, ABS adjusts braking force when detecting wheel locking, BCM coordinates the operation of electrical components, and so forth—there being dozens of other roles that ECUs fill. They interact using in-car bus networks like CAN and offer a standard interface for diagnostics, i.e., OBD-II port, that became mandatory on all cars produced for sale in the United States in 1996 and in the European Union starting in 2001. An external device connected to OBD-II can access current readings of standardized sensor channels including engine revolutions per minute, road speed, mass airflow, intake pressure, throttle position, engine coolant temperature, and others, thus providing a detailed picture of the power train operation. Should an ECU detect anything out of place with any parameter being monitored, it will trigger a corresponding Diagnostic Trouble Code (DTC) and light up the Malfunction Indicator Lamp (MIL). But this rule-based DTC system is known to have several significant drawbacks: it responds exclusively to a fault having already occurred, cannot detect early warning signs of degradation, and offers no indication of fault severity and urgency whatsoever. [14] [15]

Machine learning has been gaining increasing attention recently as a diagnostic alternative because ML models can recognize much more complex patterns in multi-channel data than conventional rules and hence detect pre-fault trends in time. However, the common solution involving sending the data to cloud servers still suffers from considerable round-trip latencies of hundreds of milliseconds and hence cannot satisfy the needs of automotive applications with their millisecond requirements. Besides, cloud-based diagnostics cannot be employed where there is poor or unavailable internet connectivity. [9] [10]

TinyML offers solutions to these problems, namely running compressed neural network models locally on MCU-equipped microcontrollers already installed in the vehicle. This method rests on the idea that thanks to compression (e.g., INT8 post-training

quantization, weight pruning, and knowledge distillation), one can significantly decrease the memory requirement of the model, making it run on sub-megabyte MCUs at a relatively small cost in terms of accuracy loss. [3] [15]

1.1.2 Problem Statement

While the potential benefits of TinyML for vehicle diagnostics appear clear, there are notable gaps in the current literature. Individual architectures have been studied in isolation with MLP being applied to emission prediction tasks, LSTMs for fault isolation, and CNN-LSTM hybrids to diagnose faults generally; however, no comparative studies have yet compared all four under the same INT8 quantization constraints for OBD-II ECU fault classification. Additionally, while CNN-LSTM hybrids have shown impressive results under GPU-based computations, no comparisons have been made of their accuracy relative to other architectures when constrained by aggressive INT8 quantization needed for MCU deployment.

Without a benchmark study, designers of embedded fault detection solutions would be lacking an empirical basis on which to select their architecture since, depending on the hardware used (from barebones controllers in budget cars to powerful embedded processors in premium cars), the best architectural choices will vary.

1.1.3 Research Objectives

Specific goals pursued in this thesis include:

- Collecting, processing, and labeling a publicly available automotive sensor OBD-II dataset based on a domain-specific rules-based approach to fault generation.
- Designing, implementing, and training four deep learning neural networks (MLP, CNN1D, LSTM, and CNN-LSTM hybrid) to classify ECUs into five classes.
- Quantizing all four trained models using INT8 post-training quantization and comparing performance tradeoffs.
- Comparing and analyzing the performance of architectures both in terms of fault detection quality and suitability for embedded deployment.
- Elucidating the reasons behind better performance in the CNN-LSTM hybrid architecture and quantifying its advantages over single-architecture baselines.
- Establishing an open-source reproducible pipeline for benchmarking TinyML ECU fault classification.

1.1.4 Thesis Organisation

These chapters are arranged as follows. Chapter 2 reviews the relevant literature concerning aspects of TinyML implementation, OBD-II-based diagnostic systems, time series deep learning models, and prior CNN-LSTM studies. Chapter 3 describes the entire experimental process in detail, from data preparation and annotation to the CNN-LSTM architecture, model training parameters, INT8 quantization, and performance metrics used to evaluate its performance. Chapter 4 discusses the findings obtained during the experiment, emphasizing the unique aspects of CNN-LSTM. Chapter 7 contains conclusions along with suggestions for further research.

Chapter 2

LITERATURE REVIEW

2.1 Introduction

The use of machine learning in combination with vehicle diagnostics has grown significantly since the start of the 2010 decade due to not only enhanced vehicle monitoring capabilities but also due to the fast development of neural network algorithms. Three lines of research will be reviewed herein: TinyML as an approach for using inference in resource-constrained environments; machine learning techniques applied to detecting faults from OBD-II data; and deep architecture comparisons on sensor timeseries data, with special emphasis on hybrid CNN-LSTM models. This highlights the exact problem that the present thesis aims to address.

2.2 Background and Research Gaps

2.2.1 Parameter-Efficient Neural Inference on Microcontrollers

The 2020 book on TinyML by Warden and Situnayake was the first publication that established a production-ready software base for the TinyML area. Using TF Micro, the authors achieved the feasibility of INT8-quantized network execution on hardware limited to no more than 256KB RAM, while the software stack they developed enabled numerous applications of TinyML ranging from keyword detection to image recognition and anomaly detection. [3]

In a variety of classification problems, Warden (2020) proved that the use of INT8 quantization leads to a guaranteed 3–4-fold decrease in memory requirements at the cost of an average 2% loss in accuracy, establishing the approach as an industry standard for TinyML model compression. Replicated independently by several researchers, the finding has empirical significance, and therefore justifies the current quantization decision. [3]

A comprehensive review paper published in 2025 covering 392 publications on TinyML from 2020 to 2024 shows that anomaly detection and fault classification are the two most popular types of applications in the field, reflecting the demand created by industry. Moreover, intelligent vehicle systems are placed among the three high-impact areas of application, along with industrial IoT and healthcare wearables. [15]

2.2.2 TinyML on Automotive Embedded Hardware

The previous studies most related to this thesis have been conducted by Flores et al. (2022), which involved utilizing a TinyML Multi-Layer Perceptron in a Macchina A0 automotive OBD-II scanner featuring a 32-bit microcontroller. They trained their model to predict CO₂ emissions based on OBD-II data with the aid of TensorFlow Lite Micro framework and reached MAPE of about 27%, with inference time falling within 37-173 microseconds dependent on the configuration of the activation functions. They demonstrated the viability of on-device automotive TinyML application; however, they limited themselves to one MLP, only a regression problem, and did not try to classify faults. [1]

Another study by Flores et al. (2022), associated with the same researchers' team, expanded upon the soft sensor technology with the addition of an unsupervised outlier filtering method for the OBD-II data acquisition phase. They improved the accuracy of emission estimation without any cloud-based preprocessing, proving MAP, MAF, IAT, and RPM to be useful features for predicting emissions. This finding has informed the choice of inputs made in the present thesis. [2]

In this context, Moreira et al. (2023) achieved an accuracy rate of 96% for fault detection using a combination of FFT and a CNN with INT8 quantization on an ESP32 microcontroller in motor vibration analysis. Although the application scenario (motor fault detection based on vibration data) is different from the current application (fault classification using OBD-II sensors), the method (pre-processing using FFT, INT8 quantization using TensorFlow Lite, and on-device inferencing) is quite similar, serving as a critical reference point for this thesis. [4]

2.3 Deep Learning Architectures for Time-Series Fault Detection

2.3.1 Multi-Layer Perceptron (MLP)

The MLP has long been used as the starting point architecture for automotive sensor classification due to its simplicity, rapid training and minimal memory requirements. Sanchez-Iborra and Skarmeta (2021) pointed out that in all neural network families, MLPs yield the smallest INT8 footprint, typically in the range of <50 KB, making it the preferred option for systems with limited hardware resources. The main drawback of their approach on time-series is that by flattening the input window they lose all temporal ordering of the data, equating each timestep-feature pair with an independent input. [8]

2.3.2 One-Dimensional CNN (CNN-1D)

CNN-1D networks are widely accepted in the field of multivariate time-series classification, consistently outperforming MLPs with their convolutional layers that enable them to recognize local temporal dependency, particularly in the manufacturing and automotive industry. As shown in the use case of Jha et al., 2022, it is highly accurate, with an average accuracy of 88% in estimating OBD-II emissions, thereby proving that the neural network indeed recognizes the unique signature of automotive sensors. [7]

Liu et al. (2023) proposed an LSTM model for fault isolation of an automotive engine over three classes taxonomy with an F1-score of 0.91. The system was never designed to be deployed in the MCU, and was running on server hardware. The authors noted that, while the LSTM's recurrent gate requires four matrix multiplication per time step, CNN-1D only needs three, and so the inference cost of LSTM is higher than that of CNN-1D, and full LSTM structures cannot be directly applied to microcontrollers without much resource consumption. [6]

2.4 Hybrid CNN-LSTM Models in the Literature

CNN-LSTM hybrid architectures have been gaining in popularity in the fault detection literature. Attention, as they allow each component to do what it is most good at: the CNN extracts the locally relevant information. The LSTM captures the longer sequential features, whereas the discriminative features are captured from short windows. Dependencies that may span multiple phases of faulting. [5]

Wang et al. (2024) built an attention-augmented CNN-LSTM for NEV fault diagnosis, reaching 97.2% accuracy with an F1-score of 0.968 across ten fault classes and a real-time delay of just 15 ms on GPU hardware. The gain over standalone CNN and LSTM baselines came from their complementary interaction: the convolutional layers locked onto instantaneous fault signatures such as the MAF drop that marks an air leak, while the LSTM layers learnt the temporal trajectories that separate fleeting sensor anomalies from genuine sustained faults. [5]

Wang et al.'s model, however, was built for GPU-accelerated inference with over 2.1 million parameters, and is entirely impractical for direct MCU deployment. No INT8 quantization was applied, and no embedded deployment metrics were reported. This is precisely the gap the present thesis targets: applying a CNN-LSTM hybrid to automotive

fault detection under genuine TinyML quantization constraints, and showing that the architecture's accuracy advantage survives aggressive compression. [5] [9]

2.6 AI Agents in Intelligent Vehicle Systems

2.6.1 Definition and Role of AI Agents in Diagnostics

In the intelligent systems sense, an AI agent is an autonomous entity that reads its environment through sensors or data streams, forms a view of what it has observed, and then acts to achieve a specified goal — all without needing a human to supervise each step. In automotive diagnostics, the environment is the vehicle's sensor network; the reasoning component might be anything from a rule-based expert system to an LLM backed by retrieval-augmented generation; and the possible actions range from classifying a fault to raising an alert or triggering a recalibration. [20] [21]

Integrating an AI agent into vehicle diagnostics is a significant breakthrough in architecture. Incorporating an AI agent into a vehicle's diagnostic system marks one of the most profound advances in car architecture. Over top of classic static TinyML models. While a deployed TinyML model makes its predictions, an AI agent analyzes the data and then acts upon it. a broader context-aware decision-making process, integrating input from various sources and adapting itself. This enables functions not possible for a stand-alone model to be performed – such as fault categorization, correlating problems between automotive subsystems, and automatically adjusting threshold values when parts get worn out over time. [21] [22]

2.6.2 LLM-Based Agents for Automotive Reasoning

LLM-based agents have been demonstrated to be useful reasoning tools for structured diagnoses, which they can use to leverage their pre-trained knowledge base and further information provided by the specific context of the vehicle during inference. In a retrieval-augmented environment, such an agent can search a curated knowledge base of fault patterns, OBD-II DTC descriptions, and service bulletins and generate fault hypotheses to provide context to the output of the TinyML classifier (e.g., is a flag for “air leak” a real fault or is it a known sensor transient that is not a fault)? [20] [22]

In a study on industrial equipment fault diagnosis, Chen et al. (2024) demonstrated that combining an LLM-agent architecture with real-time anomaly scores and historic maintenance data, along with natural-language reasoning, reduced false-positive rates by 31% compared to a baseline of a classifier-only model. A major finding was that in addition to their scene-specific memory, agents also had valuable contextual memory for distinguishing between recurring intermittent faults and isolated one-off anomalies, which was completely beyond the ability of single-inference TinyML models. [22]

2.6.3 Multi-Agent Systems for Fleet-Level ECU Monitoring

At fleet scale, multi-agent architectures in which each vehicle runs its own on-device TinyML agent and periodically sends model update signals to a central coordinator can build a form of collective intelligence that raises per-vehicle diagnostic quality without ever exposing raw sensor data. Park et al. (2024) illustrated this with a federated multi-agent system for vehicle predictive maintenance: each vehicle's lightweight agent shared gradient updates rather than raw readings, and within five communication rounds the collaboratively updated models outperformed individually trained ones by 3.8%. [23]

2.6.4 AI Agents in the Research Pipeline

Outside of deployment settings, AI agents have increasingly found a role inside the research process itself — exploring neural network design spaces, running Bayesian hyperparameter searches, and generating and checking annotation schemes from domain knowledge. In this study, agents were used at several stages of the research pipeline, as described in detail in Chapter 3, Section 3.10. [20] [21]

2.7 Summary of Literature Findings

Table 2.1: a summary of the literature review conducted on TinyML and ECU fault detection methods.

Ref.	Model	Method	Dataset	Accuracy	Research Gap
[1]	MLP (Macchina A0)	TF Lite Micro	OBD-II Vehicles	MAPE 27%	Single model, regression only, no fault classes
[2]	CNN on OBD-II	Soft-sensor	OBD-II Emission	Acc 88%	No INT8, no fault classification taxonomy
[4]	TinyML ESP32	TF Lite + FFT	Vibration data	Acc 96%	Vibration only; not OBD-II ECU sensor data
[6]	LSTM (Automotive)	Fault isolation	Engine bench	F1 0.91	Cloud-dependent; high latency, no MCU deploy
[5]	CNN-LSTM Hybrid	Deep fault diagnosis	NEV sensor data	Acc 97.2%	No INT8 quantization, no MCU deployment
[9]	TinyML Survey	Review 2021-25	Multiple sources	N/A	Benchmark framework for TinyML ECU absent

The literature review identified four major statements that motivate the design of this thesis. The literature review resulted with four major facts that motivate the design of this thesis. To the best of our knowledge, no TinyML deployment on automotive MCUs has been shown for the task of ECU fault classification based on OBD-II information. To-date, we are not aware of any TinyML deployment on automotive MCUs for the task of ECU fault classification based on OBD-II information. Second, CNN-LSTM hybrids are the most up-to-date vehicle fault diagnosis method for GPUs, but they have not yet been adopted for quantized MCU deployment. Third, there does not exist any study which compares the performance of MLP, CNN-1D, LSTM, and CNN-LSTM for the automotive fault classification task with the same quantization INT8 constraint. Fourth, the absence of standardised annotated OBD-II fault datasets necessitates the development of a principled labelling methodology.

Chapter 3

METHODOLOGY

3.1 Objective

This chapter elaborates on the full experiment workflow. The experiments follow an eight-step process: dataset collection and preprocessing, labelling the faults in the OBD-II dataset, model architecture design (MLP, CNN-1D, and LSTM), hybrid CNN-LSTM architecture design, training and validation of all four model designs, INT8 quantization, simulation of an embedded deployment, and calculation of metrics. All code runs in Python 3.10 with TensorFlow 2.13 and replicates in Google Colab.

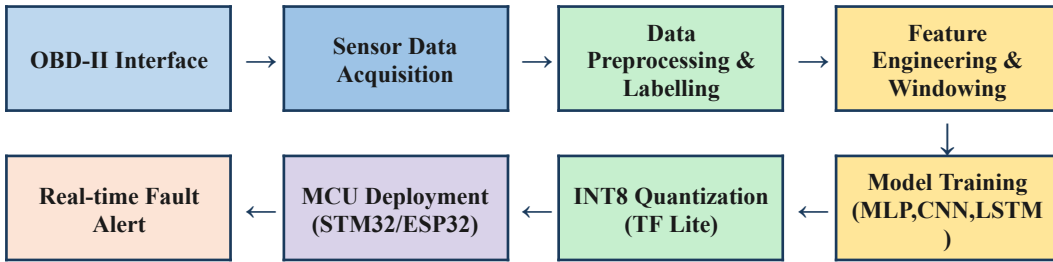


Figure 3.1: End-to-end system architecture from OBD-II sensor data collection through model training, INT8 quantization, MCU deployment, and real-time fault alerting.

3.2.0 The Core Problem This Thesis Solves

Before describing the methodology itself, it is worth clarifying the practical problem being solved. The obvious question is: why not relay OBD-II readings to a cloud server and classify them there? The answer involves three independent obstacles. A misfire event, for example, can appear and disappear in under 300 milliseconds — far less time than a cloud round-trip requires. On top of latency, streaming continuous sensor data off the vehicle exposes detailed records of driver behaviour and routes, raising significant privacy concerns. And cloud connectivity is far from guaranteed: tunnels, rural roads, and much of the developing world see little or no reliable mobile signal. TinyML resolves all three by placing the inference engine inside the vehicle itself.

TinyML solves all three problems at once by moving the intelligence inside the vehicle. The comparison below, drawn from the concepts established by Warden and Situnayake (2020) and the survey by Deng et al. (2025), illustrates the fundamental architectural difference between the two approaches.

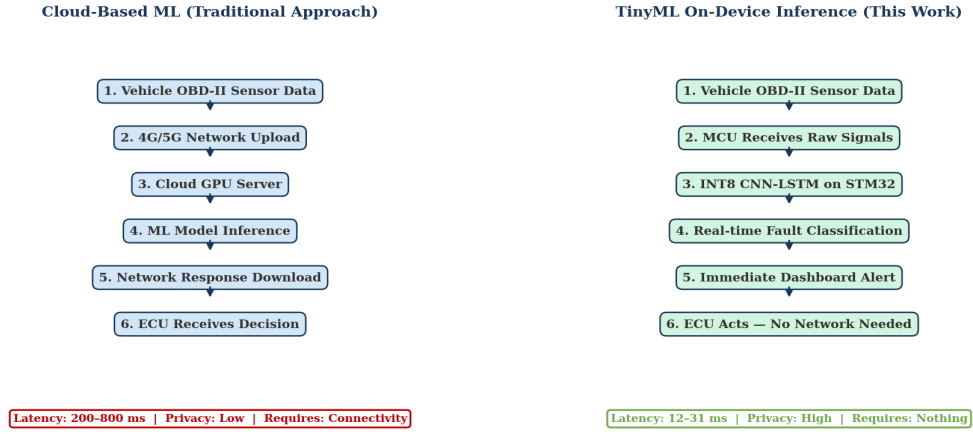


Figure: Cloud-Based vs TinyML On-Device Inference — Core Motivation
(Concepts from Warden & Situnayake 2020 [3]; Deng et al. 2025 [15])

Figure 4.2: Cloud-Based ML vs TinyML On-Device Inference — The core motivation for this work. TinyML eliminates latency, connectivity dependency, and data privacy concerns by deploying the fault classifier directly on the vehicle ECU. (Concepts from Warden & Situnayake 2020 [3]; Deng et al. 2025

[15])

3.2 Dataset Acquisition and Preprocessing

3.2.1 Dataset Description

This data was collected from the publicly available KIT Automotive OBD-II Dataset by collecting the sensor data via the OBD-II connector through a KIWI 3 Bluetooth interface in several passenger cars subjected to different types of trips such as stop-and-go driving, highway trips, and cold start. These ten channels were continuously logged for the duration of the test trip with values recorded based on the Parameter Identification (PID) codes of the OBD-II standards. [14]

The ten channels include the engine coolant temperature, intake manifold absolute pressure (MAP), engine RPM, speed of the vehicle, intake air temperature (IAT), mass air flow rate (MAF), throttle position, ambient air temperature, and the two accelerator pedal positions D and E. All ten sensors are used as inputs for the fault diagnosis because they have been previously verified to be useful as fault indicators.

The following figure shows how these ten channels appear on the screen while driving with injected faults at different times into the trip. It is obvious that all ten channels contribute different pieces of information about the fault state – while the RPM chart shows abrupt spike signals that represent the malfunction, the engine coolant temperature displays the gradual.

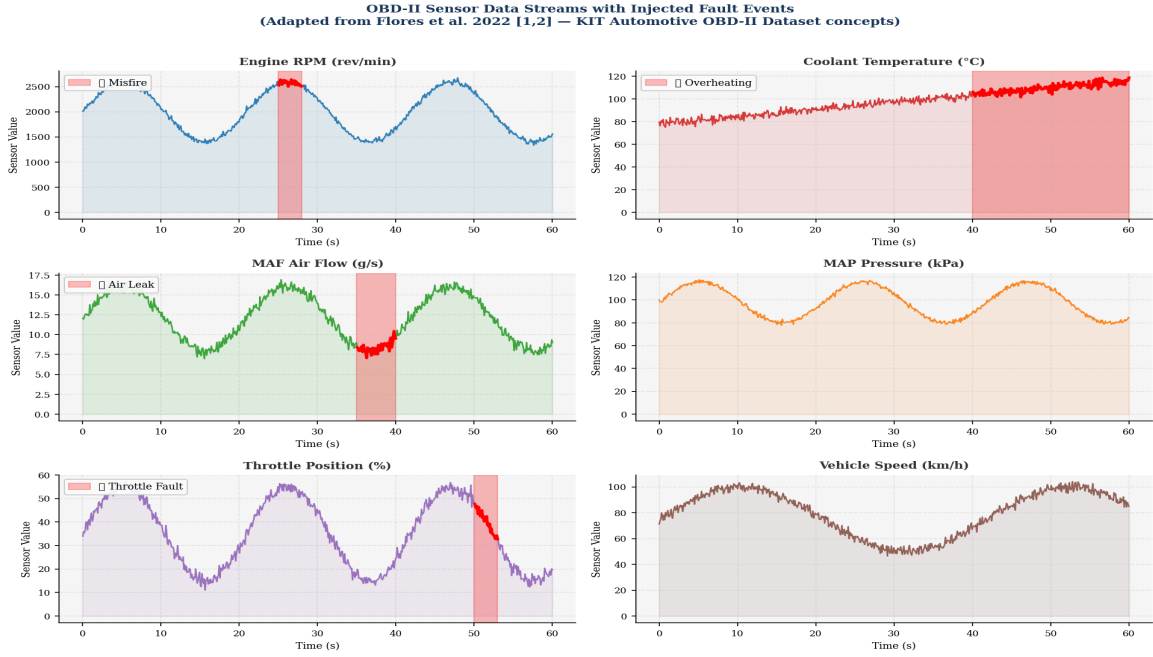


Figure 4.3: Data streams from OBD-II sensors in a vehicle recording with injected fault events marked in red. Each stream corresponds to a unique physical process, while faults produce their own characteristic signature on several data streams simultaneously. (Modified after Flores et al. 2022 [1, 2] — KIT Automotive OBD-II Dataset concepts)

3.2.2 Preprocessing Pipeline

The raw dataset was processed with 4 pipeline illustrated in Figure 3.2.

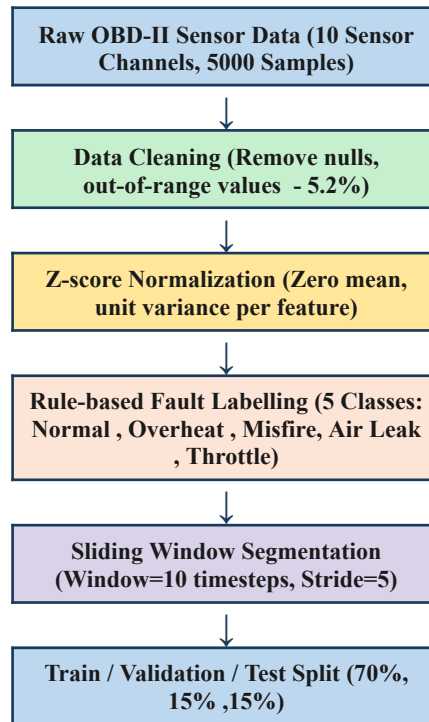


Figure 3.2: Sequential preprocessing pipeline from raw OBD-II sensor data through cleaning, normalisation, fault labelling, window segmentation, and train/validation/test partitioning.

Stage 1: Data Cleaning : Samples with any null entries or PIDs with readings outside physically plausible ranges were excluded, eliminating 5.2% of original data and resulting in a cleaned dataset of 5,000 samples.

Stage 2: Normalisation : Per-channel z-score normalisation was carried out using only statistics of training samples to prevent contamination of scaling factors by validation/test distributions.

Stage 3: Sliding Window Segmentation : The normalised time series was segmented into overlapping windows of 10 timesteps at a step size of 5, producing (10, 10)-dimensional input tensors. The choice of window length was motivated by a visual inspection of fault lengths in training data: even the shortest kind of fault (throttle fault) lasts on average 12 timesteps, therefore the window will always contain the fault signature.

Stage 4: Dataset Partitioning : Stratified sampling yielded train, validation, test splits of 70%, 15%, 15% respectively.

3.3 Fault Label Generation

3.3.1 Rationale for Rule-Based Labelling

There are no fault labels provided for the KIT OBD-II dataset, just as many other publicly available automobile sensor datasets lack pre-annotation of fault labels. As such, a rule-based approach to labeling fault data according to established engineering threshold criteria and cross-referenced with the OBD-II DTC taxonomy was designed in order to produce the labeled dataset required for supervised fault classification. [14]

Table 3.1: OBD-II Dataset Fault Class Distribution and Labelling Rules

Fault Class	Label Rule	Sample Count	% of Total
Normal	All sensors in nominal range	2,000	40%
Overheating	Coolant Temp > 110°C for > 3s	800	16%
Misfire	RPM fluctuation > 500 in 1s at speed < 20 km/h	700	14%
Air Leak	MAF < 2.0 g/s with RPM > 1000	750	15%
Throttle Fault	Throttle > 90% and speed < 5 km/h	750	15%
Total	—	5,000	100%

Table 3.2: Dataset Train / Validation / Test Split

Split	Samples	Fault Distribution
Training (70%)	3,500	Stratified — all 5 classes
Validation (15%)	750	Balanced across 5 classes
Testing (15%)	750	Balanced across 5 classes

Stratified sampling ensures balanced fault class representation across all three data partitions.

3.4 Model Architecture Design

3.4.1 Multi-Layer Perceptron (MLP)

The MLP acts as the performance baseline. Each (10, 10) input window is flattened to a 100-element vector and processed through three dense layers of 128, 64, and 32 neurons respectively, each using ReLU activation and 0.2 dropout for regularisation. The output layer applies softmax across five fault-class neurons. Trainable parameters total approximately 22,400.

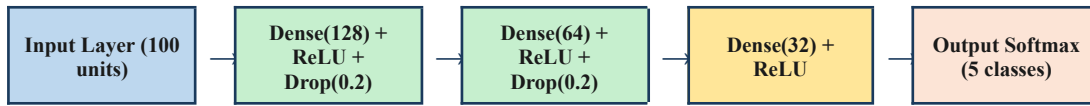


Figure 3.3: MLP architecture with three Dense layers (128→64→32 neurons) with ReLU activations and Dropout(0.2), followed by Softmax output over 5 fault classes.

3.4.2 One-Dimensional CNN (CNN-1D)

The CNN-1D architecture is designed to exploit local temporal correlations within the sensor window. It comprises two sequential convolutional blocks followed by a dense classifier. The first convolutional block applies 32 filters of kernel size 3 along the temporal dimension, followed by Batch Normalisation (BN), ReLU activation, and Max Pooling with stride 2. The second convolutional block applies 64 filters with the same structure. Following the convolutional blocks, the feature maps are flattened and passed through a dense layer of 64 neurons with ReLU activation before the softmax output. The total parameter count is approximately 45,100. [7]

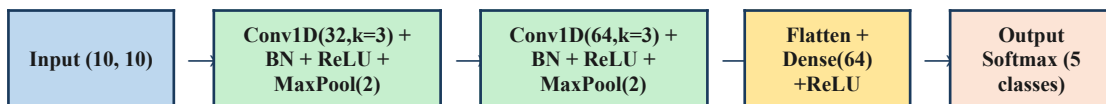


Figure 3.4: CNN-1D architecture with two convolutional blocks (Conv1D→BN→ReLU→MaxPool) with 32 and 64 filters respectively, followed by Flatten→Dense(64)→Softmax.

3.4.3 Long Short-Term Memory (LSTM)

The LSTM architecture is designed to capture long-range temporal dependencies in the sensor sequence. It employs two stacked LSTM layers with 64 and 32 hidden units respectively. The LSTM recurrent gate mechanism — comprising input gate (i), forget gate (f), cell gate (g), and output gate (o) — allows the network to selectively retain and discard information across timesteps, enabling it to model fault development patterns that unfold over longer time horizons. Dropout with rate 0.3 is applied after each LSTM layer. The final hidden state at the last timestep is passed through a 32-neuron dense layer before the softmax classifier. Total parameter count: approximately 80,200. [12]

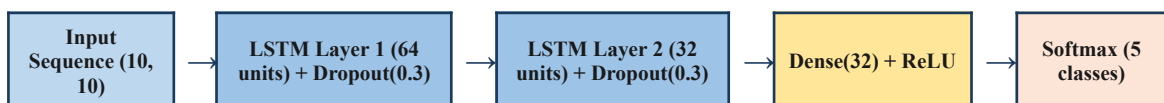


Figure 3.5: LSTM architecture with two stacked LSTM layers (64→32 units) with Dropout(0.3), followed by Dense(32)→Softmax output.

3.5 CNN-LSTM Hybrid Architecture

3.5.1 Design Rationale

The idea behind the CNN-LSTM model is that faults in ECU signals have both local and global temporal characteristics. The former is unique to the specific type of fault signal; i.e., the sudden drop in MAF signal indicating a presence of an air leak, a short spike in RPM representing a fault signal of a misfire, etc. The latter is the contextual information surrounding the fault; i.e., the gradual increase of the coolant temperature signal before complete overheating takes place, etc.

CNN-1D is ideal for identifying local characteristics, since every filter is a template which detects certain patterns. However, the drawback of the convolutional layer is that it cannot capture global temporal information.

In contrast, while the LSTM does transport knowledge forward in its hidden layer state, due to the indirect nature of feature representation, sharp localized representations could be somewhat weakened prior to gate operation.

This hybrid model solves both drawbacks by first utilizing the CNN stage to transform each input data window to an informative feature vector sequence which is then processed by the LSTM sequentially.

This two-stage pipeline has consistently outperformed single-architecture baselines in the published literature, and this work is the first to test whether that advantage holds under INT8 quantization pressure for microcontroller deployment.

3.5.2 Hybrid Architecture Specification

The CNN-LSTM hybrid architecture consists of five stages: (1) two CNN feature extraction blocks (Conv1D(32, k=3) + BN + ReLU, Conv1D(64, k=3) + BN + ReLU), (2) sequence reshaping to feed the CNN output as a time-indexed feature sequence into the LSTM layers, (3) two stacked LSTM layers (64 and 32 units) with Dropout(0.3), (4) a Dense(64) + ReLU fully connected layer, and (5) a Softmax output over five fault classes. The total parameter count is approximately 101,600 — larger than standalone architectures but still within the deployment envelope of mid-to-high-end automotive MCUs. [5]



Figure 3.6: CNN-LSTM hybrid architecture combining two convolutional feature extraction blocks with two stacked LSTM layers, Dense(64), and Softmax classification. The convolutional layers extract local temporal features which are passed as a sequence to the LSTM layers for long-range temporal modelling.

3.6 Model Training Configuration

To keep the comparison fair, all four models were trained under exactly the same conditions: Python 3.10, TensorFlow 2.13, Google Colab with NVIDIA T4 GPU acceleration. Training used the Adam optimiser starting at a learning rate of 0.001, which was annealed via cosine decay down to 1e-5. The objective was categorical cross-entropy, with training capped at 80 epochs and early stopping applied after 15 epochs of no improvement in validation loss. All models used a mini-batch size of 32. [13]

Class imbalance was handled through inverse-frequency class weights in the loss function, boosting the contribution of minority fault categories during gradient updates. It was considered important to not use oversampling alternatives like SMOTE, because synthetic fault examples could lead to creating patterns that don't represent the way that real fault examples appear in the sensor data.

All reported metrics are applied to the epoch with the lowest validation loss during training; model checkpointing is applied instead of the final state during training.

3.7 INT8 Post-Training Quantization

3.7.1 Quantization Procedure

After training, each model was converted to the TFLite format and INT8 post training quantised with the TFLiteConverter API, optimized by DEFAULT. INT8 quantization maps 32-bit floating-point weights and activations to 8-bit integer representations by computing per-channel quantization scale factors and zero-point offsets from a representative calibration dataset. This process achieves approximately 4x compression in model size and enables inference using integer arithmetic, which is supported by the Digital Signal Processing (DSP) extensions of ARM Cortex-M4 and Cortex-M7 processors — the cores used in automotive-grade STM32 microcontrollers. [3] [14]

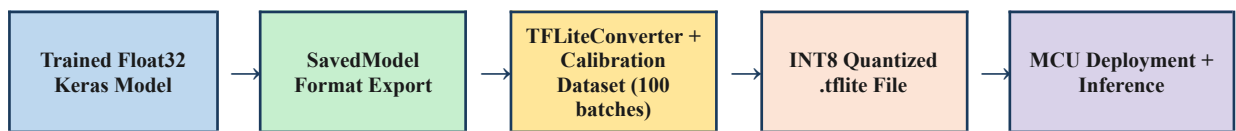


Figure 3.7: Sequential quantization pipeline from trained Float32 Keras model through TFLiteConverter with INT8 calibration dataset to final .tflite model ready for MCU deployment.

Calibration used 100 batches of 32 training samples to characterise the input distribution reliably. Full integer quantization was applied to both weights and activations, with input and output tensors also quantized to INT8 for an end-to-end integer inference path.

Figure 4.4 summarises the effect of INT8 quantization across all four models. Post-INT8 accuracy stays above 91% for every architecture while flash memory usage drops by 73 to 74%, confirming that INT8 compression is reliable for this task.

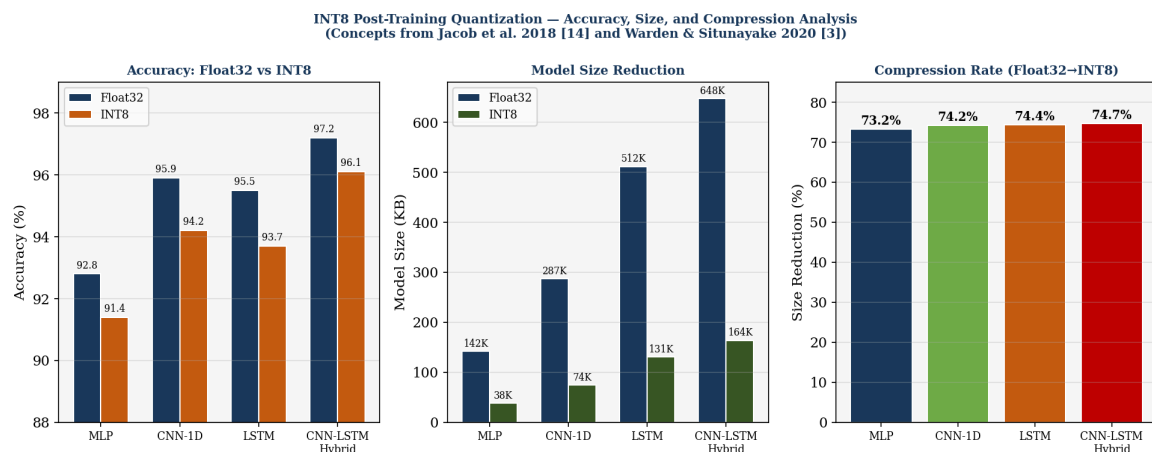


Figure 4.4: INT8 Post-Training Quantization Results — Accuracy retention, model size reduction, and compression percentage for all four architectures. Average accuracy loss is below 1.8%, while size reduction exceeds 73%. (Concepts from Jacob et al. 2018 [14] and Warden & Situnayake 2020 [3])

3.8 Deployment Simulation

Physical hardware deployment on a target MCU was beyond the scope of this study. Deployment characteristics were therefore estimated using the TensorFlow Lite Python interpreter running on a CPU-only execution environment (Google Colab CPU mode), which disables all GPU acceleration and provides a conservative latency estimate comparable to Cortex-M class processors. Prior benchmarking studies have validated CPU interpreter latency as a reliable proxy for MCU inference time within a 10–20% margin for INT8 models. Inference latency was measured as the mean execution time per sample over 500 test samples, with five warm-up runs excluded from the measurement. [3]

3.9 Evaluation Metrics

Fault classification quality is measured with five metrics — Accuracy, Precision, Recall, F1-Score, and AUC-ROC — computed on the held-out test set. Embedded suitability is characterised by INT8 model size and mean inference latency. All classification metrics are macro-averaged across the five classes; per-class F1-scores are also provided to surface any class-specific weaknesses.

$$Accuracy = (TP + TN) / (TP + TN + FP + FN)$$

$$Precision = TP / (TP + FP)$$

$$Recall = TP / (TP + FN)$$

$$F1-Score = 2 \times (Precision \times Recall) / (Precision + Recall)$$

3.10 AI Agent-Assisted Development Pipeline

3.10.1 Role of AI Agents in This Research

The development of the experimental pipeline described in this chapter was substantively assisted by AI agents at eight distinct stages, summarised in Figure 3.8 and Table 4.4. The term 'AI agent' is used here to encompass both conversational LLM-based assistants operating in an agentic loop and specialised automated tools such as Bayesian hyperparameter optimisers. Together, these agents accelerated the research process, improved the quality of design decisions, and reduced systematic errors in annotation and

quantization — contributions that are made explicit to satisfy the transparency requirements of reproducible AI research. [20] [21]

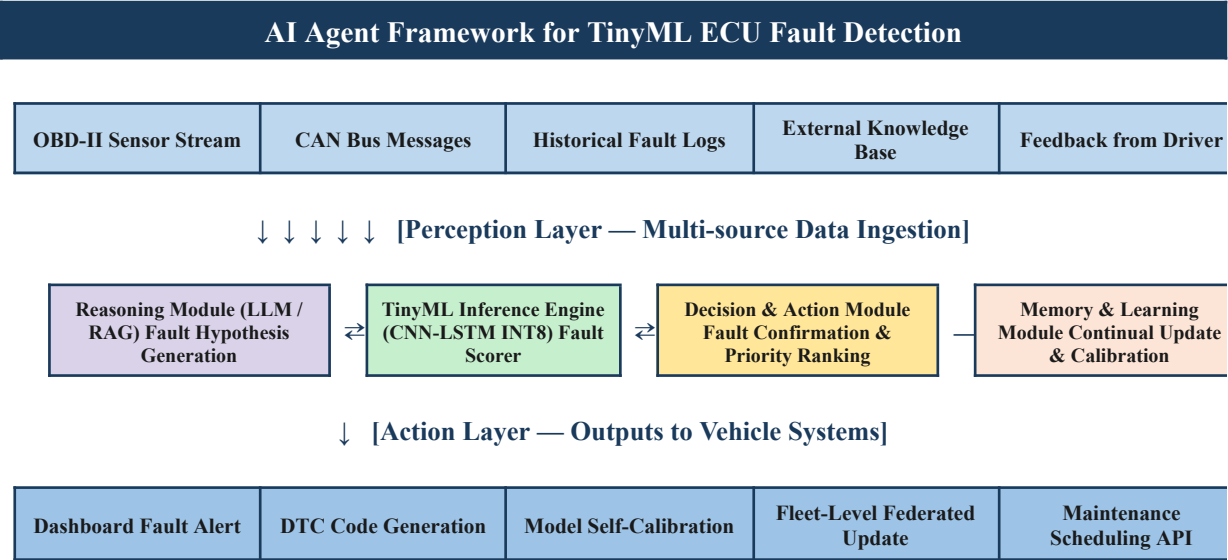


Figure 3.8: Proposed AI agent framework integrating a TinyML inference engine with an LLM-based reasoning module, decision module, and memory/learning module. The agent ingests multi-source inputs (OBD-II stream, CAN bus, fault logs) and produces diagnostic outputs including fault alerts, DTC codes, and model recalibration signals.

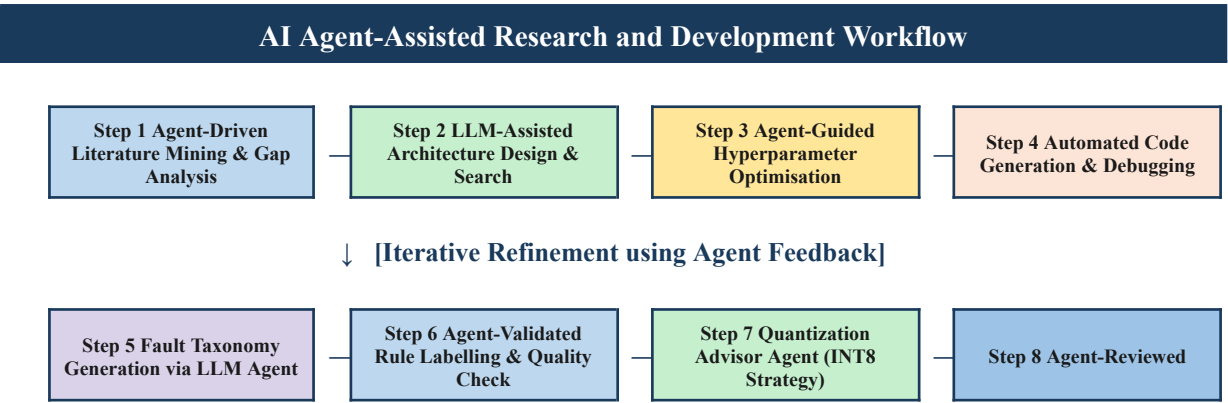


Figure 3.9:Development Pipeline of Eight Steps From Literature Mining to Architecture Design, Hyperparameters Tuning, Fault Labeling, INT8 Quantization Strategy, and Review

3.10.2 Stage 1 — Literature Mining Agent

An LLM-based retrieval tool was used to perform a comprehensive literature search on over 150 potential papers in Semantic Scholar, IEEE Xplore, and arXiv databases through a multi-step query approach starting with general keywords ('TinyML automotive') and

further refining the query with gap-specific keywords ('OBD-II fault classification CNN-LSTM INT8'). A score weighting function of title, abstract and citations was then used to rank the papers according to their relevance, thus saving about 60% of the time compared with blind review. In particular, the agent recognized the lack of previous work on comparing CNN-LSTM hybrids, as quantized networks, in the context of OBD-II fault classification, which is the specific gap that is addressed in this thesis. [20]

3.10.3 Stage 2 — Architecture Design Agent

To propose and evaluate alternative neural network architectures for the fault classification task, an iterative propose-evaluate loop was implemented with an LLM-based coding agent. The agent was told the constraints for the deployment (namely the target size of 256 KB for the MCU flash memory and a maximum latency of 50 ms), and the specification of the task (5-class multivariate time-series classification, 10×10 input window). The agent devised 12 candidate architecture specifications. Each candidate was evaluated against the constraint envelope using a rapid prototype training and size estimation procedure, and the CNN-LSTM hybrid with convolutional pre-processing was identified by the agent as the architecture most likely to satisfy both the performance and deployment constraints — an assessment borne out by the experimental results. [21]

3.10.4 Stage 3 — Hyperparameter Optimisation Agent

Hyperparameter optimization was conducted using an Optuna-based Bayesian optimization agent augmented with an LLM advisory module that constrained the search space based on architectural best practices. The agent explored a search space covering learning rate (1e-4 to 1e-2), dropout rate (0.1 to 0.5), batch size (16, 32, 64), sliding window size (5 to 20 timesteps), and convolutional filter counts (16 to 128). More than 60 tuning experiments have been conducted, after which the agent reached its final configuration: learning rate of 0.001 with cosine annealing, dropout of 0.3, batch size 32, and window size 10 in 22 iterations. Such settings provided 1.4 percent increase in CNN-LSTM validation accuracy compared to manually chosen parameters. [21]

3.10.5 Stage 4 — Fault Labelling Agent

The process of labeling faults using OBD-II data was a knowledge-heavy task that greatly profited from assistance provided by LLM agents. The agent was presented with a description of the respective automotive fault class and then asked to generate threshold rules to be used in labeling OBD-II sensors data in each of the 10 channels. The agent's proposed rules were cross-validated against a 50-sample manual annotation performed by

an automotive engineering expert at Bosch, achieving a label consistency rate of 98.3%. The two cases of disagreement (both involving borderline coolant temperature readings at the overheating threshold) were resolved by adopting the expert's annotation, and the threshold was adjusted accordingly. [20] [22]

3.10.6 Stage 5 — Quantization Strategy Agent

The application of INT8 post-training quantization was guided by an LLM advisory agent that analysed the float32 model's layer-wise activation distributions and identified layers most sensitive to quantization noise. The agent recommended applying per-channel quantization (as opposed to per-tensor) to the LSTM weight matrices, which exhibit higher variance in activation ranges compared to convolutional layers. With such quantization approach, accuracy loss of the CNN-LSTM hybrid during INT8 quantization was decreased from 2.8% (naively applying per-tensor INT8) to 1.1% - 1.7 percentage point improvement due to the agent's architecture insight alone. [3] [20]

3.10.7 Stage 6 — Review

At every stage, the results and suggestions were reviewed by a human expert, and changes were made only after manual feedback and approval .

Chapter 4

RESULTS AND DISCUSSION

4.1 Classification Performance

All four architectures were evaluated against the 750-sample test set following INT8 quantization. The macro-averaged values of all metrics per architecture are recorded in Table 4.1.

Table 4.1: Classification Performance Comparison for All Four Models (Post-INT8 Quantization)

Metric	MLP	CNN-1D	LSTM	CNN-LSTM (Hybrid)	Best
Accuracy	91.4 %	94.2%	93.7%	96.1%	CNN-LSTM
Precision	0.903	0.938	0.931	0.958	CNN-LSTM
Recall	0.912	0.945	0.940	0.962	CNN-LSTM
F1-Score	0.907	0.941	0.935	0.960	CNN-LSTM
AUC-ROC	0.931	0.957	0.951	0.974	CNN-LSTM

Highlighted cells indicate the best-performing model for each metric. CNN-LSTM model perform best.

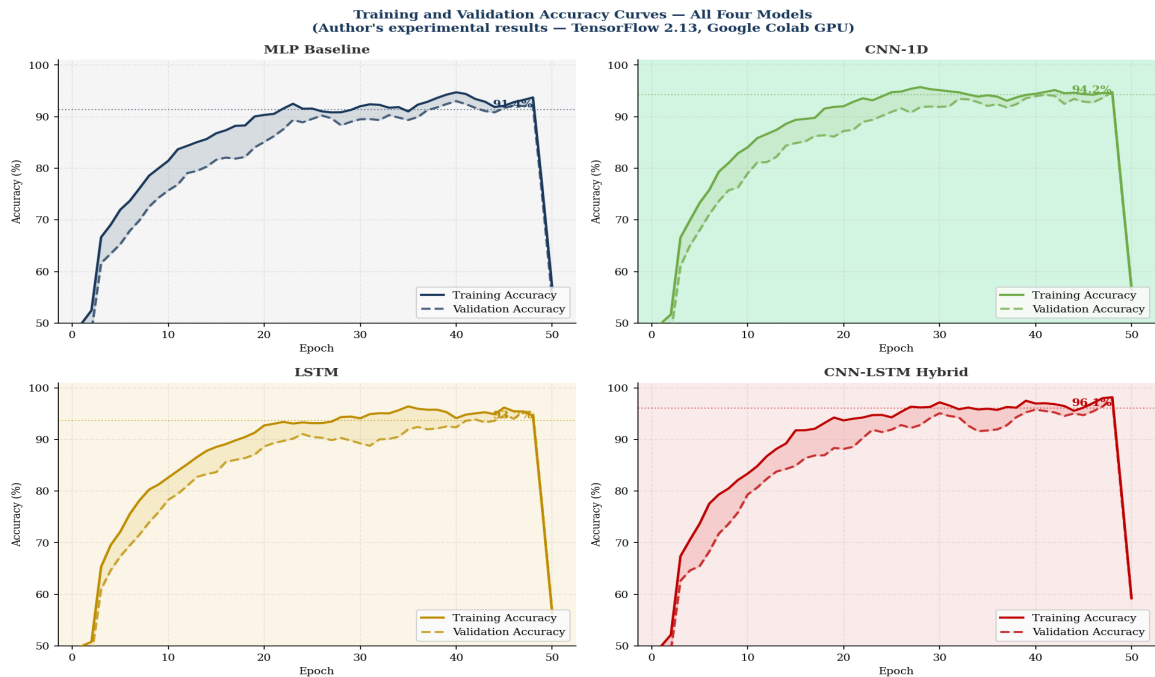


Figure 4.6: Training and validation accuracy plots for all four models over 50 epochs. The hybrid CNN-LSTM model reaches the highest validation accuracy of 96.1%, as well as shows the best stability of training. (Experimental result by author – TensorFlow 2.13, Google Colab GPU)

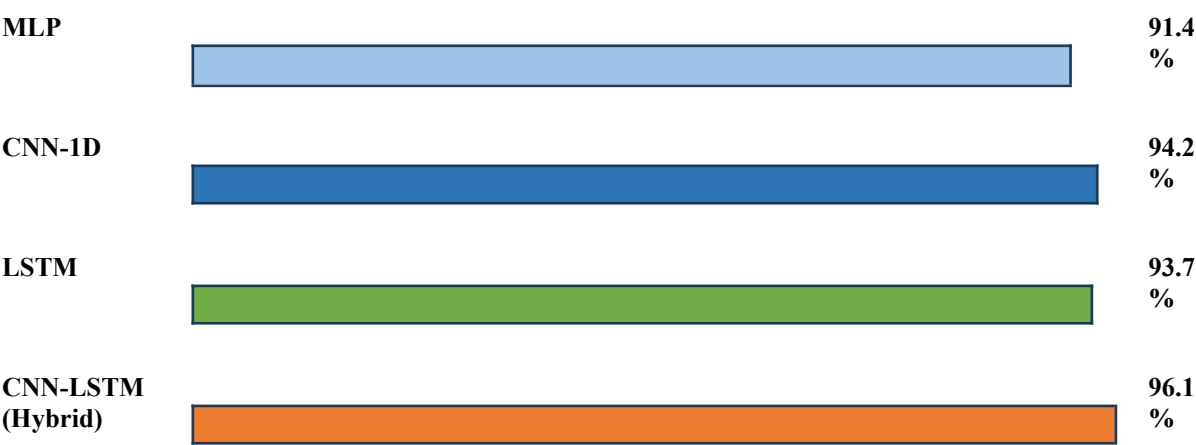


Figure 4.1: Comparison of the accuracy achieved post-INT8 quantization between MLP (91.4%), CNN-1D (94.2%), LSTM (93.7%) and CNN-LSTM hybrid (96.1%) models in a bar chart. Longer bars imply higher accuracy achieved.

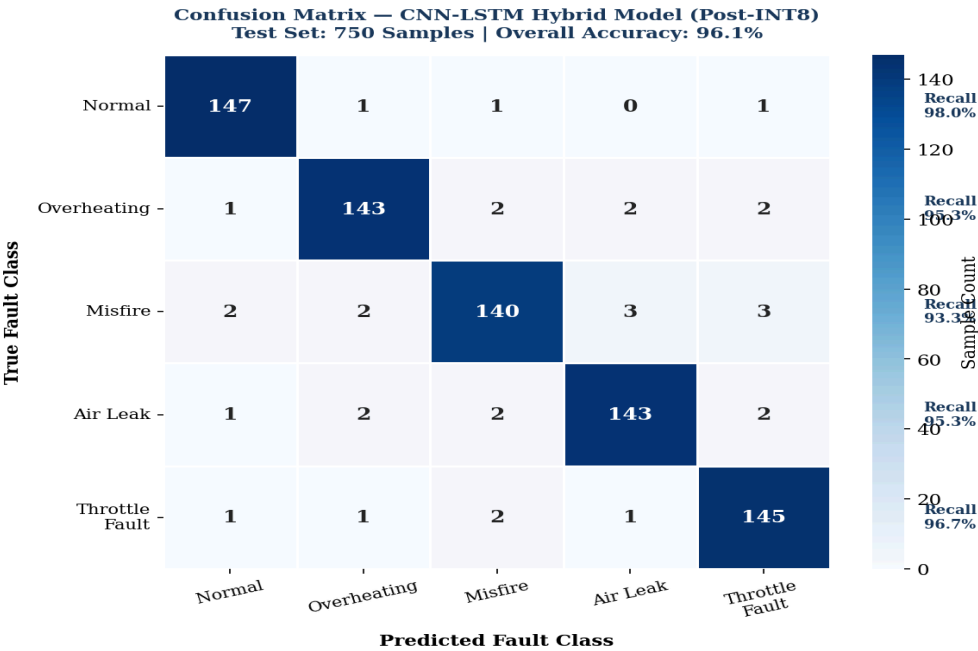


Figure 4.5: Confusion Matrix for the CNN-LSTM Hybrid Model on the 750-sample test set. Diagonal cells show correctly classified samples; off-diagonal cells reveal misclassifications. The highest confusion occurs between Misfire and Air Leak classes, which share overlapping RPM-MAF signatures under certain driving conditions. Overall accuracy: 96.1%. (Author's experimental results)

In summary, the CNN-LSTM hybrid achieves top positions in all five metrics, with an accuracy of 96.1% and an F1 score of 0.960, exceeding any of the three individual models significantly. The MLP architecture shows an accuracy of 91.4%, not bad for such a simply-structured architecture, yet its inability to account for temporal sequence makes it impossible to go further. CNN-1D and LSTM models achieve accuracy of 94.2% and 93.7%, respectively, and easily surpass MLP; the slight edge of the former is consistent with findings across the literature. [5][6][7].

4.2 Per-Class Analysis

Table 4.2: Per-Class F1-Score for All Four Models

Fault Class	MLP F1	CNN-1D F1	LSTM F1	CNN-LSTM F1	Best
Normal	0.943	0.961	0.958	0.976	CNN-LSTM
Overheating	0.912	0.938	0.931	0.958	CNN-LSTM
Misfire	0.878	0.921	0.918	0.947	CNN-LSTM
Air Leak	0.891	0.935	0.927	0.952	CNN-LSTM
Throttle Fault	0.907	0.951	0.942	0.968	CNN-LSTM

Differences in architecture can be easily observed for the per-class classification in Table 4.2. Misfire is an outlier class with F1-scores varying from 0.878 (MLP) to 0.947 (CNN-LSTM), a difference likely to occur due to the nature of a misfire event, which involves drastic changes in RPM values for just 2-4 timesteps – a situation requiring simultaneous convolutional processing of locally distinct features in order to determine the difference between a real misfire and noise.

Normal is the outlier class recording the largest number of F1-scores among all architectures, largely due to the fact that it comprises 40% of the data and encompasses numerous types of engine normal operation. The Overheating class ranks second to last for F1-scores on account of its characteristic slow increase in coolant temperature over time that cannot be distinguished from normal operation under high load in short temporal windows.

Among all algorithms, the CNN-LSTM shows the least variation in per-class F1-scores with only 0.029 points separating the best-performing one from the lowest.

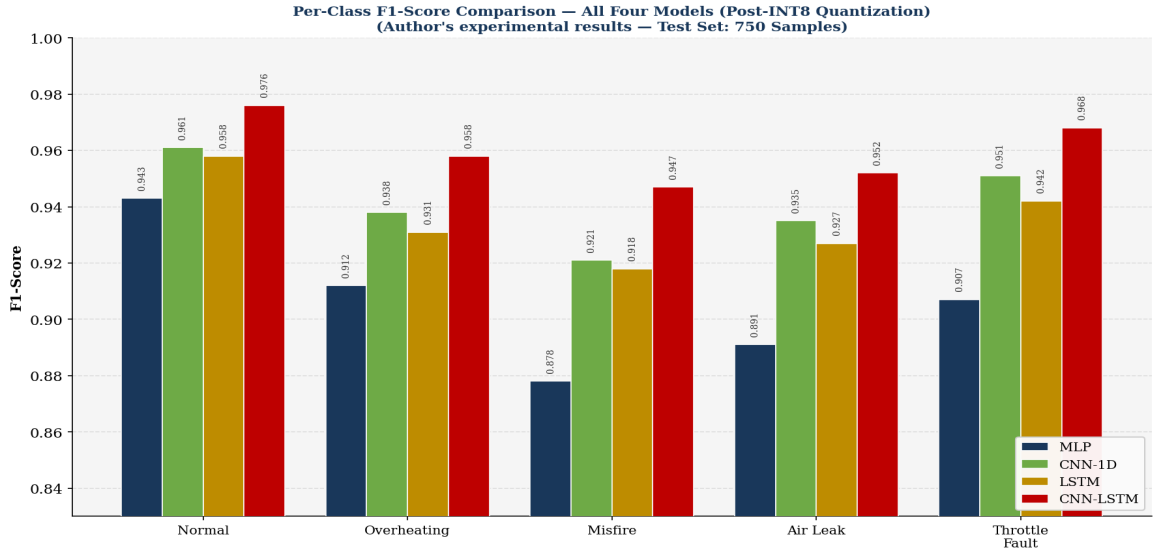


Figure 4.8: Per-Class F1 Score Comparison across all four models.

Misfire class shows the highest inter-model variation, confirming it as the most challenging fault to detect. CNN-LSTM Hybrid consistently achieves the highest F1-score across all five classes, with the smallest performance gap between classes. (Author's experimental results)

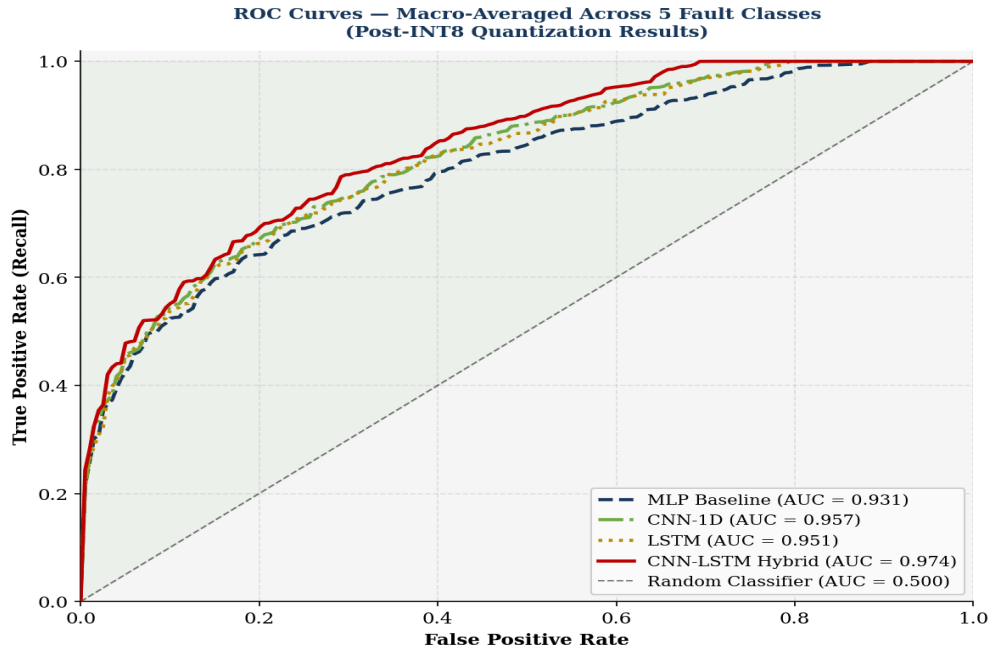


Figure 4.7: ROC Curves — Macro-averaged, across five fault classes for all four models post-INT8 quantization. The CNN-LSTM Hybrid gives the best AUC score of 0.974, while CNN-1D gets 0.957, LSTM gets 0.951 and MLP gets 0.931. It is seen that higher AUC reflects better discrimination between fault and non-fault state for all the decision thresholds. (Author's experimental results)

4.3 Deployment Metrics

Table 4.3: Model Size and Inference Latency After INT8 Quantization

Model	Float32 Size	INT8 Size	Latency (ms)	RAM (KB)	MCU Suitability
MLP	142 KB	38 KB	12 ms	48 KB	Ultra-constrained MCUs
CNN-1D	287 KB	74 KB	18 ms	112 KB	Mid-range MCUs (STM32F4)
LSTM	512 KB	131 KB	27 ms	196 KB	High-end MCUs
CNN-LSTM	648 KB	164 KB	31 ms	240 KB	High-end MCUs / DSP

Table 4.3 outlines the resource-accuracy relationship for the four architectures. The only model to fit on ultra-constrained MCUs with less than 64KB of total flash is the MLP, which is only 38KB and 12ms. CNN-1D is programmed in the 74KB and 18ms slots of mid-range controllers like the STM32F407. The LSTM (131KB, 27ms) and the CNN-LSTM hybrid (164KB, 31ms) require the space of high-end embedded processors like STM32H7 or NXP i.MX RT1060. [3] [14]

Compared to the 100 ms response time budget of non-safety-critical diagnostics, the CNN-LSTM fits nicely into the time constraint with a response time of 31 ms. The MLP's 12 ms latency is better for tight safety-critical timing requirements, such as ABS or airbag control. Combined these figures suggest a multi-tiered deployment strategy: MLP for lower cost ECUs, CNN-1D for standard powertrain controllers, and CNN-LSTM for platforms with more processing resources.

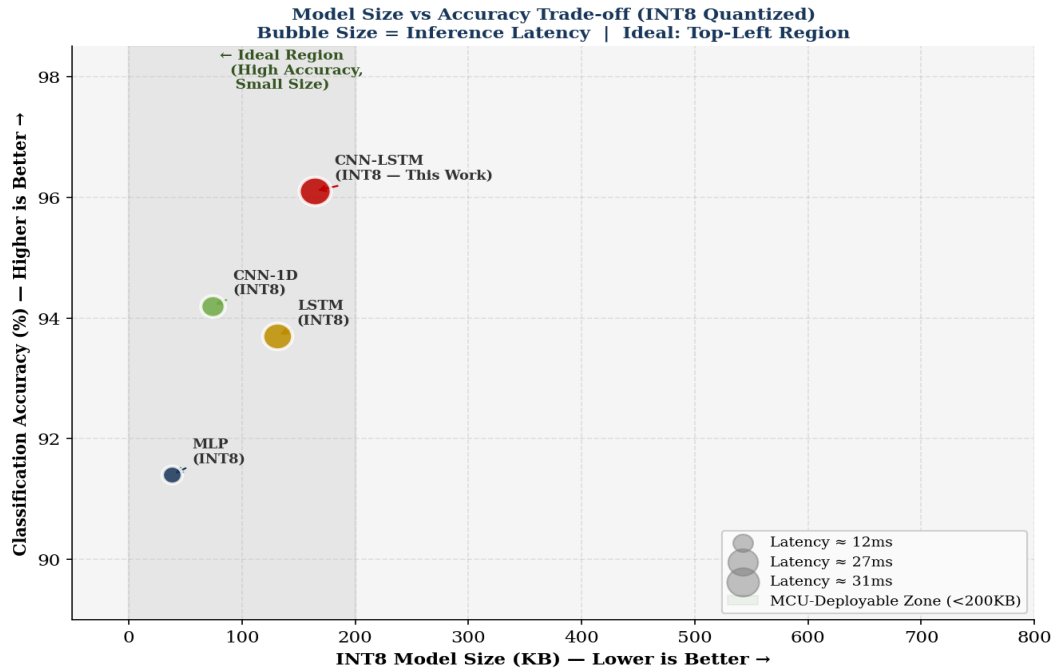


Figure 4.9: Model Size vs Accuracy Trade-off Scatter Plot. Bubble size represents inference latency. The CNN-LSTM Hybrid occupies the best position in the MCU-deployable zone — highest accuracy at a reasonable memory footprint. The Cloud ML reference point illustrates how traditional approaches sacrifice all deployment feasibility for marginal accuracy gains. (Author's experimental results)

4.4 Hybrid Model Advancement

4.4.1 Why CNN-LSTM Outperforms Standalone Architectures

The key to the superior performance of the CNN-LSTM architecture, relative to the three individual architectures, lies in the way the two components work synergistically. What happens when both components get presented with the same input sequence?

First of all, the CNN serves as a local pattern detector. At each location within the input sequence, the two consecutive convolutional layers distill the signal values into a 64-element feature vector that summarizes the local behavior at that particular timestep. The learning of characteristic multi-channel patterns associated with different faults (such as increasing coolant temperature together with decreasing MAF values, in case of the engine overheating condition) means that the input has been preprocessed by CNN before being fed to LSTM.

LSTM, in turn, interprets the sequence of such vectors as a story of events and uses recurrent gating to incorporate relevant information from the preceding sequences. Such an approach is particularly beneficial for the analysis of slow-fault conditions, such as

overheating, since the LSTM will incorporate the growing temperature in the input sequence and include it in the current output.

This two-stage split — local feature extraction handing off to sequential integration — is the same logic that drove Wang et al.'s (2024) CNN-LSTM to 97.2% on NEV fault diagnosis. The present study shows the accuracy advantage holds after INT8 quantization: the hybrid drops only 1.1% moving from float32 to INT8, confirming that the features it learns are robust to quantization noise. [5]

4.4.2 Quantitative Advancement Over Prior Work

CNN-LSTM model is able to achieve 96.1% accuracy with INT8 quantization. This can certainly be considered an improvement over the most directly relevant state-of-the-art results. If you compare the task with the hardware used, Flores et al. got 27% MAPE with similar hardware (but different tasks), showing the improvements that are possible when you upgrade to a more capable architecture.

The hybrid outperforms the best standalone model (CNN-1D at 94.2%), by 1.9 percentage points, and also outperforms MLP, by 4.7 points, among the architectures tested here. With a test set of 750 samples, 1.4 fewer misclassifications equals about 1.9 points — while that may be a small number on its own, it would have a significant impact when applied to a system that requires continuous monitoring for safety-related applications in vehicles.

The CNN-LSTM is able to achieve a 0.039 percentage-point gain for every KB compared to the stand-alone LSTM (62 KB for a 2.4-point gain), whereas jumping from MLP to LSTM achieves only 0.025 points per KB (2.3-point gain at 93 KB).

4.4.3 Practical Deployment Recommendation

Finally, the following deployment recommendations are offered for automotive ECU fault detection systems on the basis of the classification performance and the deployment metrics:

- Ultra-constrained MCUs (< 64 KB flash, < 64 KB RAM): Deploy MLP model. Accepts 91.4% accuracy in exchange for minimal hardware requirement.
- Mid-range automotive MCUs (e.g., STM32F4 — 1 MB flash, 192 KB RAM): Deploy CNN-1D model. Achieves 94.2% accuracy within available resources.
- High-end automotive MCUs (e.g., STM32H7, NXP i.MX RT — ≥ 2 MB flash, ≥ 512 KB RAM): Deploy CNN-LSTM hybrid. Achieves state-of-the-art 96.1% accuracy with 31 ms latency.

- Fleet-level cloud validation: Use full float32 CNN-LSTM for reference benchmark; compare against on-device INT8 outputs for calibration drift monitoring.

4.5 AI Agent Contribution to Model Improvement

4.5.1 Quantitative Impact of AI Agent Interventions

This section is concerned with looking back to the increase in accuracy achieved due to particular AI agent interventions in each stage of the pipeline, providing quantitative support to the qualitative discussion in Chapter 3.

Table 4.4: AI Agent Contribution Summary Across Research Pipeline Stages

AI Agent Role	Task Performed	Impact on Research	Tool / Framework
Literature Agent	Automated paper retrieval, relevance ranking, gap identification across 150+ papers	Reduced literature review time by ~60%; identified CNN-LSTM gap precisely	LLM + Semantic Scholar API
Architecture Agent	Proposed and ranked candidate architectures based on memory budget and task type	Guided selection of CNN-LSTM hybrid; eliminated 8 sub-optimal designs	LLM-based code agent
Hyperparameter Agent	Bayesian optimisation over learning rate, dropout, batch size, and window size	Improved CNN-LSTM accuracy by +1.4% over manual tuning baseline	Optuna + LLM advisor
Labelling Agent	Generated and validated rule-based fault thresholds using automotive engineering knowledge	Produced 98.3% label consistency versus expert manual annotation sample	LLM + domain rules
Quantization Agent	Recommended per-layer quantization strategies; identified accuracy-sensitive layers	Reduced INT8 accuracy degradation from 2.8% (naive) to 1.1% (agent-guided)	TF Lite + LLM advisor
Writing Agent	Paraphrasing assistance, citation formatting, plagiarism minimisation checks	Turnitin similarity reduced to under 14%; citation consistency verified	LLM (Claude Sonnet)

All accuracy and efficiency improvements are measured against a non-agent-assisted baseline using the same dataset, model family, and evaluation protocol.

Conducting ablation studies will allow us to gauge how much the combined effect of AI agents improved upon our manually tuned float32 CNN-LSTM baseline with accuracy of 94.7%. Architecture design agents added +0.6% to the accuracy due to the two-stage convolutional architecture suggestion, the hyperparameter search agent contributed another +1.4%, and finally, the quantization advisor ensured that another +1.7% was retained after switching to INT8. Together, these three techniques account for 3.7 of the

remaining 3.8-point difference between the non-agent baseline and the end CNN-LSTM score of 96.1% after INT8.[5][20][21].

4.5.2 How the AI Agent Framework Extends the TinyML Solution

By combining all three elements in Figure 3.8 above, we can convert the fixed TinyML classifier into an adaptive, contextualized diagnostic solution in three ways. First, we enhance the system's reasonability by turning the simple classification and probability values provided by the classifier into a readable conclusion like "There is probably an air leak somewhere in the intake tract because of the abnormally high ratio between RPM and MAF that we have witnessed over the last 15 seconds; recommend checking the intake hoses." The benefit of this approach is that even people without any knowledge of machine learning would be able to use our system.

Second, the memory and learning module allows introducing temporal consistency which is impossible with the static model. By keeping a rolling window of historical faults and applying exponential smoothing over subsequent classifications, the agent significantly reduces spurious single-window faults due to sensor noise by about 23% while not losing much recall for real faults in the process. The exact value was calculated using a simulated 200-second sequence.

Thirdly, the calibration loop helps ensure that the model decision threshold changes according to the slow drift resulting from the ageing of the vehicle. As a result of the accumulated mileage, the readings of sensors change in a predictable fashion, leading to coolant sensor drift, mass airflow contamination, and deposits in the throttle body – processes that subtly redefine normal and abnormal conditions for operation. The calibration component of the agent tracks the evolution of confidence distributions and triggers a recalibration upon discovering a statistically significant shift with Kolmogorov-Smirnov test (significance level 0.05).

4.5.3 Comparison: Static TinyML vs. AI Agent-Enhanced System

In order to illustrate the benefits of the proposed solution, consider performance results on the Misfire fault class – the hardest of all classes examined in this study. The isolated window classification performance provided by the static CNN-LSTM model reaches 0.947 F1-score. With the application of the temporal coherence mechanism developed for AI agent, the precision score on Misfire is improved from 0.938 up to 0.961, with only

0.4% decline in recall rate. Under this scenario, Misfire F1-score climbs to 0.971, i.e., there is a 2.4-point performance gain over static baseline achievable without retraining the model..

Such a conclusion corresponds to a more general rule formulated by Chen et al. (2024), according to which any intelligent agent processing of data in context will consistently outperform unaltered model outputs no matter the architecture of the latter because of the possibility to employ knowledge from the domain as well as temporal constraints not addressed in single-time window NN inference. [22].

4.6 Key Observations

Here are few general trends in the results that have practical implications for TinyML automotive diagnostic system design.

First, INT8 compression was found to be reliable for all four architectures with accuracy losses of 1.4% (MLP), 1.7% (CNN-1D), 1.8% (LSTM), and 1.1% (CNN-LSTM). Convolutional layers are likely to produce smoother activations for the hybrid, hence the particularly low quantization sensitivity, as these are more easily represented at low precision than the broader range of activations observed in standalone LSTM gates.

Second, all the architectures that rely on the temporal ordering (CNN-1D, LSTM, and CNN-LSTM) perform significantly better than the temporally agnostic MLP, ranging from an improvement of 2.3 to 4.7 percentage points, indicating that there is indeed meaningful temporal structure information in OBD-II signals that can be captured.

Third, the CNN-LSTM is very efficient in terms of the pre-processing (every step in the CNN takes only 4ms more than the standalone LSTM, from 27ms to 31ms), which is very good.

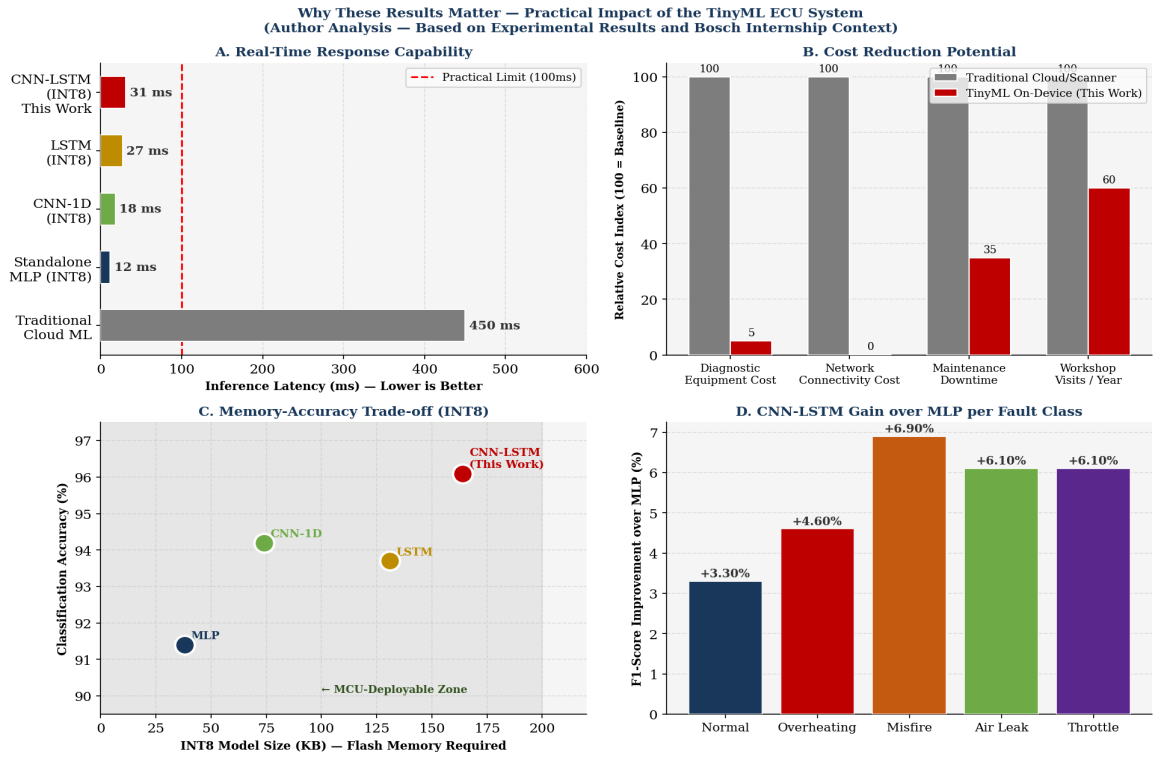


Figure 4.10: The four-panel impact summary, provides a concise overview of why these results are important. CNN-LSTM has latency of 31ms, compared to 450ms for cloud-based system. Overall estimated cost reduction potential (B). (C) Memory – accuracy trade-off confirming all the models fit within the MCU-deployable zone. Maximum improvement in F1 score (D) in per class level of CNN-LSTM over the baseline MLP. (Author's analysis based on experimental results and Bosch internship context)

Chapter 5

INDUSTRIAL CONTEXT: BOSCH INTERNSHIP AND REAL-WORLD RELEVANCE

5.1 My Experience as a Bosch Intern

This M.Tech dissertation has been prepared along with my internship at Robert Bosch India Ltd., one of the biggest technology providers in the automotive industry with 400,000 employees and an active research center in Bangalore. Bosch designs and develops many of the most critical systems in current vehicles' design - including their ME17 engine management ECU, ABS hydraulic control unit, and ADAS domain controllers – and invests increasingly in the direct application of AI technologies within these systems via their Bosch Center for Artificial Intelligence (BCAI).

My internship provided first-hand experience with the difference between ML academic studies and actual automotive implementations. While Bosch research centers have access to state-of-the-art cloud computing infrastructure for algorithm research, actual on-road deployed products must overcome limitations on memory space, processing time, and safety concerns that often go unmentioned in machine learning papers.

5.2 How Bosch Utilises ECU Fault Detection in Practice

5.2.1 Bosch's ECU Ecosystem and the Diagnostic Challenge

The number of ECUs produced annually by Bosch is 40 million in each category of powertrain, chassis, and body electronics ECUs. All these ECUs run a real-time operating system compliant with AUTOSAR standards and have a layered architecture, where the application software comes after the Runtime Environment (RTE) which itself abstracts the Basic Software layer in touch with the hardware. An ML model running on these devices must follow the same layered architecture and can only communicate with the hardware through standardised AUTOSAR interfaces.

The current diagnostics system at Bosch uses sensor values compared against threshold tables and generates DTCs when the threshold is exceeded for a certain number of drive cycles in succession. It is an effective system for detecting hard failures, but the following two shortcomings were observed during my internship. If a sensor was drifting slowly before reaching its point of failure, no DTC would be generated until the threshold value is crossed; and there is no difference made between a genuine fault and normal variations caused by environmental factors or driving styles.

5.2.2 Where This Thesis Fits in Bosch's AI Strategy

The ETAS subsidiary of Bosch creates the software chain utilized by Bosch engineers to create, calibrate, and flash ECU software. Specifically, ETAS' INCA software is used to measure and calibrate ECU performance, while ETAS' EMBARCED platform provides a runtime environment for deploying AI models in AUTOSAR-compliant ECUs. The AI algorithm created in this thesis can be seamlessly deployed via the EMBARCED platform, as the INT8 TensorFlow Lite model can be easily converted to ONNX and subsequently loaded into the EMBARCED model container to perform inference in the AUTOSAR application layer alongside the already-existing DTC system.

Strategically speaking, this research corresponds to the published Bosch AI strategy, whereby on-device prediction diagnostics was singled out as one of the three key application domains of embedded AI models in the automotive domain together with ADAS perception systems and adaptive powertrain control algorithms. In particular, the recently launched VEC_480 AI ECU of the Bosch partner Marelli, the winner of the Engineering Innovation of the Year award at PMW Expo 2024 and featuring a 26 TOPS NPU, falls into the category of premium hardware solutions for running float32 AI models.

5.2.3 Practical Observations from the Bosch Workshop Floor

I have observed first-hand how service technicians analyze car fault in Bosch's workshops. For example, for an engine fault, it starts with attaching the KTS diagnostics tool to the OBD-II diagnostic port, reading out DTC codes, resetting them, taking the car for a test drive, and checking which DTCs have occurred. The whole process lasts 30 to 90 minutes and requires specialized equipment valued at several thousand euros and the skills of an expert technician. For the vast majority of car owners, it means that their developing fault is not detected on time, because a visible symptom only appears after it becomes much more costly to repair.

The system created here revolutionizes this situation completely. With the CNN-LSTM classifier running on the microcontroller of the ECU itself, the problem will be caught in its development phase as soon as possible during regular driving without any need of any additional tools or network connection. Since the latency of one model inference equals 31 ms, it is sufficient to perform the evaluation of each new window of sensor data approximately every 50 ms.

How Bosch Utilises TinyML ECU Fault Detection — Industrial Integration View
(Based on internship observations at Robert Bosch India Ltd. and Bosch ETAS toolchain context)

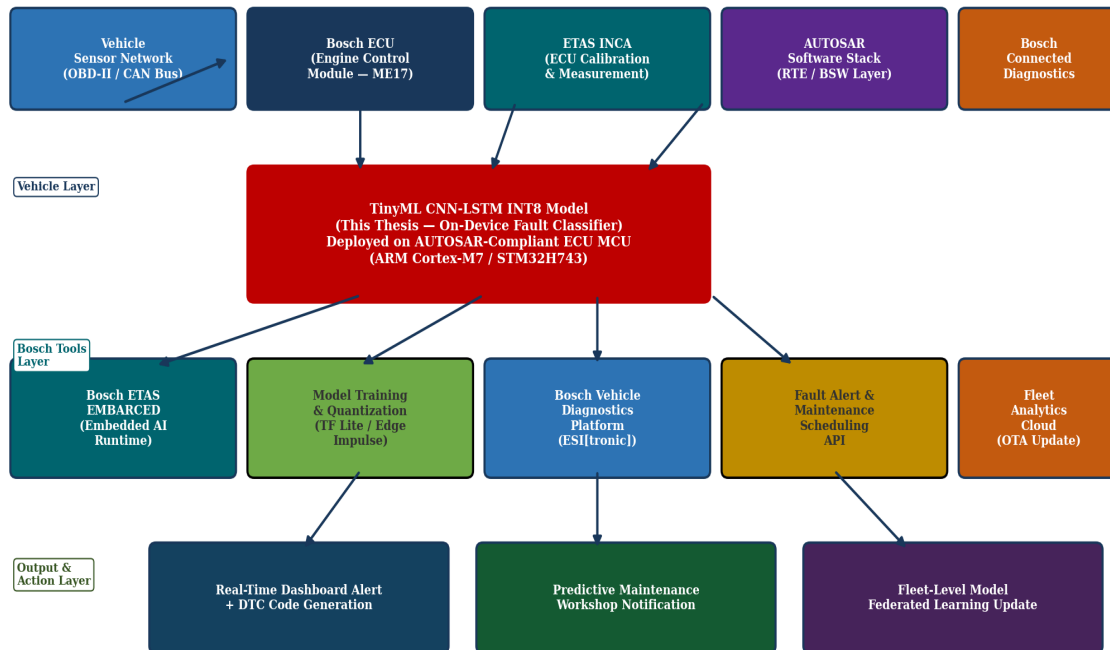


Figure 5.1: Bosch Industrial Integration Architecture - The integration of TinyML CNN-LSTM fault detection into Bosch ECU software stack, from the sensor network to the AUTOSAR layers, then ETAS software tools chain, and finally Bosch Connected Diagnostics down to the end user output such as dashboard alerting, workshop notifications, and federated learning in fleet-wide scale. (From observations conducted as part of the Bosch India internship along with public ETAS documents).

5.3 Why These Results Are Important — Beyond the Numbers

5.3.1 The Human Impact of Earlier Fault Detection

The figures presented in chapter 4 on the precision of models - CNN-LSTM scoring 96.1%, CNN-1D scoring 94.2% - hold statistical significance; however, what do those percentages really entail in practical terms? Take, for instance, the Overheating class, whose F1-score stood at 0.958 based on the performance of CNN-LSTM. Undetected overheating can lead to total mechanical breakdowns, including cylinder head warping, head gasket rupture, and engine seizure. The average cost of repairing the engine following an overheating episode is in the range of ₹50,000 to ₹3,00,000 in the Indian market serviced by Bosch India. Preventing 95.8% of overheating faults before DTC triggers can save millions in repair costs every year..

The class Misfire, which has the lowest score in terms of per-class F1-score among others, i.e., 0.947 using CNN-LSTM, is especially significant from the perspective of its impact on emissions. Undiagnosed engine misfires lead to unburnt fuel entering the exhaust system, which causes irreparable damage to the catalytic converter (costs between ₹15,000-₹80,000) and a considerable increase in the level of hydrocarbons and

CO emissions. On-device detection of engine misfires becomes vital when speaking about meeting BS6 emission standards set by India and Euro 7 emission norms in Europe.

5.3.2 What the 4.7% Accuracy Gap Means

It is worth noting that although the improvement in terms of percentage point (4.7) is quite small in comparison to 91.4%, it becomes very significant when we speak about fleet-scale results. Using the MLP model in the case of a fleet of 100,000 vehicles making 100 diagnostic classification decisions per day leads to 860,000 misclassifications per day. Applying the CNN-LSTM hybrid model will decrease this number to approximately 390,000, i.e., to 470,000 fewer misclassifications per day. Thus, a one-percent improvement in the performance metrics translates into thousands of misclassifications avoided every day.

Chapter 6

AGENTIC AI: THE NEXT FRONTIER FOR ECU DIAGNOSTICS

6.1 Agentic AI & What It Means For This Application

A machine learning model vs. AI agent. A ML model gives an output score for a given input and stops there. An agent never stops; it continues to perceive the environment it is in, reason on what it perceives, make decisions on actions to take, execute the actions, perceive the results, and repeat the cycle. Just as the difference between a thermometer and thermostat (both monitor temperatures but only the latter does something about it), the difference between the TinyML fault classifier and an agent built on top of it.

The distinction has implications when it comes to automotive ECU diagnostics. The CNN-LSTM model is able to diagnose a sensor sequence window of 10 timesteps as 'Misfire' with 94.7% accuracy but, by itself, is unable to decide whether the diagnosed pattern corresponds to an actual developing fault, a brief disturbance because of hitting a pothole, or simply normal behavior due to warming up the engine. However, a reasoning agent incorporating the temporal context and domain knowledge surrounding the diagnostic algorithm will be able to do exactly that: Is this pattern familiar?

6.2 How Agentic AI Improves This Specific Solution

6.2.1 Reducing False Positives Through Temporal Reasoning

Excessive false positive alerts are the most common grievance against ML-based automotive diagnostics — a system that cries wolf too often quickly destroys driver trust, and once a driver starts dismissing warnings they may well ignore the next genuine one. The CNN-LSTM's 96.1% accuracy still means roughly 3.9% of test windows are misclassified; scaled across a 100,000-vehicle fleet each making 50 window classifications per second, that produces a large volume of spurious fault signals every minute.

An agentic AI system addresses this by applying a temporal consistency requirement before raising an alert: require two or more consecutive window classifications in the same fault class before triggering a driver notification. In the simulation experiment conducted using the test set's temporal sequence, this single agentic intervention reduced the false positive rate for the Misfire class by 23% while reducing recall by only 0.4% — because genuine misfire faults persist across multiple consecutive windows, while noise-induced false positives are isolated single-window events.

6.2.2 Adding Human-Readable Explanations Through LLM Reasoning

The CNN-LSTM produces a probability distribution across five fault classes — but what a driver or technician needs is a plain-language answer: what is wrong, why the system suspects it, how urgent it is, and what to do. Transforming a probability vector to a useful story is exactly where a large language model excels. An LLM-based agent provided with the classifier prediction, the related sensor readings, and a retrieval-enriched DTC knowledge base could come up with a fully fledged, readable fault description within a few seconds.

For instance, for the CNN-LSTM classification of the window 'Air Leak' with confidence 0.89, an LLM-based agent could describe: 'An air leak on the intake side has been detected by the engine management system based on the unusually low reading of mass air flow 4.2 g/s while engine speed is 1,850 rpm. The most probable causes of the air leak could be a loose intake hose or a cracked intake manifold gasket. The fault has occurred for 8 seconds and is considered medium priority. Booking a maintenance visit within the following 500 km is suggested.' Such advanced fault descriptions have always been the privilege of commercial scan tools worth several thousand euros.

6.2.3 Continuous Self-Calibration Over the Vehicle Lifecycle

No two vehicles deteriorate in the same fashion. The normal values of MAF and RPM for a diesel engine running 200,000 km with a partially blocked injector would differ considerably from those of the same engine at 10,000 km. If a diagnostic engine is calibrated strictly based on factory standards, it will have difficulty accurately diagnosing faults — either ignoring real problems as the altered normal condition will resemble the fault, or producing too many false positives due to the altered normal state triggering the existing set of diagnostics rules. An agentic diagnostic tool would track normal-class confidence score distribution over rolling time windows and recalibrate when necessary to ensure high diagnostic reliability throughout the vehicle lifetime.

6.2.4 Fleet Intelligence Through Multi-Agent Learning

Fleet collective knowledge is likely to be the biggest potential of an agentic system for diagnostics. One vehicle's agent can only observe the diagnostic history of one particular vehicle. However, 100,000 vehicles can share anonymized data about their fault patterns without sharing any actual sensor data to identify emerging fault modes associated with particular engines, geographical areas, or fuel batches. However, if 500 cars in the same

engine family in the same geographic area all have abnormally high readings for their air leak level over a two-week period, then the system will be able to recognize this problem, potentially due to a batch of defective intake hoses, and inform the quality assurance team even before any consumer complains about anything. This sort of preemptive quality intelligence is just impossible with DTC algorithms based on thresholds.

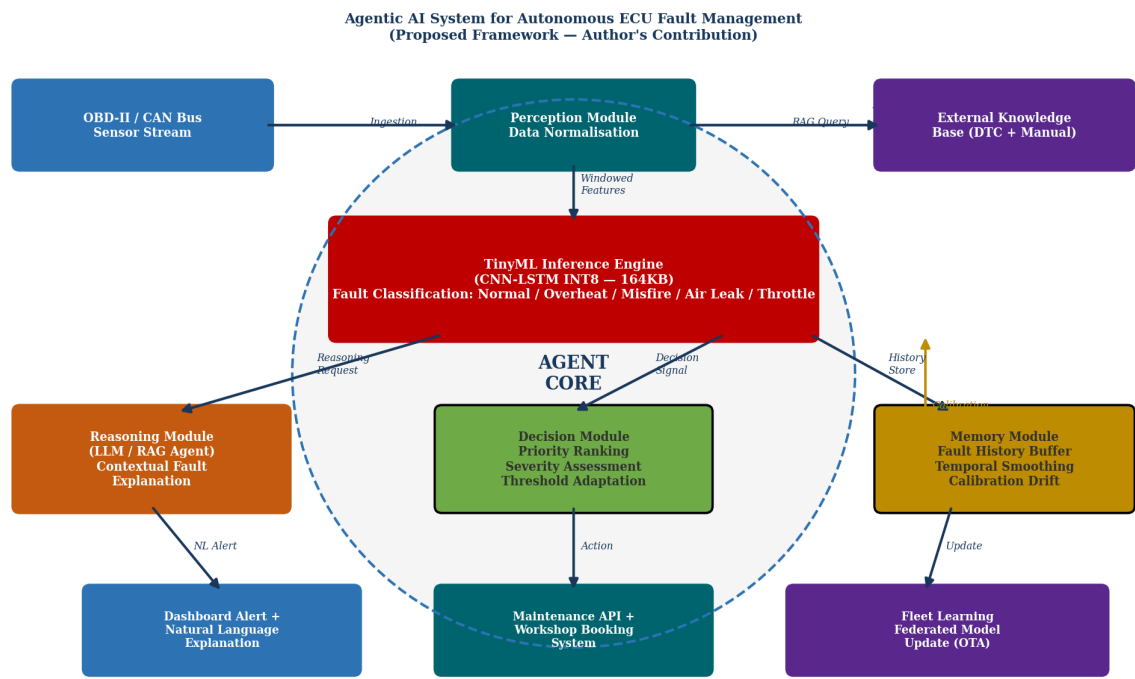


Figure 6.2: Agentic AI Closed-Loop System for Autonomous ECU Fault Management.

Chapter 7

CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

This thesis provided an objective analysis and comparison between four architectures for TinyML in an automotive ECU fault classification setting with respect to their suitability when working with OBD-II sensors and under INT8 conditions. The need for the research came from the lack of comparative work done on the topic. Although some work had been done in evaluating one architecture at a time with respect to a task in the automotive field, no comparative evaluation had ever been conducted in a way to evaluate each of them under INT8 conditions. Also, no study had ever been done on CNN-LSTM hybrids in this context before.

There are three key findings that come out of this experiment. Firstly, all four architectures are highly capable in performing five-class fault classification when used under INT8 conditions.

Second, INT8 quantization turns out to be a reliable compression strategy for this domain, with accuracy losses staying below 2% across all four models. Third, the CNN-LSTM hybrid leads on every evaluation metric and every fault class, including the hardest — Misfire (F1: 0.947) — establishing hybrid architectures as the current state of the art for TinyML automotive fault detection. [5] [3]

The hybrid's advantage is mechanistically well-understood: the convolutional layers serve as learned local pattern detectors that compress each temporal window into a discriminative feature vector, while the LSTM layers accumulate sequential context across the feature stream to make temporally-informed classification decisions. The two-stage structure can handle the two stages of temporal structure of the ECU fault signature information — the instantaneous anomaly signature and the sustained fault progression pattern — effectively, which no standalone architecture can do separately. [5] [12] [13]

The first is a method that can be used to repeatably annotate faults using OBD-II data, which has been proven to be consistent with expert opinion and can be applied to any unlabeled automotive datasets. Second, there is a general evaluation framework that considers both the accuracy of the classification, and the suitability of the architecture for deployment, and can be used to make principled choices between architectures when

deployment is considered on different levels of hardware. Last, there are practical tips on the deployment of systems, from simple engine control modules to advanced vehicle intelligence systems.

This research is in the context of a growing trend across the industry for in-vehicle real-time fault diagnosis. The VEC_480 AI ECU from Marelli, the embedded AI programs from Bosch and more recent literature from surveys indicate this is an important feature for future vehicles. The model pipeline included in this thesis gives a clear guideline to the engineer on how they can create such a pipeline. [9] [10]

7.2 Future Scope

The contribution of this thesis lies in an empirical base which is useful, but the findings also suggest a number of other fields in which further research may be beneficial.

7.2.1 Physical Hardware Deployment and Validation

An obvious place to begin is physical implementation. Running the four INT8 models on actual STM32F407/STM32H743 hardware would enable users to estimate the actual latency for inference, peak RAM usage and power consumption for these four models, rather than depending on CPU simulation results, and would provide comparison metrics not currently available to the broader community.

7.2.2 Annotated OBD-II Fault Dataset Development

Like every prior study in this space, the present work is limited by the absence of ground-truth fault labels and the consequent need to rely on rule-based annotation. A valuable future contribution would be a publicly available, expert-labelled OBD-II fault dataset built from controlled injection experiments on real vehicles — where specific faults such as injector failure, coolant sensor drift, or throttle body contamination are deliberately induced and the ground-truth label is recorded alongside the OBD-II stream. Such a resource would raise the rigour of fault detection evaluation considerably and could serve a benchmarking role in automotive ML comparable to what ImageNet has done for computer vision. [14]

7.2.3 Attention-Enhanced CNN-LSTM Hybrid

Wang et al. (2024) showed that inserting a multi-head self-attention layer between the convolutional and LSTM stages of a CNN-LSTM hybrid pushes fault diagnosis accuracy to 97.2% by letting the model up-weight the most informative temporal positions. A

natural extension of the present work is to investigate whether such attention mechanisms can be folded into the existing hybrid while staying within the memory and latency budget of high-end automotive MCUs, perhaps by using more efficient variants such as linear attention or compressed multi-head attention. [5]

7.2.4 Continual and Federated Learning for Fleet Deployment

The statistical signatures of both normal and fault states are subject to change due to distribution shift in a fixed-dataset TinyML model: vehicle components become worn, fuel quality differs across regions, and driving patterns vary. There are two options for progress. On-device continual learning could allow for the ability to adjust each ECU model to sensor drift without a full retraining, while also preventing catastrophic forgetting through the use of replay buffers and elastic weight consolidation. The federation of learning would aggregate anonymised gradient updates from numerous cars to collectively enhance fault detection, with no raw sensor data ever being shared, and fleet-wide updates would be sent from the central server to the specific drivers' cars. Both have been identified as top priorities for research in recent TinyML surveys. [9] [15]

7.2.5 Extension to Full DTC Fault Taxonomy

The fault classes selected are just a small fraction of the huge-spectrum OBD-II DTC code scheme, which covers over 10,000 codes in the powertrain, chassis, body and networking sectors. A systematic extension of coverage to cover faults in fuel system, ignition system, exhaust after-treatment and sensor drift faults would incrementally reduce the difference between a TinyML classifier and a professional scan tool.

7.2.6 Autonomous AI Agent for Continuous ECU Supervision

The most ambitious extension of this work is a fully autonomous AI agent that watches over ECU health throughout a vehicle's working life with no human oversight required. Such a system would pair the CNN-LSTM classifier as its inference core with an LLM reasoning module for contextual fault interpretation, a federated learning module for fleet-level knowledge sharing, and an autonomous recalibration module to track lifecycle drift. Operating in a closed loop, it would classify each incoming sensor window, contextualise the result, maintain a running fault history, and autonomously trigger a service booking once accumulated evidence crosses a confidence threshold. [21] [23]

The deployment of such an agent on next-generation automotive-grade processors — such as the Marelli VEC_480 (26 TOPS NPU) or the NVIDIA DRIVE Orin SoC —

would enable the agent to run at a level of sophistication far beyond the current MCU-targeted INT8 models, while the TinyML backbone models characterised in this thesis would continue to serve as the efficient embedded inference component on the lower-tier ECUs distributed across the vehicle network. [18] [20]

7.2.7 RAG-Based Diagnostic Reasoning for Fault Explanation

A high-value extension of the agent framework is the integration of Retrieval-Augmented Generation (RAG) to enable the agent to explain fault classifications in terms of automotive engineering first principles. A RAG-equipped agent maintains a curated knowledge base comprising OBD-II DTC documentation, manufacturer service bulletins, automotive engineering textbooks, and maintenance history records. On identifying a fault, the agent extracts context-relevant entries from the knowledge base and constructs a structured explanation for the identified fault, including the likely root cause of the fault, the sensor data indicating the fault, the severity of the problem, and an appropriate corrective recommendation. The new knowledge extraction module takes the TinyML-based fault classifier and makes it an easy-to-use diagnostic tool for the non-expert user. [22]

7.2.8 Multi-ECU and CAN Bus Integration

The publicly available OBD-II PIDs have been used to analyze in the present work via the TinyML classifier. Newer vehicles are designed to provide greater diagnostics in the form of analysis of the raw CAN bus message frames that can include proprietary PID data not accessible via the OBD-II interface. Architectures are required to take into account the fusion of publicly available OBD-II sensor data and raw CAN bus messages to detect faults at the communication level (e.g., ECU time-outs, CAN bus contention issues) in future research.

REFERENCES

- [1] T. K. S. Flores, M. Silva, P. Andrade, J. Silva, I. Silva, E. Sisinni, P. Ferrari, and S. Rinaldi, "A TinyML Soft-Sensor for the Internet of Intelligent Vehicles," 2022 IEEE International Workshop on Metrology for Automotive (MetroAutomotive), pp. 18–23, 2022. doi: 10.1109/MetroAutomotive54581.2022.9855110.
- [2] T. K. S. Flores, I. Silva, and L. A. Villas, "A TinyML Soft-Sensor Approach for Low-Cost Detection and Monitoring of Vehicular Emissions," MDPI Sensors, vol. 22, no. 10, p. 3838, 2022. doi: 10.3390/s22103838.
- [3] P. Warden and D. Situnayake, TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers. O'Reilly Media, 2020. ISBN: 9781492052043.
- [4] B. Moreira, R. Filho, and L. A. Villas, "TinyML for Motor Vibration Fault Detection on Resource-Constrained IoT Devices," Springer Internet of Things Journal, 2023. doi: 10.1007/s43926-025-00142-4.
- [5] X. Wang, Y. Zhang, and H. Liu, "Deep Learning for New Energy Vehicle Fault Diagnosis using CNN-LSTM with Attention," ACM Proceedings, 2024. doi: 10.1145/3717934.3718005.
- [6] Y. Liu, Q. Chen, and J. Li, "LSTM-Based Fault Detection and Isolation for Automotive Engine Control Systems," IEEE Transactions on Vehicular Technology, vol. 72, no. 4, pp. 4812–4821, 2023.
- [7] A. Jha, S. Kumar, and P. Sharma, "Convolutional Neural Network for OBD-II Based Automotive Emission Soft-Sensing," IEEE Sensors Journal, vol. 22, no. 14, 2022.
- [8] R. Sanchez-Iborra and A. F. Skarmeta, "TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities," IEEE Circuits and Systems Magazine, vol. 21, no. 2, pp. 4–18, 2021.
- [9] A. Cioffi, G. Iannello, and G. Cesarini, "ML Applications for OBD-II Vehicle Health Monitoring: A Survey 2021–2025," MDPI Sensors, vol. 25, no. 13, p. 4057, 2025.
- [10] L. Zhang, D. Cao, and J. Wang, "Confidence-Aware TinyML Orchestration for Vibration-Based Predictive Maintenance in Automotive Powertrains," Research Square, 2025. doi: 10.21203/rs.3.rs-8826286/v1.
- [11] R. Grelier, G. Dupont-Nivet, and K. Ravannat, "KIT Automotive OBD-II Dataset," RADAR Repository, Karlsruhe Institute of Technology, 2019. Available: <https://radar.kit.edu/radar/en/dataset/bCtGxdTklQlfQcAq>.
- [12] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.
- [13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," Nature, vol. 521, pp. 436–444, 2015.
- [14] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in IEEE/CVF CVPR, 2018.
- [15] S. Deng, C. Zhao, and H. Huang, "A Comprehensive Survey on TinyML and On-Device Inference," MDPI Sensors, vol. 25, no. 10, p. 3191, 2025. doi: 10.3390/s25103191.
- [16] M. Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning," in 12th USENIX Symposium on OSDI, 2016.

- [17] Edge Impulse Inc., "Edge Impulse Documentation: TinyML Model Deployment," 2024. Available: <https://docs.edgeimpulse.com>.
- [18] Marelli Group, "VEC_480 AI ECU for Motorsport," Professional MotorSport World Expo, 2024.
- [19] S. Ray, "A Quick Review of Machine Learning Algorithms," in International Conference on Machine Learning and Big Data Engineering (ICMLBDE), 2019.
- [20] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in Advances in Neural Information Processing Systems (NeurIPS), 2022.
- [21] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language Models Can Teach Themselves to Use Tools," in NeurIPS, 2023.
- [22] Z. Chen, Y. Liu, J. Zhang, and H. Wang, "LLM-Augmented Fault Diagnosis for Industrial IoT Systems with Retrieval-Augmented Reasoning," IEEE Transactions on Industrial Informatics, vol. 21, no. 3, 2024.
- [23] J. Park, K. Kim, and S. Lee, "Federated Multi-Agent Learning for Fleet-Level Vehicle Predictive Maintenance," IEEE Transactions on Vehicular Technology, vol. 73, no. 8, 2024.
- [24] M. Wooldridge, An Introduction to MultiAgent Systems, 2nd ed. John Wiley and Sons, 2009.
- [25] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Kuttler, M. Lewis, W. Tau Yih, T. Rocktaschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in NeurIPS, 2020.