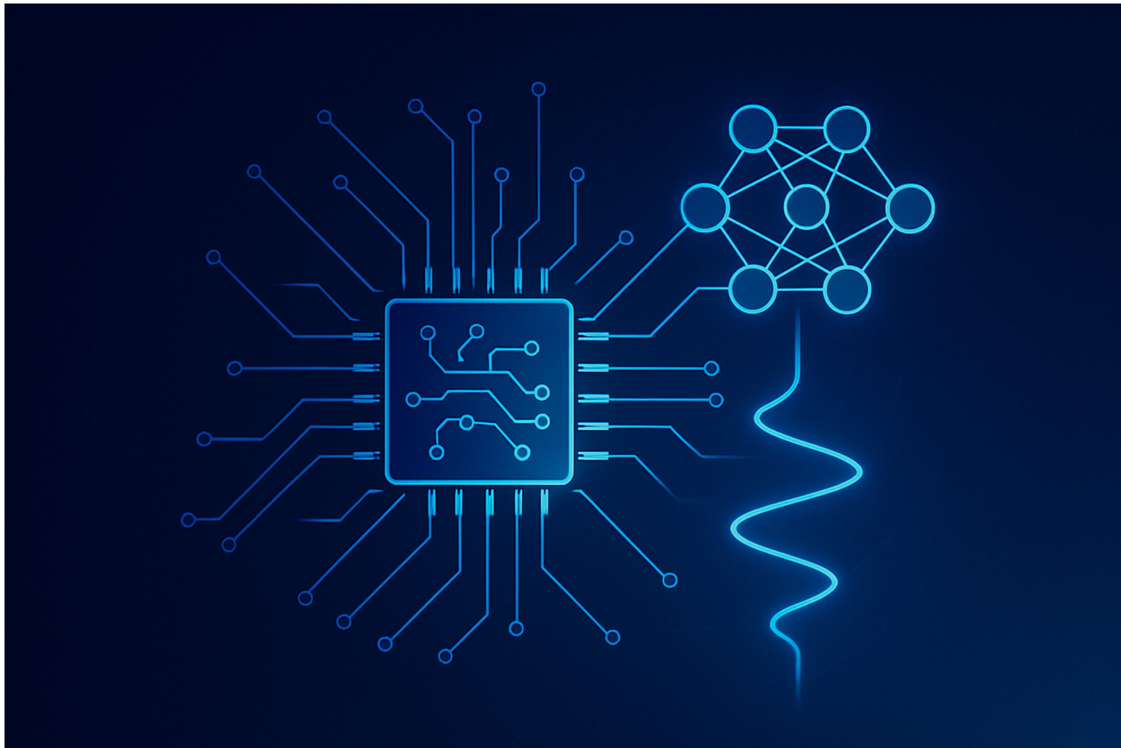


Anomaly Detection on Power Input Using Machine Learning on Microcontroller Systems

Master's thesis in Systems, Control and Mechatronics

Qirui Yang & Haoming Yi

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Anomaly Detection on Power Input Using Machine Learning on Microcontroller Systems

QIRUI YANG & HAOMING YI

DEPARTMENT of ELECTRICAL ENGINEERING
Chalmers University of Technology
Gothenburg, Sweden 2025

Anomaly Detection on Power Input Using Machine Learning on Microcontroller Systems
QIRUI YANG & HAOMING YI

© QIRUI YANG & HAOMING YI, 2025.

Supervisor: Björn Olsson, Ericsson AB
Advisor: Alireza Bordbar, Department of Electrical Engineering, Chalmers University of Technology
Examiner: Erik Agrell, Department of Electrical Engineering, Chalmers University of Technology

Master's Thesis 2025
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Ericsson AB
Lindholmspiren 11
SE-417 56 Gothenburg
Telephone +46 10 719 0000

This report was written with the assistance of ChatGPT.
Cover: Illustration of a microcontroller with neural network integration and voltage waveform acquisition, highlighting real-time anomaly detection in power inputs.

Typeset in LaTeX
Gothenburg, Sweden 2025

Anomaly Detection on Power Input Using Machine Learning on Microcontroller Systems
QIRUI YANG & HAOMING YI
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Robust −48 V power input stages are essential for modern telecommunication equipment, including radio and baseband units. In practice, the input line experiences power-line disturbances (PLDs) such as square-wave ripple and voltage dips, which complicate real-time fault detection. Threshold-based protection is effective for well-known cases but lacks adaptability to varying transients.

This thesis develops and evaluates lightweight machine-learning methods for on-device PLD classification on a resource-constrained microcontroller (MCU). A four-class dataset (normal, ripple, milddip, severedip) is constructed by sampling at 1 kHz and segmenting signals into 20 ms windows, yielding 40,000 labeled examples. Three compact models are compared: a 1D convolutional neural network (1D-CNN), a long short-term memory (LSTM) network, and a hybrid CNN+LSTM. Models are trained and validated offline and then quantized with TensorFlow Lite (TFLite) for embedded deployment on an STM32G474.

On the held-out test set, the 1D-CNN and the hybrid achieve accuracy and macro-F1 around $> 99\%$, whereas the standalone LSTM is lower under the same 20 ms context. After full-integer (INT8) quantization, the 1D-CNN preserves accuracy while reducing the model size to about 21 kB and achieving ≈ 0.03 ms inference per 20 ms window, meeting real-time MCU constraints. In contrast, the recurrent models require `SELECT_TF_OPS` support in TFLite, which makes bare metal deployment less practical.

These results demonstrate that a quantized 1D-CNN provides an effective and deployable solution for on-device monitoring of −48 V power inputs, enabling reliable, low-latency anomaly detection in embedded telecommunication systems.

Keywords: PLD, anomaly detection, 1D-CNN, CNN–LSTM, quantization (INT8), TensorFlow Lite, embedded microcontroller (STM32G474)

Acknowledgements

We would like to express our sincere gratitude to our supervisor, Björn Olsson at Ericsson AB, for his continuous guidance, encouragement, and valuable industrial insights throughout this thesis. We are also deeply thankful to our academic advisor, Dr. Alireza Bordbar, from the Department of Electrical Engineering at Chalmers University of Technology, for his constructive feedback and support, as well as to our examiner, Prof. Erik Agrell, for his careful evaluation and valuable suggestions.

We are grateful to our manager, Johan Håkansson, for providing the opportunity to carry out this thesis project in close collaboration with Ericsson AB, and for his encouragement during the process.

Finally, we wish to thank both Ericsson AB and Chalmers University of Technology for providing the resources, environment, and support that made this work possible.

Qirui Yang & Haoming Yi, Gothenburg, September 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

1D-CNN	One-Dimensional Convolutional Neural Network
ACC	Accuracy
Adagrad	Adaptive Gradient
ANN	Artificial Neural Networ
API	Application Programming Interface
ARIMA	AutoRegressive Integrated Moving Average
DC	Direct Current
FP32	32-bit Floating Point
FLOPs	Floating Point Operations
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
INT8	8-bit Full Integer Quantization
LSTM	Long Short-Term Memory (Recurrent Neural Network)
Macro-F1	Macro-averaged F1-score
MCU	Microcontroller Unit
ML	Machine Learning
MLP	Multilayer Perceptron
PLD	Power-Line Disturbance
PR	Precision–Recall
QAT	Quantization-Aware Training
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
ROC	Receiver Operating Characteristic
RMSProp	Root Mean Square Propagation
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SRAM	Static Random Access Memory
STM	STMicroelectronics (MCU vendor family)
SVM	Support Vector Machine
TFLite	TensorFlow Lite
TFLM	TensorFlow Lite Micro
TPR	True Positive Rate

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

t	Discrete time index within a window, $t = 1, \dots, N$
k	Window (segment) index in a signal stream
c	Class index, $c \in \mathcal{C}$
l	Neural network layer index
i, j	Neuron indices in layer equations (feedforward networks)

Sets

$\mathcal{C}$	Set of classes: $\{\text{normal}, \text{ripple}, \text{milddip}, \text{severedip}\}$
$\mathcal{X}$	Set of input windows (each $\mathbf{x} \in \mathbb{R}^{N\times 1}$)
$\mathcal{Y}$	Set of labels aligned with $\mathcal{X}$
$\mathcal{W}$	Set of trainable parameters (weights and biases) of the network
$\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}, \mathcal{D}_{\text{test}}$	Training / validation / test subsets

Parameters

V_{nom}	Nominal DC input voltage ($-48\,\text{V}$)
f_{s}	Sampling frequency ($1\,\text{kHz}$)
N	Samples per window (20 samples $\Rightarrow$ $20\,\text{ms}$)
μ_{train}	Training-set mean for global standardization
σ_{train}	Training-set standard for global standardization
ϵ	Small constant to avoid division by zero in normalization
$\theta_{10\%},\ \theta_{20\%}$	Dip thresholds ($-43.2\,\text{V},\ -38.3\,\text{V}$)

$\tau_{\min}$	Minimum dwell time to confirm a dip within a window
α	Base learning rate for Adam optimizer
M	Max training epochs
B	Batch size
p_{drop}	Dropout probability
$s_{\text{in}}, z_{\text{in}}$	Input quantization scale and zero-point (INT8)
$s_{\text{out}}, z_{\text{out}}$	Output quantization scale and zero-point (INT8)
$w^{(l)}, b^{(l)}$	Weights and biases of layer l

Variables

x_t	Raw voltage sample at time t
x	One input window (20×1) before normalization
$\tilde{x}$	Standardized window: $\tilde{x}_t = (x_t - \mu_{\text{train}})/(\sigma_{\text{train}} + \epsilon)$
x_{INT8}	Quantized input: $x_{\text{INT8}} = \text{round}(\tilde{x}/s_{\text{in}} + z_{\text{in}})$
y	Ground-truth class label, $y \in \mathcal{C}$
$\hat{p}$	Predicted class-probability vector (softmax), $\sum_c \hat{p}_c = 1$
$\hat{y}$	Predicted class, $\hat{y} = \arg\max_c \hat{p}_c$
L	Loss value (cross-entropy unless otherwise stated)
h_t, c_t	LSTM hidden state and cell state at time t
f_t, i_t, o_t	Forget, input, and output gates in LSTM

Contents

List of Acronyms ix

Nomenclature xi

List of Figures xvii

List of Tables xix

1 Introduction 1
- 1.1 Background 1
- 1.2 Motivation 2
- 1.3 Research Questions 3
- 1.4 Related Work 3
  - 1.4.1 Conventional Protection and Detection Methods 3
  - 1.4.2 Machine Learning for Power Disturbance Detection 4
  - 1.4.3 Embedded Deployment and Quantization 4
  - 1.4.4 Positioning of This Work 4
- 1.5 Thesis Outline 4
- 1.6 Statement on the Use of AI Tools 6

2 Theory 7
- 2.1 Voltage Dip Classification Criteria 7
  - 2.1.1 Percentage Dip Calculation 7
  - 2.1.2 Mild Dip (10–20%) 7
  - 2.1.3 Severe Dip (≥20%) 8
  - 2.1.4 Boundary Handling and Noise Tolerance 8
- 2.2 Machine Learning 8
  - 2.2.1 Supervised learning 8
  - 2.2.2 Unsupervised learning 9
  - 2.2.3 Reinforcement learning 9
- 2.3 Artificial Neural Networks 9
  - 2.3.1 Feed-forward and Optimization 10
  - 2.3.2 Activation Functions 11
  - 2.3.3 Loss Function 13
  - 2.3.4 Regularization 14
- 2.4 Convolutional neural networks (CNNs) 15
- 2.5 1D Convolutional Neural Networks (1D-CNN) 16

2.6 Sequential machine learning . . . 16
2.6.1 Recurrent Neural Networks (RNNs) . . . 17
2.6.2 Long Short-Term Memory (LSTM) . . . 18
2.7 Evaluation . . . 19
2.7.1 F_1–Score . . . 19
2.7.2 Specificity . . . 20
2.7.3 K-Fold Cross-Validation . . . 21
2.7.4 Performance Matrix . . . 21

3 Methods **23**
3.1 System Architecture and Workflow . . . 23
3.2 Data Collection and Generation . . . 25
3.2.1 Power Line Disturbance (PLD) Overview . . . 25
3.2.2 Dataset Construction . . . 25
3.2.3 Voltage Disturbance Design Criteria . . . 27
3.3 Data Preprocessing . . . 30
3.3.1 Windowing and Labeling Strategy . . . 30
3.3.2 Global Normalization . . . 31
3.3.3 Label Encoding . . . 31
3.3.4 Data Type Conversion . . . 32
3.3.5 Train/Validation/Test Split . . . 32
3.4 Model Design and Training . . . 34
3.4.1 Training Protocol . . . 34
3.4.2 1D CNN Architecture . . . 35
3.4.3 LSTM Architecture . . . 36
3.4.4 Hybrid CNN–LSTM Architecture . . . 36
3.4.5 Implementation Notes and Reproducibility . . . 37
3.4.6 Computational Efficiency . . . 37
3.5 Microcontroller Implementation . . . 38
3.5.1 ADC Sampling and Pin Configuration . . . 38
3.5.2 Memory and Real-Time Constraints . . . 39
3.5.3 Model Deployment Workflow . . . 39
3.5.4 Model Quantization for STM32G474 Deployment . . . 40
3.6 Evaluation . . . 42
3.6.1 Procedure . . . 42
3.6.2 Deployment Validation . . . 43

4 Evaluation and Results **45**
4.1 Pre-Quantization Performance . . . 45
4.1.1 Summary Table . . . 45
4.1.2 1D-CNN . . . 46
4.1.3 LSTM . . . 48
4.1.4 Hybrid (CNN+LSTM) . . . 49
4.1.5 Discussion . . . 51
4.2 Post-Quantization Performance . . . 52
4.2.1 Summary Table . . . 53
4.2.2 1D-CNN (Quantized) . . . 54

4.2.3 LSTM (Quantized) 54
4.2.4 Hybrid CNN+LSTM (Quantized) 55
4.2.5 Discussion 55
4.3 Model Complexity and Deployability 56
4.3.1 Comparison of Model Complexity Metrics 56
4.3.2 Accuracy Before Quantization 57
4.3.3 Discussion of Deployability Trade-offs 57
4.4 Comparative Discussion 58
4.4.1 Pre- vs. Post-Quantization Trends 58
4.4.2 Deployment-Oriented Comparison 58
4.4.3 Suitability for Application Scenarios 59
4.4.4 Summary 59

5 Conclusion **61**
5.1 Summary of Contributions 61
5.2 Key Findings 62
5.3 Limitations 62
5.4 Future Work 62
5.5 Summary 63

Bibliography **65**

A Appendix A **I**

B Appendix B **III**

C Appendix C **V**

List of Figures

2.1 Structure of a typical 3-layer feedforward MLP. 10
2.2 Sigmoid Activation Function 12
2.3 Tanh Activation Function 12
2.4 ReLU Activation Function 13
2.5 Basic RNN Architecture. 18
2.6 Basic LSTM Architecture. 19

3.1 System architecture and workflow for real-time anomaly detection (Acquisition → Preprocessing → Training → Evaluation → Deployment). 24
3.2 Example waveform of Severe Dip windows (Stage 2, 40–60% drop). The plot shows the first 0.1 s segment of the voltage signal sampled at 1 kHz. 26
3.3 Example of a Normal signal with baseline −48 V and ±0.05 V uniform noise. 27
3.4 Example of a Square Ripple signal with baseline −48 V, ±0.8 V ripple amplitude, and 50% duty cycle at a sampling frequency of 100 Hz. . . 28
3.5 Example of a Mild Voltage Dip signal with baseline −48 V, 10%–20% drop amplitude, and duration of 1–15 ms per event. 29
3.6 Example of a Severe Voltage Dip signal with baseline −48 V, 20%–100% drop amplitude, and duration of 1–15 ms per event over 200 s. 30
3.7 Comparison of dataset splitting strategies. (a) Random split can cause time leakage, where temporally adjacent windows from the same disturbance event appear in both train and test sets, leading to overly optimistic accuracy. (b) Grouped split ensures that entire contiguous 1 s blocks remain in the same subset, preventing such leakage. 33
3.8 1D CNN: local feature extraction via two convolution–pooling stages followed by a compact classifier head. 35
3.9 LSTM: direct temporal modeling over 20-sample sequences with a lightweight dense head. 36
3.10 Hybrid CNN–LSTM: convolutions for local features followed by an LSTM for temporal aggregation. 37

4.1 Training/validation accuracy and loss for 1D-CNN. 46

4.2 Confusion matrices for the 1D-CNN model before quantization, evaluated on the independent 40k test dataset. The left subfigure (a) shows the standard confusion matrix, while the right subfigure (b) presents the row-normalized version. 47
4.3 Training/validation accuracy and loss for LSTM. 48
4.4 Confusion matrices for the LSTM model before quantization, evaluated on the independent 40k test dataset. The left subfigure (a) shows the standard confusion matrix, while the right subfigure (b) presents the row-normalized version. 49
4.5 Training/validation accuracy and loss for Hybrid. 50
4.6 Confusion matrices for the Hybrid CNN-LSTM model before quantization, evaluated on the independent 40k test dataset. The left subfigure (a) shows the standard confusion matrix, while the right subfigure (b) presents the row-normalized version. 51
4.7 Accuracy comparison before and after quantization. 53
4.8 Confusion matrices for the 1D-CNN model after INT8 quantization. The left subfigure (a) shows the standard confusion matrix, while the right subfigure (b) presents the row-normalized version. 54
4.9 Comparison of parameters, model size, and latency across models (original Keras models). 57
4.10 Comparison of accuracy among the three models before quantization. 57

A.1 Examples of ripple waveforms: (top) 200 Hz and (bottom) 400 Hz. . . I
A.2 Voltage Waveforms of Severe Dips at Four Depth Ranges. I

B.1 ROC curves for each disturbance class: (a) normal, (b) ripple, (c) milddip, and (d) severedip. III
B.2 Precision–Recall (PR) curves for each disturbance class: (a) normal, (b) ripple, (c) milddip, and (d) severedip. IV

C.1 ROC and PR curves for the normal class, comparing FP32 and INT8 quantized 1D-CNN. V
C.2 ROC and PR curves for the ripple class, comparing FP32 and INT8 quantized 1D-CNN. V
C.3 ROC and PR curves for the milddip class, comparing FP32 and INT8 quantized 1D-CNN. VI
C.4 ROC and PR curves for the severedip class, comparing FP32 and INT8 quantized 1D-CNN. VI

List of Tables

3.1 Summary of synthetic datasets for Nominal and Ripple. 26
3.2 Summary of synthetic datasets for Voltage Dips. 26
3.3 Sensor and control signal configuration on STM32G474. 39

4.1 Performance on the test set before quantization 46
4.2 Training and validation scores of the 1D CNN model at different learning rate stages during training. 47
4.3 Training and validation scores of the LSTM model at representative epochs during training. The last row reports the final test set performance. 48
4.4 Training and validation scores of the Hybrid CNN-LSTM model at representative epochs during training. 50
4.5 Performance comparison before and after quantization 54
4.6 Comparison of model accuracy, size, and latency before and after quantization. Original Keras models VS TFLite models (INT8 or mixed) 56

1
Introduction

The rapid expansion of telecommunication networks has intensified the demand for robust and efficient power delivery systems in base stations and related infrastructure. In particular, −48 V power input stages—widely adopted in radio and baseband units—must remain stable under diverse and often harsh electrical conditions[1, 2]. Power-line disturbances (PLDs)[2], such as ripple, voltage dips and transient surges, pose serious risks to system stability, potentially causing performance degradation, service interruptions or even hardware damage. Traditional threshold-based fault detection methods are straightforward to implement but lack adaptability to diverse disturbance patterns and dynamic operating environments.

Recent advances in machine learning (ML) and embedded computing have opened opportunities for real-time anomaly detection directly on resource-constrained microcontrollers (MCUs). With compact and computationally efficient models, it has become feasible to deploy intelligent detection algorithms at the edge, enabling faster response, reduced reliance on centralized processors and enhanced resilience of telecommunication power systems. This thesis investigates the design, evaluation and deployment of lightweight ML models for PLD classification, with the goal of meeting the stringent accuracy, latency and memory requirements of embedded platforms.

1.1 Background

Modern telecommunication infrastructure relies on stable and uninterrupted power delivery to ensure the continuous operation of critical components such as radio units, baseband processors and network switching equipment. Among these, −48 V direct current (DC) power input stages are widely adopted due to their safety, efficiency and compliance with industry standards [3]. These stages typically incorporate surge protection, filtering and backup power integration to ensure robustness under varying load conditions.

In practice, power lines are frequently exposed to PLDs such as voltage dips, ripple waveforms and transient surges [2]. These may originate from sudden load changes, grid fluctuations, lightning-induced transients or backup power switching. Without timely detection and mitigation, PLDs can degrade system performance, disrupt services or damage sensitive hardware.

Traditional detection mechanisms—such as fixed threshold-based monitoring—provide low-cost implementations but are inherently limited in handling diverse disturbance patterns and dynamic operating environments [4]. This shortcoming has motivated the adoption of data-driven approaches, particularly machine learning, which can automatically learn disturbance signatures from measurement data.

Recent advances in embedded computing have made it feasible to deploy ML-based detection models on MCUs. This edge-computing paradigm enables real-time, on-device decision-making, reducing reliance on centralized processors, lowering latency and enhancing system resilience [5]. Such benefits are critical in telecommunication contexts, where even short interruptions can lead to significant operational and economic consequences.

1.2 Motivation

Ensuring the robustness of $-48\,\mathrm{V}$ power input stages is essential for maintaining the reliability of modern telecommunication systems. In practice, PLDs are inevitable due to environmental influences, equipment aging and complex interactions within the power distribution network. These disturbances exhibit diverse temporal and spectral characteristics, making accurate and timely detection particularly challenging.

Conventional threshold-based detection schemes, though simple and cost-effective, are inadequate for handling the variability and complexity of PLD patterns [6]. Fixed thresholds may either overlook subtle anomalies or generate excessive false alarms when disturbance profiles deviate from predefined limits. Such limitations can lead to delayed responses, unnecessary maintenance or in severe cases, service interruptions.

Machine learning provides a promising alternative by enabling models to learn disturbance characteristics directly from historical and real-time measurements, without relying on manually crafted rules [7]. By exploiting patterns in both time and frequency domains, ML models can achieve higher detection accuracy, adaptability to unseen disturbances, and robustness to noise. However, practical deployment in telecommunication systems introduces additional constraints: models must operate on resource-limited MCUs with strict memory budgets, low power availability and real-time processing demands.

The motivation of this thesis is to bridge the gap between high-performance ML algorithms and the practical limitations of embedded hardware in telecommunication power systems. By developing lightweight yet accurate models with optimized for inference speed and memory efficiency, it becomes feasible to perform on-device PLD classification, thereby reducing dependency on centralized computation, minimizing latency and enhancing overall system resilience.

1.3 Research Questions

The overarching goal of this thesis is to investigate and implement machine learning techniques for real-time classification of PLDs on resource-constrained MCUs in $-48\,\mathrm{V}$ telecommunication power input stages. To achieve this objective, the following research questions are addressed:

1. **Model Design:** Which lightweight ML architectures, such as one-dimensional convolutional neural networks (1D-CNNs), recurrent models like long short-term memory (LSTM) or hybrid CNN+LSTM designs, can deliver high classification accuracy while remaining suitable for MCU deployment under strict memory and latency constraints?

2. **Embedded Deployment:** How can these models be quantized and optimized using frameworks such as TensorFlow Lite to minimize memory footprint and inference time without incurring significant accuracy loss?

3. **Performance Evaluation:** Under realistic disturbance scenarios, how do different models compare in terms of accuracy, macro-F1 score, latency and memory usage both before and after quantization?

4. **Practical Feasibility:** Can the selected model meet real-time processing requirements on STM32-class MCUs when integrated into a telecommunication power system and what trade-offs arise between detection accuracy and resource efficiency?

By systematically addressing these questions, this thesis seeks to provide both a theoretical framework and a practical pathway for deploying ML-based PLD detection in embedded telecommunication systems.

1.4 Related Work

1.4.1 Conventional Protection and Detection Methods

In telecommunication power systems, conventional protection strategies primarily rely on fixed threshold-based detection [8]. Surge protection devices, filtering circuits and hot-swap controllers are widely employed to safeguard the $-48\,\mathrm{V}$ input stages against transient events and overvoltage conditions. While cost-effective and straightforward to implement, these approaches lack adaptability to diverse and evolving disturbance patterns. Threshold-based schemes are effective in predefined scenarios but often fail to capture more complex conditions such as multi-frequency ripple components or subtle voltage dips induced by intermittent load switching.

1.4.2 Machine Learning for Power Disturbance Detection

The integration of machine learning into power quality monitoring has enabled data-driven classification of disturbances without extensive manual feature engineering. Early studies commonly employed multilayer perceptrons (MLPs) and support vector machines (SVMs) to classify events such as sags, swells and harmonics [6], typically using offline datasets and computationally intensive models. However, these approaches were unsuitable for deployment on resource-limited microcontrollers.

More recent work has investigated lightweight neural architectures, including 1D-CNNs, recurrent neural networks (RNNs) such as LSTM [9, 10], and hybrid CNN+RNN designs. These models demonstrate improved robustness to noise and variability in disturbance signals. Nevertheless, the majority of studies remain oriented toward server- or desktop-based environments, where computational and memory constraints are less critical.

1.4.3 Embedded Deployment and Quantization

Deploying ML models on microcontrollers imposes strict limitations on memory, compute capability and energy consumption. To address these challenges, various model compression techniques—such as pruning, weight sharing and quantization—have been developed [10]. In particular, full-integer (INT8) quantization using TensorFlow Lite [11] has proven effective for reducing model size and latency while preserving acceptable accuracy. Prior research has shown that compact CNN models can be reduced to kilobyte-scale storage requirements and still perform real-time inference on devices with limited random-access memory (RAM) and flash memory.

Despite these advances, relatively few studies have targeted ML-based PLD classification in $-48\,\mathrm{V}$ telecommunication systems. Existing embedded ML research has largely emphasized general anomaly detection or industrial condition monitoring, leaving a gap in domain-specific, resource-aware PLD solutions.

1.4.4 Positioning of This Work

This thesis aims to bridge this gap by integrating domain-specific PLD data with the design of lightweight, quantized neural networks tailored for MCU deployment. Through a systematic evaluation of 1D-CNN, LSTM and CNN+LSTM models under both floating-point and quantized conditions, this work contributes practical insights into achieving accurate, low-latency and memory-efficient PLD classification for embedded telecommunication applications.

1.5 Thesis Outline

The remainder of this thesis is organized as follows:

- **Chapter 2 – Methodology:** This chapter details the theoretical foundations and practical methodology of the work. It first reviews strategies to prevent overfitting, including regularization, dropout, early stopping, data augmentation and batch normalization. The principles of CNNs are introduced, followed by their adaptation to 1D-CNNs and the fundamentals of sequential models with emphasis on LSTM for capturing temporal dependencies. The dataset preparation process is then described, including segmentation, labeling and prevention of time leakage. Finally, the evaluation methodology is presented, covering performance metrics (accuracy, F1-score, specificity, confusion matrix) and cross-validation, ensuring rigorous and unbiased model assessment.

- **Chapter 3 – Experimental Setup:** This chapter presents the complete experimental framework, encompassing both hardware and software components. On the hardware side, it specifies the STM32G474 microcontroller and its ADC and comparator configurations. In particular, the focus is on the Hotswap module, a circuit that enables safe insertion and removal of boards under power by controlling inrush current and protecting the system from faults; its input voltage channel (V_{in}) was selected as the primary reference for dataset generation. On the software side, the chapter details the synthetic dataset construction aligned with the Hotswap V_{in} signal characteristics, model architecture design (1D-CNN, LSTM, and Hybrid CNN–LSTM), training hyperparameters, and evaluation metrics. The workflow for converting trained models to quantized TensorFlow Lite Micro (TFLM) format and integrating them into MCU firmware is described, along with the methodology for measuring accuracy, inference latency and memory footprint under deployment constraints.

- **Chapter 4 – Results and Discussion:** This chapter reports the experimental results, covering classification accuracy, macro-F1 score, inference latency, and memory footprint across different models. Both pre- and post-quantization performances are compared, revealing the robustness of CNN-based architectures versus the degradation observed in recurrent models. A deployment-oriented analysis highlights trade-offs between accuracy, latency, and resource usage, identifying the quantized 1D-CNN as the most suitable candidate for real-time MCU applications.

- **Chapter 5 – Conclusion and Future Work:** This chapter synthesizes the main findings, emphasizing that convolutional architectures are inherently more resilient to quantization and thus practical for embedded PLD detection. The broader implications for telecommunication systems are discussed, along with limitations of the current study, such as the restricted disturbance classes and reliance on a single MCU platform. Future directions include extending the dataset to capture more realistic disturbance scenarios, exploring alternative lightweight neural architectures, and integrating the proposed approach into a comprehensive fault management framework.

1.6 Statement on the Use of AI Tools

In accordance with the Chalmers regulations on the use of AI tools in theses [12], this work acknowledges the limited and transparent use of AI-based assistance during the thesis process.

AI tools (specifically OpenAI's ChatGPT5) were used as a supportive resource for the following purposes:

- **Language refinement:** to rephrase sentences, improve clarity, and correct grammar or spelling errors in parts of the text. The final wording and structure were always reviewed and approved by the authors.

- **Translation assistance:** occasionally used to translate complex or lengthy sentences into clearer academic English, particularly when drafting from another language.

AI tools were *not* used to generate novel technical content, figures, experimental data, or analysis results. All research design, experiments, data processing, and technical conclusions were conducted independently by the authors. References were collected and managed using standard academic tools, not generated by AI.
The use of AI tools in this thesis was therefore limited to writing support and presentation improvements, without influencing the technical substance or the scientific conclusions of the work.

2

Theory

This chapter provides the theoretical background necessary for designing and evaluating machine learning models for PLD classification. It first outlines the classification criteria for voltage dips, then introduces key ML concepts and neural network architectures relevant to time-series analysis. Finally, it presents the evaluation metrics that will be used to assess model performance in later chapters.

2.1 Voltage Dip Classification Criteria

We classify voltage dips into two severity levels mild and severe using depth (percentage deviation from nominal) and dwell time as primary determinants, with thresholds aligned to telecom practice for $-48\,\mathrm{V}$ DC rails and the immunity limits in the internal PLD requirement [2]. Detecting such dips matters because transient undervoltage can push front-end converters into undervoltage lockout, trigger logic resets, corrupt in-flight data, cause relay or protection chatter, and induce large recovery inrush currents; in telecom systems even brief dips degrade service quality or momentarily drop modules, while deeper or longer events may trip hot-swap or protection circuitry [2].

The depth–time interplay reflects finite ride-through energy: for load P and input capacitance C, the approximate hold-up from V_1 to minimum sustainable V_2 is $t_{\text{hold}} \approx \frac{C}{2P}\,(V_1^2 - V_2^2)$; thus deeper and longer dips are more likely to exhaust hold-up and cause interruption, motivating a joint depth and dwell classification.

2.1.1 Percentage Dip Calculation

Given the nominal level $V_{\text{nom}} = -48\,\mathrm{V}$ and the minimum observed voltage V_{min} within an analysis window, we define the relative dip depth as

$$\Delta\% \;=\; \frac{|V_{\text{nom}}| - |V_{\text{min}}|}{|V_{\text{nom}}|} \times 100\% \tag{2.1}$$

Using equation (2.1), a minimum of $-43.2\,\mathrm{V}$ corresponds to a 10% dip, and $-38.3\,\mathrm{V}$ corresponds to a 20% dip.

2.1.2 Mild Dip (10–20%)

A mild dip satisfies

$$10\% \le \Delta\% < 20\% \quad\Longleftrightarrow\quad -43.2\,\mathrm{V} \le V_{\text{min}}(\mathcal{W}) < -38.3\,\mathrm{V},$$

with $\tau_{-43.2\mathrm{V}}(\mathcal{W}) \geq \tau_{\min}$. Let $\mathcal{W}$ denote a non-overlapping 20 ms window (20 samples at 1 kHz). These events typically arise from momentary load changes or line fluctuations and, per immunity practice, are often tolerated without functional interruption, provided the dwell time remains short [2].

2.1.3 Severe Dip (≥20%)

A severe dip satisfies

$$\Delta\% \geq 20\% \quad \Longleftrightarrow \quad V_{\min}(\mathcal{W}) \geq -38.3\,\mathrm{V},$$

with $\tau_{-38.3\mathrm{V}}(\mathcal{W}) \geq \tau_{\min}$. These deeper sags are characteristic of heavy inrush, transient undervoltage, or partial faults and are more likely to stress downstream converters or trigger resets. In practical systems, sufficiently deep and/or long dips can activate protection or hot-swap logic like disconnection or shut down the supply, after which fine-grained classification ceases to be operationally meaningful [2].

2.1.4 Boundary Handling and Noise Tolerance

To improve robustness under measurement noise (nominal noise $\sigma \approx 0.05\,\mathrm{V}$), we apply a small hysteresis band ϵ around the decision levels:

$$\theta_{10\%} = -43.2\,\mathrm{V} \pm \epsilon, \qquad \theta_{20\%} = -38.3\,\mathrm{V} \pm \epsilon,$$

with $\epsilon = 50\,\mathrm{mV}$ in our experiments. Windows with $V_{\min}$ falling inside the band are resolved by dwell time comparison and continuity with adjacent windows (temporal smoothing), to avoid label flicker near thresholds.

2.2 Machine Learning

Machine learning is a subfield of artificial intelligence focused on enabling systems to learn from data and improve their performance over time. As Brown [13] highlights, instead of depending on manually crafted rules, these algorithms automatically detect patterns within large datasets, making them highly effective for tackling problems with complex, non-linear relationships. This data-driven approach allows models to generalize and infer insights in situations where explicit programming would be impractical. There are three subcategories of machine learning:

2.2.1 Supervised learning

Supervised learning is training models with labeled datasets and during each regression the result will become more accurate with leaning e.g. tuning parameters [13]. This paradigm enables the model to directly learn the mapping from input features to target labels, improving its ability to generalize and minimize prediction error on unseen data.

2.2.2 Unsupervised learning

Unsupervised learning deals with unlabeled data, enabling the discovery of hidden patterns, structures, or trends that may not be immediately apparent to humans. Typical applications include clustering customers based on purchasing behavior, grouping network traffic by similarity for intrusion detection, organizing large collections of documents by topic, and identifying latent structures in high-dimensional scientific data such as genomic sequences or sensor measurements [13].

Such methods are particularly valuable in anomaly detection scenarios where labeled fault data is scarce, as they can learn the distribution of normal operating conditions and identify deviations as potential anomalies.

2.2.3 Reinforcement learning

Reinforcement Learning (RL) is a machine learning paradigm in which an agent learns how to act within an environment by receiving feedback in the form of rewards or penalties [13]. The agent's objective is to maximize its cumulative reward over time, which encourages behaviors that are beneficial and discourages those that are not.

Unlike supervised learning, where models are trained on fixed input–output pairs, RL emphasizes sequential decision-making under uncertainty. Each action taken by the agent not only yields an immediate outcome but also influences future states and rewards, creating a dynamic feedback loop.

Applications of RL span diverse domains, including autonomous robotics, strategic game playing (e.g., AlphaGo), recommendation systems, and adaptive resource management [14]. Its strength lies in the ability to improve performance through trial and error, even in environments where explicit labels or complete knowledge are unavailable.

2.3 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models inspired by the human brain's network of neurons. Instead of biological neurons, ANNs use interconnected artificial neurons arranged in layers to process information, detect patterns, and generate predictions [14]. Through a learning process—adjusting connection weights based on data—these networks can solve complex tasks ranging from image recognition to personalized recommendation systems. There are many different kinds of neural networks, including feedforward networks, CNNs, RNNs, autoencoders, and graph neural networks. Among these, the most common and foundational architecture is the feedforward MLPs, which consists of an input layer, one or more fully connected hidden layers, and an output layer. In an MLP each neuron in one layer is connected to every neuron in the next layer, and information flows strictly forward without any cyclic or temporal feedback from inputs to outputs. This simplicity

makes MLPs easy to implement, train with backpropagation, and adapt to a wide variety of tasks ranging from regression to classification.

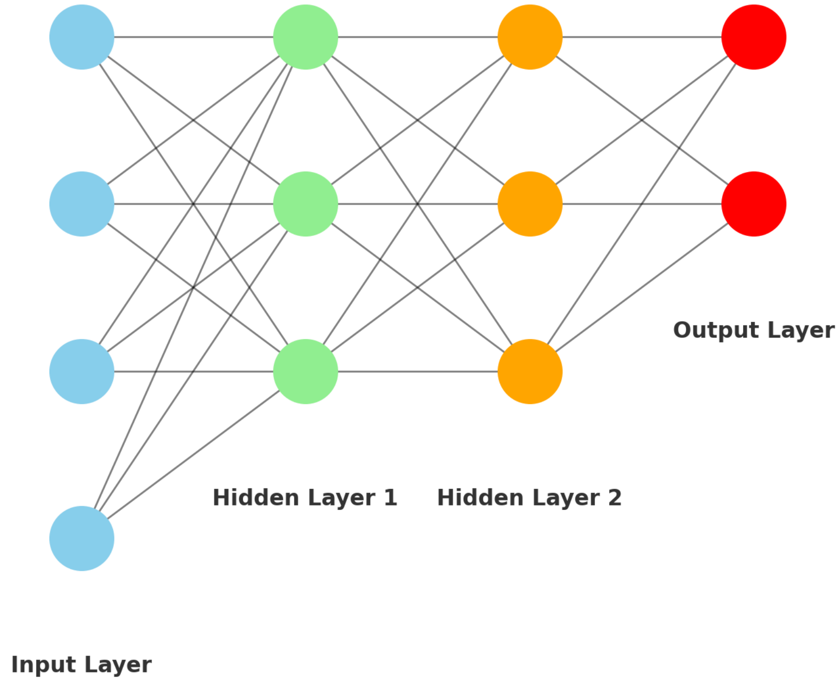


Figure 2.1: Structure of a typical 3-layer feedforward MLP.

2.3.1 Feed-forward and Optimization

The feed-forward pass of a neural network computes each layer's pre-activation as a weighted sum of the previous layer's outputs plus a bias, followed by a nonlinear activation. For layer L and node i, this is

$$V_i^{(L)} = g^{(L)}\left(\sum_{j=1}^{N_{L-1}} V_j^{(L-1)} w_{ji}^{(L)} + b_i^{(L)}\right), \qquad i = 1, \ldots, N_L \tag{2.2}$$

where N_{L-1} and N_L denote the number of neurons in layers $L-1$ and L, respectively. Here $V_i^{(L)}$ represents the output (post-activation value) of the i-th neuron in layer L. The terms $w_{ji}^{(L)}$ and $b_i^{(L)}$ are the weight connecting neuron j in the previous layer to neuron i in the current layer, and the bias of neuron i, respectively. Finally, $g^{(L)}(\cdot)$ denotes the nonlinear activation function applied at layer L. The activation function is crucial as it introduces non-linearity into the model, enabling the network to learn complex patterns. Common choices include sigmoid, tanh, and ReLU (Rectified Linear Unit) [15].

In a feedforward neural network, weights ($w^{(L)}$) and biases ($b^{(L)}$) are two critical parameters that determine the strength of connections between neurons and the neurons' outputs.

Weights ($w^{(L)}$): Weights are the connection parameters in a neural network that quantify the strength of links between neurons. During the forward pass, each input is multiplied by its corresponding weight and then summed, which directly influences the neuron's output. The magnitude and sign of a weight determine how strongly an input contributes to the final activation and in what direction.

Biases ($b^{(L)}$): A bias is an additional parameter in a neural network that shifts the activation function's input, similar to the intercept term in a linear equation. By allowing the activation to be offset from zero, biases enable neurons to model more complex functions and ensure that neurons can activate even when all inputs are zero.

The loss function provides a mathematical measure of the discrepancy between a model's predictions and the true target values [16]. Training proceeds by minimizing a loss function via backpropagation and gradient descent. A basic weight update rule is

$$w_t = w_{t-1} - \alpha \frac{\partial \mathcal{L}}{\partial w_{t-1}} \tag{2.3}$$

where w_t denotes the parameter value at iteration t, α is the learning rate, and $\mathcal{L}$ is the loss function. The learning rate controls the step size of each update: if α is set too high, the parameter updates may overshoot the optimum, causing oscillation or divergence; if too low, training becomes slow and may stagnate in suboptimal regions. A properly chosen learning rate therefore balances speed and stability, enabling the optimization process to converge efficiently.

A more advanced optimizer, Adam [17], is a stochastic optimization method introduced by OpenAI. It builds upon SGD (stochastic Gradient Descent) by integrating ideas from Adagrad (Adaptive Gradient) and RMSProp (Root Mean Square Propagation), adapting per-parameter learning rates using first and second moment estimates of the gradient to keep computational costs low. Its key advantages include update magnitudes that are invariant to the scale of the gradients; step sizes constrained by user-defined hyperparameters; no requirement for a fixed objective function; natural support for sparse gradients; and an inherent step-size annealing mechanism. Its update at time step t is

$$\theta_t = \theta_{t-1} - \alpha \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon} \tag{2.4}$$

where θ_t denotes the parameter vector at iteration t, α is the base step size, $\widehat{m}_t$ and $\widehat{v}_t$ are the bias-corrected first- and second-moment estimates of the gradient, and ϵ is a small constant included to prevent division by zero.

2.3.2 Activation Functions

Different activation functions have distinct characteristics that influence convergence speed, gradient stability, and overall performance.

The sigmoid function maps its input to the range $(0, 1)$ via

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

The sigmoid is smooth and differentiable, making it suitable for binary classification, but can suffer from vanishing gradients and output non-zero-centered activations, which may slow convergence [19].

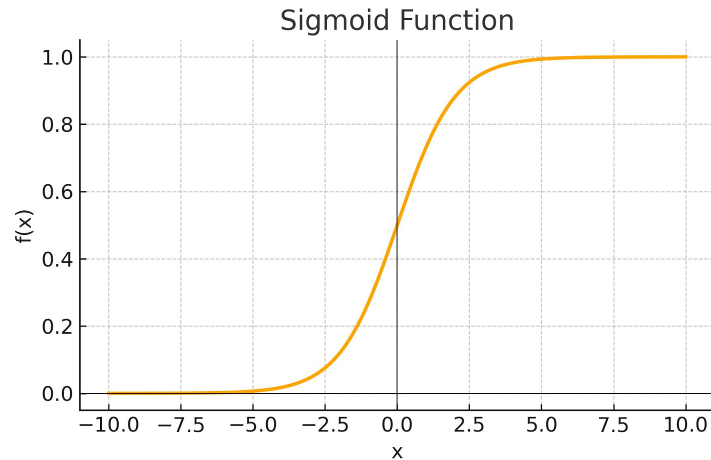


Figure 2.2: Sigmoid Activation Function

The tanh function scales inputs to $(-1, 1)$:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.6}$$

Compared to sigmoid, tanh is zero-centered and has steeper gradients, often leading to faster convergence; however, it similarly suffers from gradient saturation for large $|x|$ [20].

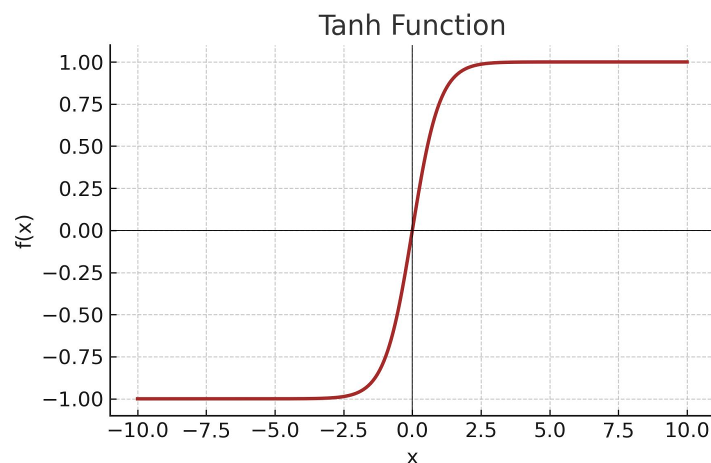


Figure 2.3: Tanh Activation Function

ReLU is defined as

$$\mathrm{ReLU}(x) = \max(0, x) \tag{2.7}$$

It is computationally efficient and alleviates vanishing gradients by maintaining constant positive gradients, but can lead to "dead" neurons when inputs stay negative [21]. Variants such as leaky ReLU address this issue by allowing a small slope for $x < 0$.

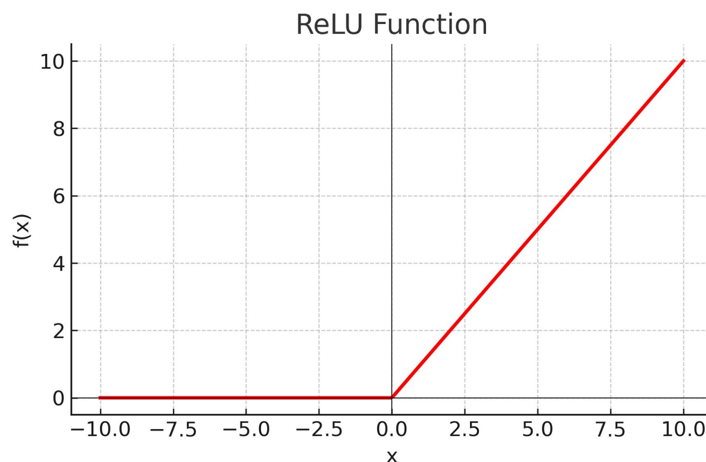


Figure 2.4: ReLU Activation Function

2.3.3 Loss Function

A loss function is a non-negative real-valued function that measures the discrepancy between the model's prediction $f(x)$ and the true label Y. Formally, it is denoted as

$$\mathcal{L}(Y,\, f(x)) \tag{2.8}$$

where a smaller value of $\mathcal{L}$ indicates closer alignment between predictions and ground truth, and thus a better performing and more robust model.

During training, each mini-batch of inputs is passed through the network via forward propagation to produce predictions. The loss function then computes the error between these predictions and the actual labels. This error signal is propagated backward through the network (backpropagation), and the model parameters are updated to minimize the loss. By iteratively reducing $\mathcal{L}$, the network's outputs converge toward the true values, enabling effective learning [22].

For multi-class classification, the cross-entropy loss is defined as:

$$\mathcal{L}(Y \mid f(x)) \;=\; -\sum_{i=1}^{N} Y_i \,\log \hat{y}_i \tag{2.9}$$

where N is the number of classes, Y_i is the one-hot encoded ground-truth label for class i, and $\hat{y}_i = f(x)_i$ is the predicted probability assigned by the model to class

i. A one-hot encoding represents the true label as a binary vector of length N, where the entry corresponding to the correct class is 1 and all other entries are 0. Cross-entropy originates from information theory as a measure of dissimilarity between two probability distributions, and in neural networks it quantifies how well the predicted probability distribution $\hat{y}$ matches the ground truth Y.

In practice, cross-entropy is the most commonly used loss for convolutional and fully connected classifiers because it directly optimizes probabilistic outputs and helps mitigate vanishing gradients. In the binary case (two classes), it is often called *log loss* and simplifies to:

$$\mathcal{L}(p, y) = \begin{cases} -\log(p), & y = 1 \\ -\log(1-p), & y = 0 \end{cases} \tag{2.10}$$

where p is the model's predicted probability of the positive class. y is the true label.

When the dataset is imbalanced, it is common to introduce a weighting factor α for the positive class to penalize misclassification differently:

$$\mathcal{L}(p, y) = \begin{cases} -\alpha \log(p), & y = 1 \\ -\big(1-\alpha\big) \log(1-p), & y = 0 \end{cases} \tag{2.11}$$

This weighted cross-entropy ensures that errors on the minority class contribute more heavily to the loss, improving detection performance on underrepresented classes.

2.3.4 Regularization

Overfitting typically occurs when a model is excessively complex relative to the amount of training data or when it is trained for too many iterations without adequate regularization. In such cases, the model not only captures the true underlying patterns but also memorizes noise and idiosyncrasies in the training set. As a result, it achieves very low error on training data but performs poorly on unseen validation or test data, indicating a lack of generalization [23]. Regularization adds a penalty term to the loss function to discourage model parameters from growing excessively large. A common approach is ℓ_2 regularization, which incorporates a weighted sum of squared weights into the cross-entropy loss, as shown in equation (2.12).

$$\mathcal{L}(p, y) = -\Big[y \log(p) \;+\; (1-y) \log(1-p)\Big] \;+\; \lambda \sum_j w_j^2 \tag{2.12}$$

where p is the model's predicted probability, y the true label, $\{w_j\}$ the network weights, and λ the weight decay parameter to control penalty strength. This biases the optimizer toward simpler models that are less likely to memorize noise in the training data [23].

Dropout [23] is an another way to avoid overfitting, which can randomly deactivate a fraction p of neurons during each training iteration, forcing the network to

learn redundant representations and reducing co-adaptation of features. At inference time, all neurons remain active, and their outputs are scaled appropriately.

To further guard against overfitting, we can apply several complementary strategies:

- **Early stopping** [23]: Terminates training as soon as validation loss fails to improve for a predefined number of epochs, preventing the model from memorizing noise.

- **Data augmentation** [23]: Generates synthetic variants of time-series windows—by adding Gaussian noise, shifting samples in time, or scaling amplitudes—to broaden the effective training distribution.

- **Batch normalization** [24]: Standardizes each layer's inputs using mini-batch statistics, which not only accelerates convergence but also injects stochasticity into the network, improving robustness.

By combining these strategies—penalizing large weights, randomly perturbing activations, halting training at the optimum point, and enriching the dataset—neural networks achieve more robust performance on unseen data without excessive complexity.

2.4 Convolutional neural networks (CNNs)

CNNs are specialized deep architectures that automatically learn spatial hierarchies of features through three main operations: convolution, pooling, and fully connected classification [25]. In a convolutional layer, given an input feature map $X \in \mathbb{R}^{H \times W}$ and a filter $W^k \in \mathbb{R}^{H \times W}$, the output feature map Y^k is computed as

$$Y^k(i,j) = \sum_{u=1}^{h} \sum_{v=1}^{w} X(i+u-1, j+v-1)\, W^k(u,v) + b^k \tag{2.13}$$

where b^k is the bias term. Pooling layers then downsample these feature maps; for max-pooling over region $\mathcal{R}(i,j)$, the activation is

$$P(i,j) = \max_{(u,v) \in \mathcal{R}(i,j)} Y(u,v) \tag{2.14}$$

Finally, fully connected layers flatten the pooled features into a vector $\mathbf{z} \in \mathbb{R}^M$, where M is the number of inputs to the fully connected layer. The pre-activation for neuron i in this layer is then computed as

$$a_i = \sum_{j=1}^{M} W_{ij}^{\mathrm{fc}}\, z_j + b_i^{\mathrm{fc}}, \qquad i = 1, \ldots, N_{\mathrm{fc}} \tag{2.15}$$

where W_{ij}^{fc} and b_i^{fc} are the weight and bias of the fully connected layer, and N_{fc} is the number of output neurons. A subsequent softmax activation converts these

pre-activations into class probabilities:

$$\hat{y}_i = \frac{\exp(a_i)}{\sum_{j=1}^{N_{\mathrm{fc}}} \exp(a_j)}, \qquad i = 1, \ldots, N_{\mathrm{fc}} \tag{2.16}$$

where a_i is the pre-activation of class i, and $\hat{y}_i$ is the predicted probability assigned to class i.

Traditional fully connected networks require a distinct weight for every connection between M inputs and N outputs, resulting in $M \times N$ parameters that quickly become impractical for large layers. Building convolutional layers overcomes this by exploiting three key principles: sparse connectivity, where each neuron interacts only with a small, local receptive field; parameter sharing, which reuses the same filter weights across all spatial locations; and translation invariance, ensuring that learned features can be detected regardless of their position in the input. Together, these innovations allow CNNs to capture local patterns with far fewer parameters and improved generalization.

2.5 1D Convolutional Neural Networks (1D-CNN)

CNNs are not limited to two-dimensional inputs like images; they can be adapted to one-dimensional signals by using $1{\times}K$ kernels that slide along the temporal axis. In a 1D-CNN, an input sequence $\mathbf{x} \in \mathbb{R}^L$ of length L is convolved with M learnable filters $\mathbf{w}^k \in \mathbb{R}^K$, producing feature maps y^k according to

$$y^k(i) = \sum_{j=1}^{K} x(i+j-1)\, w_j^k \; + \; b^k, \quad k = 1, \ldots, M \tag{2.17}$$

where b^k is a trainable bias term. This operation captures local temporal patterns—such as voltage dips and ripples—while preserving the sequence's continuity. Following convolution, pooling layers (e.g. max-pooling) may downsample the feature maps to reduce computational load and introduce temporal translation invariance. The pooled representations are then flattened and passed through one or more fully connected layers, with all parameters optimized end-to-end via backpropagation. By learning hierarchical features directly from raw time-series data, 1D-CNNs eliminate the need for manual feature extraction and have proven effective in signal pattern recognition tasks.

2.6 Sequential machine learning

Traditional supervised learning models such as MLPs and CNNs treat each input window independently, ignoring temporal context [27]. In contrast, sequential machine learning models explicitly capture dependencies across time steps, making them well suited for tasks on temporal data and prediction. In deep learning, a sequence is an ordered collection of data points that exhibit temporal or logical

continuity. Examples include text (a series of words or characters), audio signals (samples in time), sensor readings (measurements over time), and any structured data where position and order convey meaningful information (e.g., DNA bases, user click streams). By preserving the order and dependencies between elements, sequential models can learn patterns that unfold over time or within a fixed context, enabling tasks such as language modeling, speech recognition, and anomaly detection in time-series data.

There are many algorithms used in sequence modeling, ranging from classical statistical approaches—such as ARIMA (AutoRegressive Integrated Moving Average) and Hidden Markov Models (HMMs)—to modern neural architectures like RNNs, Gated Recurrent Units (GRUs), LSTMs, and Transformer-based models [28]. Standard RNNs can model sequential dependencies but suffer from vanishing or exploding gradients over long horizons. GRUs mitigate this issue by combining the forget and input gates into a single update gate, reducing parameter count and simplifying training. LSTMs go a step further with separate input, forget, and output gates plus a dedicated cell state, enabling them to learn both short and long-range temporal patterns effectively [28]. More recently, Transformer architectures have demonstrated superior performance by replacing recurrence with self-attention mechanisms, allowing the model to relate all positions in a sequence directly and in parallel [29].

In this work, we concentrate on LSTM networks due to their proven ability to maintain stable gradients over extended sequences and to selectively retain or forget information that are critical for accurately detecting and classifying transient anomalies in power input time-series data.

2.6.1 Recurrent Neural Networks (RNNs)

RNNs are tailored for sequential inputs by retaining a hidden state that aggregates information over time. At each time step t, an RNN cell updates its hidden state $\mathbf{h}_t$ and produces an output $\mathbf{o}_t$ according to

$$\mathbf{h}_t = \phi\Big(U\,\mathbf{x}_t + V\,\mathbf{h}_{t-1} + \mathbf{b}_h\Big), \quad \mathbf{o}_t = \psi\Big(W\,\mathbf{h}_t + \mathbf{b}_o\Big), \tag{2.18}$$

where

- $\mathbf{x}_t$ is the input vector at time t,
- U, V, and W are learnable weight matrices mapping input $\rightarrow$ hidden, hidden $\rightarrow$ hidden, and hidden $\rightarrow$ output, respectively,
- $\mathbf{b}_h$ and $\mathbf{b}_o$ are bias vectors,
- ϕ (e.g. tanh or ReLU) and ψ (e.g. softmax for classification) are activation functions.

Unfolding the recurrence over T steps shows that $\mathbf{h}_T$ integrates information from all past inputs $\{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$, thereby enabling the network to capture temporal dependencies. This process is illustrated in Figure 2.5. In practice, however, standard RNNs often encounter vanishing or exploding gradients during backpropagation

through time, which either diminish early-step gradients to near zero—hindering long-range learning—or amplify them uncontrollably destabilizing training. Consequently, learning dependencies over long sequences becomes challenging with vanilla RNN cells.

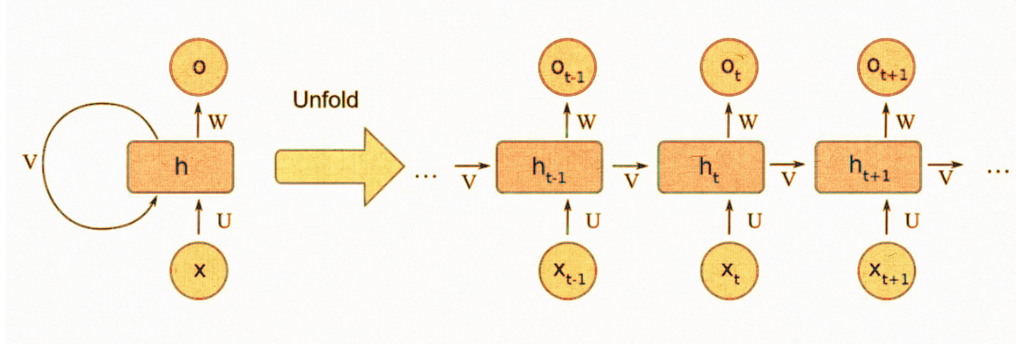


Figure 2.5: Basic RNN Architecture.

2.6.2 Long Short-Term Memory (LSTM)

LSTMs are an extension of RNNs designed to mitigate the vanishing and exploding gradient problems by introducing an explicit memory cell and three gates that regulate the flow of information. At each time step t, the LSTM takes as input the current feature vector $\mathbf{x}_t$, the previous hidden state $\mathbf{h}_{t-1}$, and the previous cell state $\mathbf{c}_{t-1}$. The updates are defined as follows:

$$\begin{aligned}
\mathbf{f}_t &= \sigma\Big(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + \mathbf{b}_f\Big) && \text{(forget gate)},\\
\mathbf{i}_t &= \sigma\Big(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + \mathbf{b}_i\Big) && \text{(input gate)},\\
\tilde{\mathbf{c}}_t &= \tanh\Big(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1} + \mathbf{b}_c\Big) && \text{(candidate cell state)},\\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t && \text{(new cell state)},\\
\mathbf{o}_t &= \sigma\Big(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + \mathbf{b}_o\Big) && \text{(output gate)},\\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t) && \text{(new hidden state)}.
\end{aligned}$$

Here:

- $\sigma(\cdot)$ is the sigmoid activation that outputs values in $(0, 1)$, used to control how much information passes through each gate.
- $\tanh(\cdot)$ is the hyperbolic tangent activation that squashes values into $(-1, 1)$.
- $\odot$ denotes element-wise multiplication.
- $\mathbf{f}_t, \mathbf{i}_t, \mathbf{o}_t$ are the forget, input, and output gates, respectively.
- $\mathbf{c}_t$ is the cell state, which acts as long-term memory.
- $\mathbf{h}_t$ is the hidden state, used for short-term memory and output to the next layer.
- W_* and U_* are weight matrices applied to the input and hidden state, and $\mathbf{b}_*$ are their corresponding bias vectors.

The forget gate $\mathbf{f}_t$ decides what fraction of the previous memory $\mathbf{c}_{t-1}$ should be retained. The input gate $\mathbf{i}_t$ and candidate state $\tilde{\mathbf{c}}_t$ determine what new information

to add to the memory. The output gate $\mathbf{o}_t$ controls how much of the memory should be revealed as the new hidden state $\mathbf{h}_t$. This gating mechanism enables LSTMs to learn long-range dependencies effectively, making them well suited for time-series anomaly detection tasks [30]. The LSTM model is illustrated in figure 2.6.

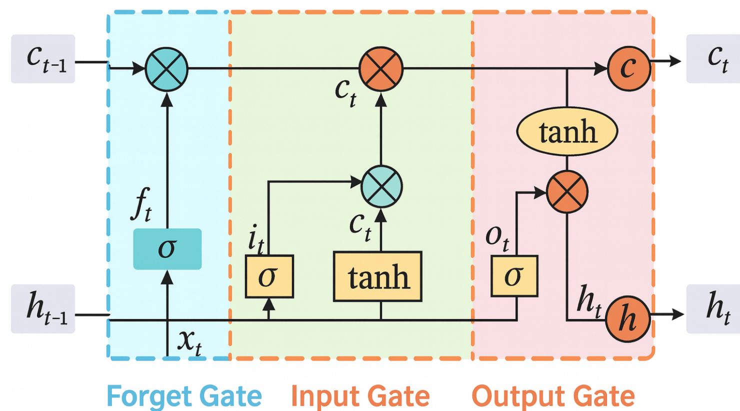


Figure 2.6: Basic LSTM Architecture.

2.7 Evaluation

Machine learning algorithms are typically assessed by partitioning the available data into three subsets: a training set, a validation set, and a test set. The training set is used to fit the model parameters, the validation set is used for hyperparameter tuning and model selection, and the test set provides an unbiased estimate of the model's generalization performance on unseen data. The majority of windows are used to train the model, while a hold-out portion (unseen during training) is reserved for evaluation. By measuring performance on this unseen subset, we gain an unbiased estimate of how the model is likely to behave in real-world applications.

During training, we monitor performance on a separate validation set to guide hyperparameter tuning and architecture selection while guarding against overfitting. Once training is complete, we then evaluate the finalized model on a distinct test set to measure its generalization ability—that is, its predictive accuracy on data it has never seen before.

Effective assessment of classification models requires more than a single train/test split. We adopt a suite of complementary metrics with confusion matrices, F_1–score, specificity, and cross-validation augmented by performance matrices to capture both class-wise and overall behavior [31].

2.7.1 F_1–Score

For each class i, let

$$\mathrm{TP}_i,\ \mathrm{FP}_i,\ \mathrm{FN}_i$$

denote true positives, false positives, and false negatives, respectively. Precision and recall are

$$\text{Precision}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i}, \quad \text{Recall}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i}. \tag{2.19}$$

Precision refers to the proportion of true positive predictions among all positive predictions made by the model. It reflects the reliability of positive predictions. Recall is the proportion of actual positive windows that are correctly predicted by the model. It indicates the model's ability to capture positive windows.

The F_1–score, the harmonic mean of precision and recall, is

$$F_{1,i} = 2\,\frac{\text{Precision}_i\ \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \tag{2.20}$$

It provides a balanced measure that takes both false positives and false negatives into account.

To summarize across all C classes, we compute the macro-averaged

$$F_1^{\text{macro}} = \frac{1}{C}\sum_{i=1}^{C} F_{1,i} \tag{2.21}$$

and the weighted average

$$F_1^{\text{weighted}} = \frac{1}{N}\sum_{i=1}^{C} n_i\, F_{1,i} \tag{2.22}$$

where n_i is the number of true instances of class i and $N = \sum_i n_i$ [31].
Macro F_1 is particularly suitable in our thesis because it treats each class equally, regardless of how many windows it contains. This property is important for anomaly detection tasks, where disturbance events such as severe voltage dips may be rare compared to the nominal baseline. If only overall accuracy were reported, the model could achieve seemingly high performance by overemphasizing the majority class (normal signals) while failing to detect infrequent but critical anomalies. By averaging F_1 scores across all classes without weighting by frequency, macro F_1 ensures that the evaluation reflects balanced performance and penalizes models that neglect minority classes [31]. This makes it a more informative and fairer metric for comparing models in imbalanced classification settings.

2.7.2 Specificity

Specificity (true negative rate) evaluates a model's ability to correctly identify non-anomalous windows for each class i:

$$\text{Specificity}_i = \frac{\text{TN}_i}{\text{TN}_i + \text{FP}_i} \tag{2.23}$$

where TN_i is the count of true negatives for class i. For binary tasks, one may also report the balanced accuracy:

$$\text{BalancedAcc} = \frac{\text{Recall}_{\text{pos}} + \text{Specificity}_{\text{pos}}}{2} \tag{2.24}$$

which equally weights sensitivity and specificity to account for class imbalance [31].

2.7.3 K-Fold Cross-Validation

Cross-validation is a statistical method that partitions the dataset into K equal folds. In each of the K rounds, one fold is held out as a validation set while the remaining $K-1$ folds are used for training. By averaging performance across all folds, cross-validation provides a robust estimate of generalization error, reducing the risk of over or under fitting to a single train/validation split. Also this a systematic way to compare and select hyperparameters by observing their impact on validation performance across multiple folds. After trainng the model this is an assessment of model stability, as the variance of metrics across folds indicates how sensitive the model is to different training subsets.

Denoting a metric M on fold k by M_k, we report

$$\overline{M} = \frac{1}{K}\sum_{k=1}^{K} M_k, \quad \sigma_M = \sqrt{\frac{1}{K}\sum_{k=1}^{K}\left(M_k - \overline{M}\right)^2} \tag{2.25}$$

This process provides mean performance and its variance across folds [31].

2.7.4 Performance Matrix

A confusion matrix was generated to visually analyze classification accuracy across individual disturbance categories [31]. The confusion matrix enables detailed assessment of misclassification patterns, highlighting strengths and weaknesses of the models in distinguishing between specific types of disturbances.

Let $\mathbf{M} \in \mathbb{N}^{C\times C}$ be the confusion matrix with entries

$$M_{ij} = \left|\{\, x : y(x) = i,\ \hat{y}(x) = j\}\right|,$$

where $y(x)$ and $\hat{y}(x)$ are true and predicted labels. We define the performance matrix $\mathbf{P}$ by row-normalizing $\mathbf{M}$:

$$P_{ij} = \frac{M_{ij}}{\sum_{j=1}^{C} M_{ij}}, \tag{2.26}$$

so that P_{ij} equals recall for class i, and off-diagonal entries measure misclassification rate [31]. This matrix provides a concise, per-class view of where the model succeeds or struggles.

3
Methods

This chapter presents the methodology for designing and deploying lightweight machine learning models for PLD classification in $-48\,\mathrm{V}$ telecommunication power systems. The workflow covers dataset construction, preprocessing, model development, and quantization, followed by deployment and evaluation on an STM32G474 microcontroller.

3.1 System Architecture and Workflow

This section describes the overall system architecture and workflow employed for implementing real-time anomaly detection in power input systems for radio and baseband. The proposed system comprises several key stages: data acquisition, signal preprocessing, machine learning-based anomaly classification, and deployment on a resource-constrained microcontroller.

The workflow for the proposed real-time anomaly detection system is as follows:

1. **Data Acquisition Stage**: In practice, PLD tests are performed in laboratory environments to characterize the behavior of -48V power inputs under controlled disturbance conditions (e.g., ripple injection, voltage dips, transient surges). While our project did not conduct new hardware-based PLD measurements, these standardized test procedures provided the reference framework for dataset design. Following the PLD specifications, we generated synthetic voltage traces in Python that emulate the same disturbance types—normal operation, ripple, mild dips, and severe dips. This approach ensured consistency with real PLD testing conditions while enabling rapid dataset construction for training and evaluation.

2. **Data Preprocessing Stage**: The acquired PLD reference signals are used to generate synthetic datasets representing ripple (square-wave disturbance), mild voltage dips, and severe voltage dips. The synthetic data are segmented into fixed-length windows, each containing 20 sampling points, followed by normalization to enhance model generalization capability and accelerate convergence during training.

3. **Model Training Stage**: Three distinct machine learning models—a 1D-CNN, a LSTM network, and a hybrid CNN-LSTM model—are designed and

trained to classify the power signals into their corresponding anomaly categories. The training stage encompasses dataset partitioning, hyperparameter optimization, model selection, and validation procedures using held-out synthetic data.

4. **Model Evaluation Stage**: Comprehensive model evaluation is performed using various performance metrics, including accuracy, macro-averaged F1-score (Macro-F1), confusion matrices, Receiver Operating Characteristic (ROC) curves, and Precision-Recall (PR) curves. Additionally, inference latency and memory usage of the models are analyzed for deployment feasibility.

5. **Deployment Stage**: The selected best-performing model is quantized to INT8 precision using TensorFlow Lite Micro to ensure compatibility and optimal performance on the resource-constrained STM32G474. Finally, the deployed model is validated under realistic scenarios with actual power line disturbances to ensure its reliability and robustness in real-world operation.

The proposed architecture and workflow ensure efficient and accurate detection and classification of power anomalies, significantly enhancing the stability and reliability of telecommunication products. The architecture is illustrated in Figure 3.1.

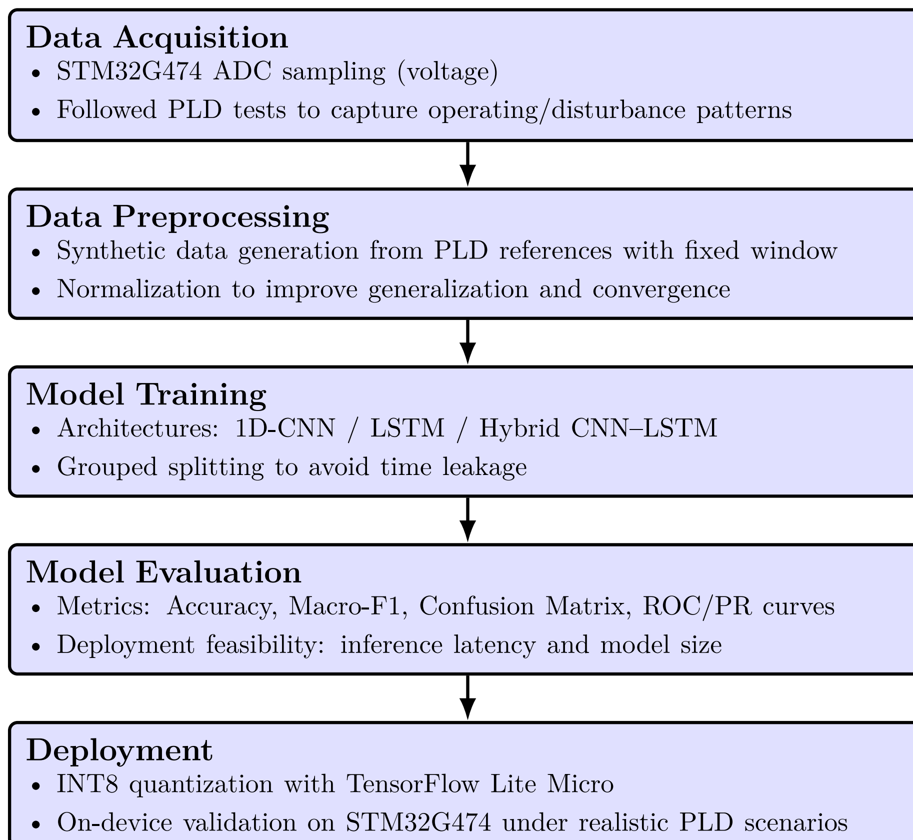


Figure 3.1: System architecture and workflow for real-time anomaly detection (Acquisition → Preprocessing → Training → Evaluation → Deployment).

3.2 Data Collection and Generation

Accurate and representative data are essential for training robust models. In the real situation, finding anomalies is very difficult. In this work, we simulate representative PLD tests in the $-48\,\text{V}$ DC power supply commonly used in radio and baseband using Python, following the internal PLD requirement to set amplitudes, durations, and timing constraints [2]. Each dataset contains only the voltage signal under controlled disturbance scenarios; current is not recorded.

3.2.1 Power Line Disturbance (PLD) Overview

According to PLD specification, a series of immunity requirements and test cases are defined for different application environments, including mast-mounted, outdoor, wall-mounted, and indoor setups. These specifications are designed to simulate real-world electrical disturbances that may occur on the power line. The PLDs studied in this work include square wave disturbance, mild and severe dip.

3.2.2 Dataset Construction

To facilitate the supervised training of anomaly detection models in embedded power systems, we constructed four synthetic datasets emulating typical voltage input conditions encountered in power supply units. These datasets simulate a $-48\,\text{V}$ nominal input with various anomalies based on physical characteristics observed in PLD environments.

Each dataset contains a 200-second voltage trace sampled at 1 ms intervals, resulting in 200,000 samples per dataset. For ease of training and annotation, the data is segmented into non-overlapping 20 ms windows (20 samples per window), each assigned a single label depending on the anomaly type present.

Dataset Summary: Each disturbance type was recorded multiple times using a sampling frequency of 1 kHz (corresponding to a sampling interval of 1 ms per sample point). The final dataset comprises 4 classes (Normal, Square wave ripple, Mild Dip, Severe Dip), with each instance containing synchronized time and voltage measurements. Data were collected from multiple test iterations to ensure variability and generalization. A detailed description of the dataset parameters for the Normal and Ripple classes is provided in Table 3.1, while the characteristics of the Mild and Severe Dip classes are summarized in Table 3.2.

To further illustrate the dataset structure, Figure 3.2 presents an example waveform of Severe Dip windows, highlighting the temporal resolution and disturbance characteristics captured in the data. As shown in the plot, the voltage periodically drops from the nominal baseline of $-48\,\text{V}$ to significantly lower levels (in this case between $-20\,\text{V}$ and $-30\,\text{V}$), corresponding to a 40–60% reduction in amplitude. The dips are characterized by sharp fall times, short low-voltage plateaus, and subsequent rapid recovery phases, all of which occur within a narrow time window of a few millisec-

onds. Additional waveform examples are provided in Appendix A. These transient events repeat several times within 40 s, clearly demonstrating the non-stationary and abrupt nature of severe anomalies. By capturing both abrupt transitions and stable baseline segments at a sampling frequency of 1 kHz, the dataset provides rich temporal patterns for training machine learning models, enabling them to distinguish between normal operation, mild variations, and critical fault-like disturbances.

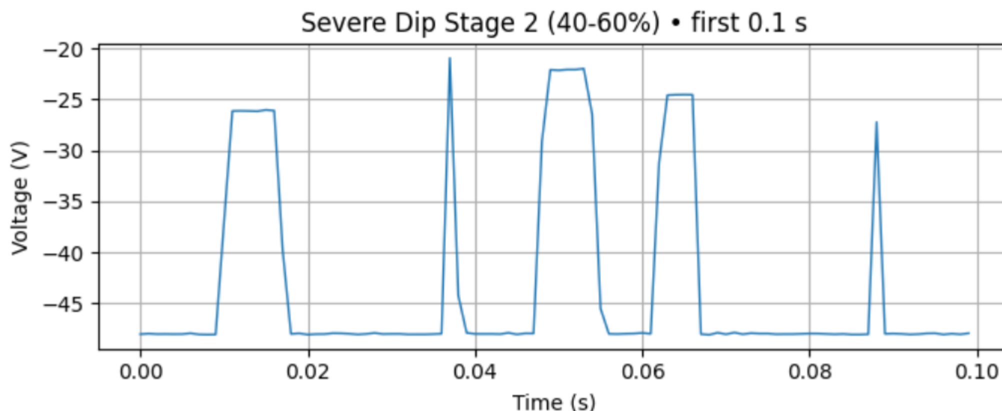


Figure 3.2: Example waveform of Severe Dip windows (Stage 2, 40–60% drop). The plot shows the first 0.1 s segment of the voltage signal sampled at 1 kHz.

Table 3.1: Summary of synthetic datasets for Nominal and Ripple.

Feature	Nominal	Square Ripple
Label	normal	ripple
Duration (ms)	Continuous	Continuous
Range (V)	-48.00 ± 0.05	-48.8 to -47.2
Windows/file	10,000	10,000
Remarks	Stable + uniform noise	±0.8V ripple, 100–500 Hz, 50% duty cycle

Table 3.2: Summary of synthetic datasets for Voltage Dips.

Feature	Mild Voltage Dip	Severe Voltage Dip
Label	milddip	severedip
Duration (ms)	1–15	1–15
Range (V)	-43.2 to -38.3	-38.4 to 0.0
Windows/file	10,000	10,000
Remarks	10%–20% drop	20%–100% drop

Detailed Signal Characteristics: For each measurement, the raw voltage signal was trimmed to remove irrelevant portions and subsequently segmented into non-overlapping windows of 20 consecutive samples, corresponding to a duration of 20 ms at the given sampling rate of 1 kHz. The window length was selected to capture the

complete temporal profile of a single disturbance event within one segment. Each segment contains only the voltage trace and is assigned a class label corresponding to its disturbance type, as defined in the dataset generation process.

3.2.3 Voltage Disturbance Design Criteria

To reflect realistic variations in power input behavior, we designed our simulated dataset. For clarity of visualization, all example waveforms presented in the subsequent figures are shown over a short duration of 0.1 s. Displaying the entire time span would make the plots unnecessarily long and difficult to interpret. It should also be noted that the dataset was generated in Python by following the procedures defined in laboratory PLD tests. Unlike real-world scenarios—where the supply voltage may remain in a normal state for extended periods and disturbances occur infrequently—the synthetic dataset deliberately contains multiple occurrences of each disturbance type within short time spans. This design choice was made to enrich the dataset and ensure sufficient representation of each class for effective model training and learning.

Normal Signal Dataset (Label: normal): The Normal class represents the baseline operating condition of the system under nominal supply voltage without any intentional disturbances. The voltage level is fixed at −48 V, reflecting the standard DC operating point of the target system, with small variations introduced to emulate measurement noise. In the synthetic data generation process, zero-mean uniform noise in the range ±0.05 V was added to the baseline to simulate realistic sensor imperfections and environmental electrical noise.

The resulting signal exhibits high stability over time, with no transient events or periodic fluctuations present. The dataset for this class has a total duration of 200 s at a sampling frequency of 1 kHz, resulting in 200,000 time–voltage pairs. This class serves as the reference category for the anomaly detection model, ensuring that deviations caused by ripple or voltage dips can be accurately identified. An example of the generated Normal waveform is shown in Figure 3.3.

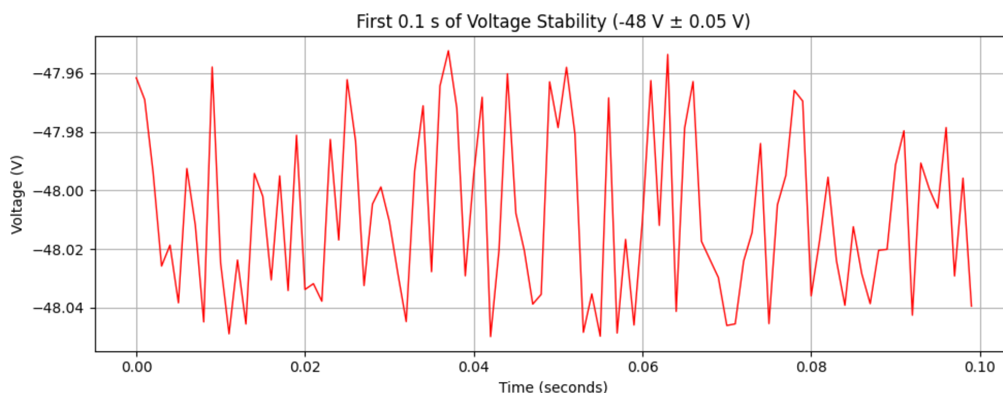


Figure 3.3: Example of a Normal signal with baseline −48 V and ±0.05 V uniform noise.

Square Wave Ripple Dataset (Label: ripple): Square wave disturbances are used to simulate ripple effects or periodic interference that may arise from switching power supplies or external coupling. It represents a nominal baseline voltage with a superimposed periodic square-wave ripple. The baseline is set to −48 V, with an amplitude variation of ±0.8 V (peak-to-peak 1.6 V) and a 50% duty cycle. In this work, the ripple is generated at five discrete frequencies: 100, 200, 300, 400, and 500 Hz, each lasting 40 s, for a total dataset duration of 200 s. Gaussian noise with a standard deviation of 0.05 V is added to simulate realistic measurement conditions. These values were selected to represent realistic ripple behavior encountered in telecommunication power systems.

To further assess the robustness of the detection model under extreme scenarios, an extended frequency sweep up to 1000 Hz was also performed, aligning with conditions from PLD test requirements. Although the analysis is performed within a 20-sample (20 ms) window, the fixed sampling rate of 1 kHz cannot represent content above 500 Hz, irrespective of the window length. This limitation introduces aliasing and signal distortion, which challenges the model's ability to classify such anomalies accurately.

Despite this constraint, the inclusion of higher-frequency ripple patterns provides an opportunity to evaluate the sensitivity of the model to under-sampled periodic signals, which may occur in real-time monitoring environments where memory and timing constraints prevent higher-resolution acquisition. An example of the generated Square Ripple waveform is shown in Figure 3.4.

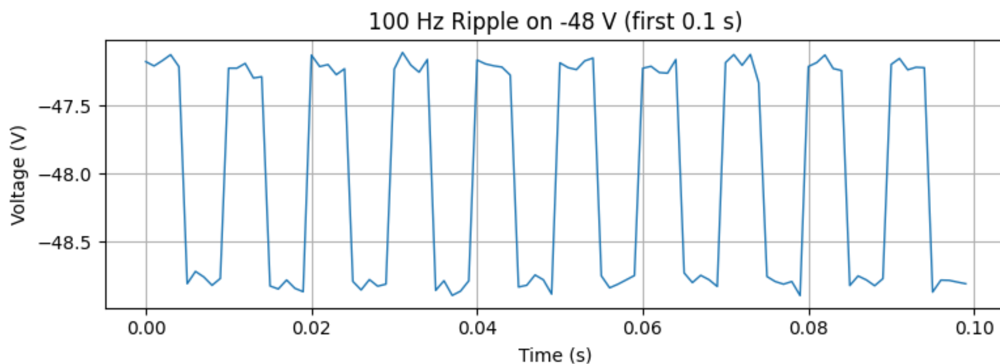


Figure 3.4: Example of a Square Ripple signal with baseline −48 V, ±0.8 V ripple amplitude, and 50% duty cycle at a sampling frequency of 100 Hz.

Voltage Dip 1 Dataset (Label: milddip): The Mild Voltage Dip class models transient events in which the voltage drops by 10%–20% relative to the nominal baseline. This corresponds to a minimum voltage range of −43.2 V to −38.3 V. Each dip event lasts between 1 and 15 ms, and is contained entirely within a 20 ms analysis window. The dips occur once per window, and Gaussian noise ($\sigma = 0.05$ V) is added to all samples. The dataset spans 200 s at a sampling frequency of 1 kHz, yielding 200,000 time–voltage pairs.

The fall time (transition from baseline to dip level) and rise time (transition back to baseline) are each randomly selected between 5% and 20% of the total dip duration, resulting in smooth ramp transitions rather than instantaneous steps. The rise and fall times of the synthetic voltage dip signals in this study were set in the range of 5–20% of the total disturbance duration (i.e., 0.05–3 ms for dip durations between 1–15 ms). This choice differs from the microsecond-scale specifications in the PLD test standard (fall time $\leq 12\ \mu$s, rise time $\leq 5\ \mu$s), and was made deliberately due to the constraints of the dataset sampling frequency.

The synthetic dataset was generated at a sampling rate of 1 kHz, corresponding to a temporal resolution of 1 ms per sample point. At this resolution, microsecond-scale transients would occur between sampling points and thus could not be accurately represented in the recorded waveforms. Instead, millisecond-scale rise and fall transitions were adopted to ensure that the disturbance shape is visible in the sampled data, allowing the classification model to learn and generalize from observable waveform features. This approach preserves the essential dynamic characteristics of voltage dips while avoiding aliasing effects that could arise from attempting to replicate sub-sample (microsecond) transitions in a low-sampling-rate dataset.

This class represents moderate disturbances that may occur due to partial load switching or brief supply interruptions. Also these dips are commonly encountered in real systems and are usually tolerated without triggering protective shutdowns. An example of the generated Mild Voltage Dip waveform is shown in Figure 3.5.

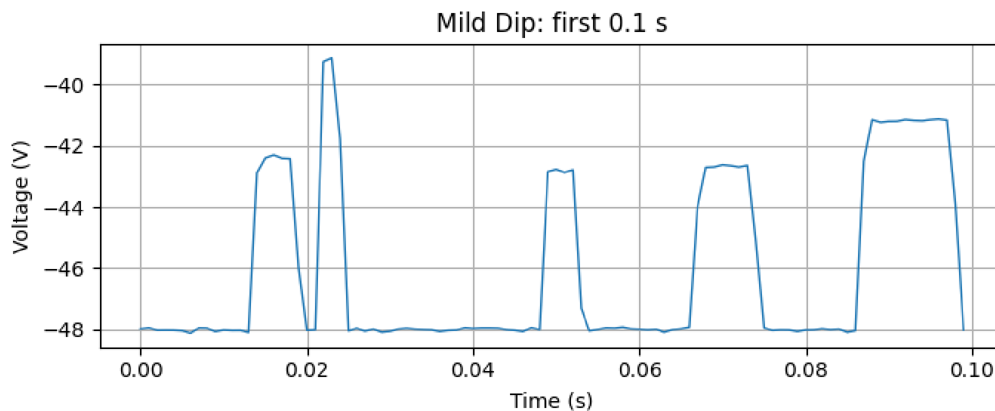


Figure 3.5: Example of a Mild Voltage Dip signal with baseline −48 V, 10%–20% drop amplitude, and duration of 1–15 ms per event.

Voltage Dip 2 Dataset (Label: severedip): The Severe Voltage Dip class models transient events where the voltage drops by 20%–100% relative to the nominal baseline. This corresponds to a minimum voltage range of −38.4 V to 0.0 V. Similar to the Mild Voltage Dip class, each event lasts between 1 and 15 ms and occurs once per 20 ms window, with Gaussian noise ($\sigma = 0.05$ V) added to all samples. The rise and fall transitions were randomly set to 5%–20% of the total dip duration, ensuring that the waveform profile is adequately captured at the 1 kHz sampling rate. The dataset covers 200 s of operation at a sampling frequency of 1 kHz, yielding 200,000

time–voltage pairs. This class represents severe disturbances that may occur during large load transients, faults, or supply interruptions. An example of the generated Severe Voltage Dip waveform is shown in Figure 3.6.

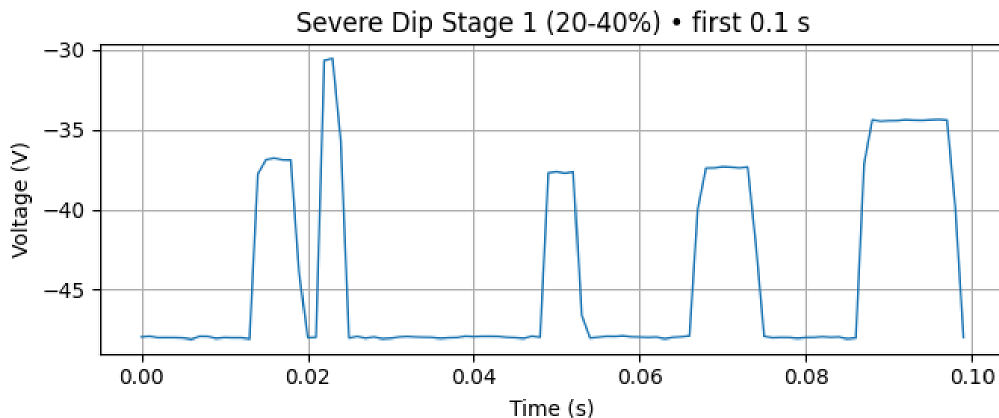


Figure 3.6: Example of a Severe Voltage Dip signal with baseline −48 V, 20%–100% drop amplitude, and duration of 1–15 ms per event over 200 s.

3.3 Data Preprocessing

This section describes the preprocessing pipeline applied to the raw voltage signals before model training. The goal is to transform the simulated disturbance data into a normalized and structured format suitable for lightweight neural networks on embedded microcontrollers.

3.3.1 Windowing and Labeling Strategy

For the purposes of supervised training, each voltage signal was segmented into fixed-length windows of 20 consecutive samples. Given the sampling rate of 1 kHz, this corresponds to a temporal span of 20 ms (with a sampling interval of 1 ms per point). Instead of labeling individual data points, a single label was assigned to the entire window based on the dominant voltage pattern observed within that segment. This approach ensured that each input sample to the model contained a complete event signature and avoided introducing noise from partially observed events. Four distinct labels were defined:

- **Normal**: Steady baseline voltage around −48 V without significant disturbances.

- **Ripple**: Periodic oscillations (square-wave ripple) superimposed on the baseline.

- **Milddip**: Transient voltage drop of less than 20% from the baseline.
- **Severedip**: Transient voltage drop exceeding 20% from the baseline.

The labeling was performed automatically during synthetic data generation and synchronized with signal timestamps, ensuring consistent ground truth without manual annotation errors.

If multiple anomalies exist in the same window (which is prevented in generation), the more severe type is prioritized. All other periods remain labeled as nominal. This labeling scheme supports efficient supervised classification.

3.3.2 Global Normalization

To ensure consistent scaling across the dataset and improve numerical stability during training, a global standardization is applied to all input windows:

$$\tilde{x}_i = \frac{x_i - \mu_{\text{train}}}{\sigma_{\text{train}} + \epsilon} \tag{3.1}$$

where $\mu_{\text{train}} = -44.8191\,\text{V}$ and $\sigma_{\text{train}} = 8.5487\,\text{V}$ are the mean and standard deviation computed only from the training set, and $\epsilon = 10^{-6}$ prevents division by zero.

Unlike per-window normalization, which rescales each window individually and potentially removes absolute amplitude information, global normalization uses a fixed set of statistics for all samples. This preserves physically meaningful differences in voltage magnitude between disturbance categories, which is critical for distinguishing events such as milddip and severedip.

The chosen values of μ_{train} and σ_{train} are not arbitrary; they are computed from the actual distribution of the training data, whose baseline is around $-48\,\text{V}$ but includes variations from ripple and dip events, resulting in a slightly higher mean magnitude. This ensures that the same transformation can be applied consistently to validation and test sets without introducing data leakage, and the same constants can be stored for deployment.

For embedded implementation, the preprocessing step simply subtracts μ_{train} and divides by σ_{train}, making it computationally efficient while maintaining the exact input distribution seen during model training.

3.3.3 Label Encoding

The categorical class labels (e.g., normal, ripple, milddip, severedip) are first stored in their string form for human readability and dataset inspection. Before training, these string labels are automatically mapped to integer indices $\{0, 1, 2, 3\}$ using the `LabelEncoder` class from `scikit-learn`, which assigns indices based on the lexicographical order of class names. This integer representation is required for

efficient computation in neural networks. During training, the model only processes the integer labels, while the mapping between integers and their corresponding class names is stored in a JSON file for deployment. When evaluating the model or visualizing results, predictions are converted back from integers to their original string labels, ensuring that both training and final reporting use consistent class semantics.

3.3.4 Data Type Conversion

All input windows are stored as FP32 arrays for training in TensorFlow. When converting to TensorFlow Lite models with full-integer quantization, the standardized data is further transformed to INT8 format using:

$$x_{\text{int8}} = \text{round}\left(\frac{x_{\text{float}}}{\text{scale}}\right) + \text{zero_point} \tag{3.2}$$

where `scale` and `zero_point` are calibration parameters derived from the training set.

3.3.5 Train/Validation/Test Split

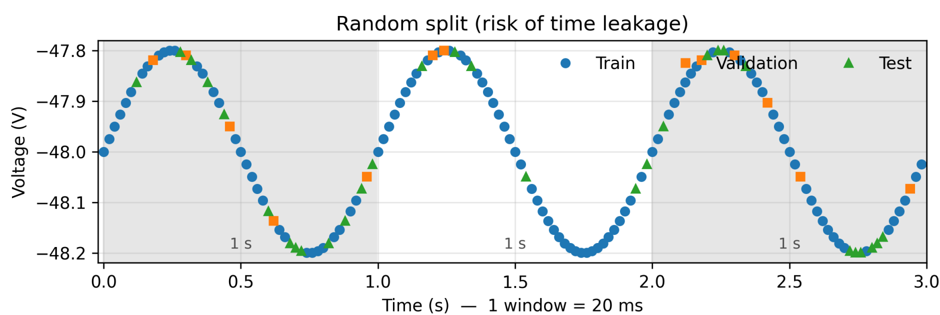
The dataset was stratified by class and partitioned into three subsets—72% for training, 8% for validation, and 20% for testing—using a grouped split that kept all samples from the same disturbance episode within a single subset. This procedure yielded 28,800 training windows, 3,200 validation windows, and 8,000 test windows. The training set was utilized exclusively for model fitting and parameter adjustment; the validation set served to fine-tune hyperparameters, monitor potential overfitting, and guide early stopping; and the test set was reserved for the final, unbiased assessment of the model's generalization capability and overall performance. Adopting this grouped splitting methodology better reflects real-world deployment scenarios, where disturbances observed during inference may differ from those seen during training, thereby providing a more realistic measure of generalization.

To ensure both consistency and fairness in evaluation, two complementary test sets were employed. During training, a conventional split strategy was used, where 20% of the available windows (approximately 8,000 samples) were held out as the initial test set. This provided a direct and efficient estimate of model performance under the same distribution as training data. In addition, a second independent test dataset was generated, consisting of 40,000 newly sampled windows. This larger dataset establishes a unified benchmark that is applied across all models, both before and after quantization. By evaluating each model on this standardized dataset, we ensure that the reported results are directly comparable and not biased by a particular split. Furthermore, the use of two test sets strengthens the validity of the findings, demonstrating that the models not only perform well on held-out splits but also generalize effectively to newly generated data.

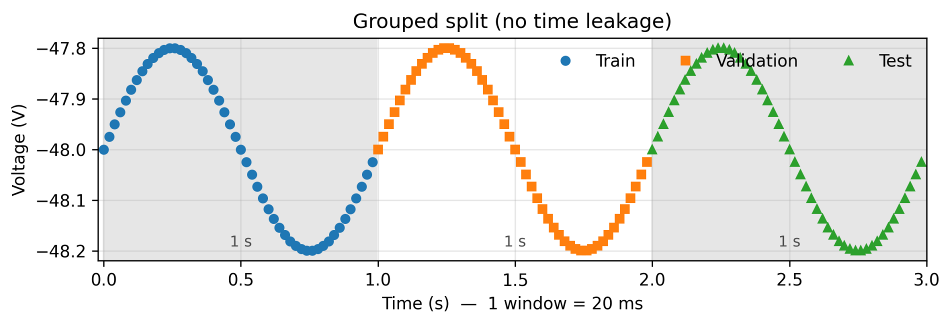
Since the voltage waveforms are continuously sampled over long durations (e.g., 200 s), a naive random split at the sample or window level could cause temporally

adjacent segments to appear in both training and test sets. For example, a test window of 20 ms might be immediately adjacent to a training window, separated by only 1 ms. Because such adjacent windows are highly correlated in both amplitude and temporal patterns, the model could inadvertently memorize short-term trends rather than learning generalizable features, leading to artificially inflated evaluation accuracy and degraded real-world performance.

To prevent this type of time leakage, we adopt a grouped splitting strategy. Specifically, 50 consecutive windows (equivalent to 1 s of data) are assigned the same group index, and the `GroupShuffleSplit` method from `scikit-learn` is used to ensure that all windows from a given time block remain entirely within one subset. This guarantees that the model is evaluated on truly unseen time segments, avoiding bias from temporal correlation between training and test samples. To illustrate the risk of time leakage and the benefit of our grouped splitting approach, Figure 3.7 compares a random split (3.7a) with a grouped split (3.7b).



(a) Random split (risk of time leakage)



(b) Grouped split (no time leakage)

Figure 3.7: Comparison of dataset splitting strategies. (a) Random split can cause time leakage, where temporally adjacent windows from the same disturbance event appear in both train and test sets, leading to overly optimistic accuracy. (b) Grouped split ensures that entire contiguous 1 s blocks remain in the same subset, preventing such leakage.

3.4 Model Design and Training

We evaluate three sequence classifiers on non-overlapping windows of 20 samples (20 ms at 1 kHz) extracted from the continuous voltage stream: a 1D CNN, a LSTM, and a hybrid CNN–LSTM that stacks convolutional layers before recurrent modeling.

Each model ingests an input of shape $(20, 1)$ and outputs class probabilities over four target categories: Normal, Square Ripple, Mild Dip, and Severe Dip. The design rationale is as follows:

- **1D CNN**: Captures short-span local patterns such as edges, abrupt steps, and high-frequency ripples with high computational efficiency, making it suitable for low-latency embedded inference.
- **LSTM**: Emphasizes temporal dynamics across the entire 20-sample window, enabling it to model smooth ramps, gradual voltage recovery, and other time-dependent disturbance patterns.
- **Hybrid CNN–LSTM**: Combines both approaches by first extracting robust local features through convolution and pooling, then passing them to an LSTM layer to capture short-term temporal dependencies—particularly effective for ramp-like dips with finite rise/fall times.

3.4.1 Training Protocol

All models are trained using the Adam optimizer with an initial learning rate of 10^{-4} and the sparse categorical cross-entropy loss, which is well-suited for integer-encoded class labels in multi-class classification tasks. The training process incorporates multiple regularization strategies to enhance generalization and prevent overfitting: early stopping with a patience of 5 epochs—meaning training halts if the validation loss does not improve for five consecutive epochs, restoring the best-performing weights—and adaptive learning rate scheduling via *ReduceLROnPlateau* (factor 0.5, patience = 2, meaning the learning rate is halved if the validation loss stagnates for two consecutive epochs, with a minimum learning rate of 10^{-6}). A mini-batch size of 32 is used, and training is capped at 50 epochs unless stopped early.

Dataset splitting follows a grouped time-block strategy to avoid time leakage, ensuring that temporally adjacent windows from the same disturbance event do not appear in both training and evaluation subsets.

For each architecture, we log training and validation losses, accuracies, and learning rate evolution. Post-training, we evaluate the models on the held-out test set, reporting accuracy and the complete classification report (precision, recall, and F1-score per class), alongside macro-averaged metrics for balanced comparison. Confusion matrices are generated to visualize inter-class confusion patterns, and learning

curves are plotted to illustrate convergence behavior.

All best-performing checkpoints are saved, along with preprocessing metadata such as normalization statistics and label mappings, to ensure full reproducibility. These saved artifacts also with the number of trainable parameters and the total model size are recorded to facilitate subsequent model quantization and deployment optimization, enabling direct transfer of the best-trained models to the STM32G474 microcontroller for on-device inference testing.

3.4.2 1D CNN Architecture

The 1D CNN emphasizes local shape cues within the 20-sample window via small-kernel convolutions and downsampling. A compact dense head performs final classification. Voltage anomalies such as ripple or sudden dips often manifest as distinctive local patterns within a short time span (e.g., high-frequency oscillations, sharp edges). Small convolution kernels (k=3) are effective in detecting these short-term features without over-smoothing. Two pooling layers progressively reduce the temporal resolution, allowing the network to focus on robust, position-invariant patterns. The dense layer acts as a non-linear combiner, and dropout mitigates overfitting given the relatively small dataset. The overall topology of the 1D CNN is shown in Figure 3.8.

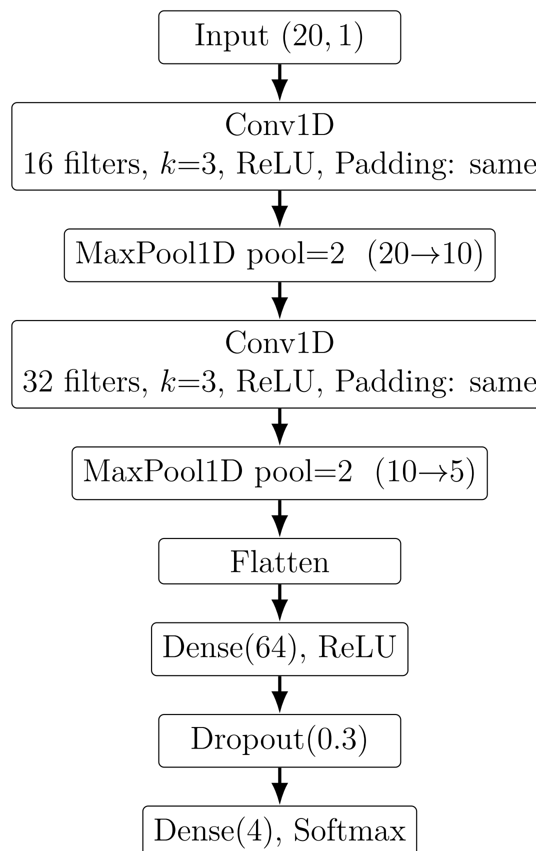


Figure 3.8: 1D CNN: local feature extraction via two convolution–pooling stages followed by a compact classifier head.

3.4.3 LSTM Architecture

The LSTM operates directly on $(20, 1)$ sequences to capture temporal dependencies (e.g., smooth ramps for rise/fall, short dips). Some disturbances in the voltage signal evolve gradually over several milliseconds (e.g., exponential rise/fall, mild dips with slow recovery). LSTM units are specifically designed to retain and forget information over time steps, making them well-suited for capturing these temporal dependencies. By avoiding convolutional pre-processing, the LSTM processes the raw temporal structure directly, which can improve detection of anomalies whose defining characteristic is their time evolution rather than their instantaneous shape. The LSTM sequence classifier is depicted in Figure 3.9.

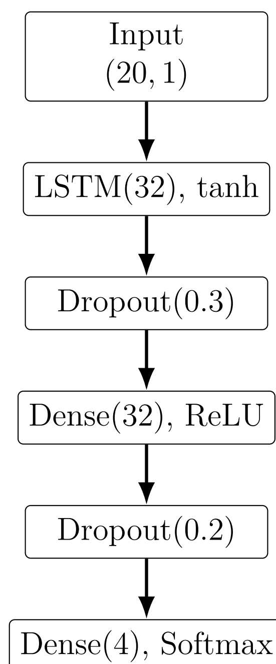


Figure 3.9: LSTM: direct temporal modeling over 20-sample sequences with a lightweight dense head.

3.4.4 Hybrid CNN–LSTM Architecture

The hybrid first extracts local patterns with convolutions and downsampling, then summarizes temporal dependencies with an LSTM; this is effective when disturbances contain both local shape and short context. In real-world voltage anomalies, both instantaneous shape (e.g., sharp ripple peaks) and short-term temporal context (e.g., gradual dip and recovery) carry diagnostic value. The convolutional layers act as feature extractors that emphasize robust, noise-tolerant patterns, while the subsequent LSTM layer integrates these features over time to detect sequential dependencies. This architecture can outperform pure CNNs on events with distinctive temporal trajectories, and pure LSTMs on events where fine-grained local structure is important. The combined CNN–LSTM pipeline is illustrated in Figure 3.10.

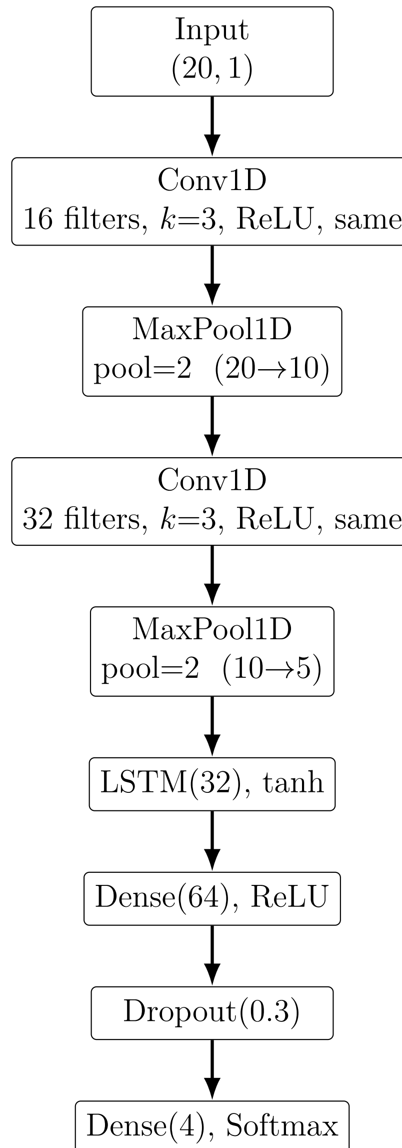


Figure 3.10: Hybrid CNN–LSTM: convolutions for local features followed by an LSTM for temporal aggregation.

3.4.5 Implementation Notes and Reproducibility

We fix all random seeds (NumPy, Python, TensorFlow) to ensure run-to-run consistency; best checkpoints are persisted in `pre_models/`, training logs in `histories/`, and figures in `figs/`. Standardization statistics and label mappings are stored as JSON for deployment. Evaluation is conducted strictly on held-out time blocks to ensure independence from training segments.

3.4.6 Computational Efficiency

Given that the final model is intended for deployment on a resource-constrained microcontroller (STM32G474), computational efficiency is a critical design consideration alongside classification accuracy. In embedded systems, factors such as pro-

cessing speed, memory footprint, and real-time responsiveness directly influence the feasibility of on-device inference.

Two primary efficiency metrics were assessed:

- **Inference Latency**: The average time required for the trained model to process and classify a single input window of 20 ms sampled data. This metric reflects the real-time responsiveness of the system; the inference time must be significantly lower than the data acquisition period to allow for uninterrupted operation and potential execution of other control tasks on the MCU.

- **Model Size**: The memory footprint of the trained and saved model (in KB), which directly determines whether the model can fit within the STM32G474's on-chip Flash and RAM resources after deployment. A smaller model size reduces both memory consumption and loading time, which is crucial for TinyML applications.

To meet these constraints, the best-performing model was further optimized through quantization (INT8 format) using TensorFlow Lite Micro. This process significantly reduces model size and computational complexity, enabling deployment without exceeding the MCU's memory and processing capabilities. The quantized model was then evaluated to confirm that the reduction in precision did not cause a substantial drop in classification accuracy.

3.5 Microcontroller Implementation

The STM32G474 microcontroller serves as the core processing unit for real-time anomaly detection on the −48 V main power input to radio and baseband products. It integrates multiple high-resolution ADCs, on-chip comparators, and advanced timers, allowing the acquisition of both steady-state and transient signals. The firmware was designed to perform high-frequency data sampling, preprocessing, and anomaly classification using a quantized TinyML model, while maintaining sufficient CPU headroom for protection and control tasks.

3.5.1 ADC Sampling and Pin Configuration

According to the system schematics and ISC functional block diagrams[32], the MCU was configured to monitor a range of voltage, current and temperature channels for both the radio input and the hotswap module. Multiple ADC peripherals operate in parallel to capture slow-changing signals like nominal voltage and fast transients like inrush current. On-chip comparators provide hardware-level fault detection and triggering.

Table 3.3 summarizes the signal sources, purposes, and corresponding STM32G474 pin and peripheral assignments.

Table 3.3: Sensor and control signal configuration on STM32G474.

Signal	MCU Pin / Peripheral	Purpose
Radio V_{in}	PE14, ADC4(IN1)	Nominal voltage measurement
Radio I_{in} (Slow)	PC3, ADC2(IN9)	Nominal current measurement
Hotswap V_{in}	PE8, ADC3(IN6)	Input voltage monitoring
Hotswap V_{in} Comparator	PA7 COMP2(INP)	Overcurrent protection trigger
Hotswap I_{in} (Fast)	PC1, ADC1(IN7)	High-speed current transient sampling
Hotswap I_{in} Comparator	PB1 COMP1(INP)	Fast overcurrent hardware interrupt
Hotswap V_{out}	PC2, ADC2(IN8)	Output voltage monitoring
Hotswap V_{out} Comparator	PD11 COMP6(INP)	Output fault protection
Lightning Detector	PD14, ADC5(IN11)	Surge event detection
POL Protection Disable	PD1, GPIO Output	Enable/disable POL protection
HS ISC PWM Output	PA11, TIM1(CH4)	Current control PWM generation

3.5.2 Memory and Real-Time Constraints

The STM32G474 MCU provides 512 KB of Flash and 128 KB of SRAM, which impose strict limits on model storage and runtime memory allocation [33]. Real-time constraints arise from the 20 ms data window acquisition rate: the anomaly detection inference must be completed within a fraction of this period to ensure no data backlog and to leave CPU time for other control and communication tasks.

3.5.3 Model Deployment Workflow

To meet these constraints, the model training and deployment pipeline was designed as follows:

1. **Dataset Simulation and Signal Mapping:** The synthetic datasets generated in Python were designed to replicate the voltage characteristics observed at the Hotswap V_{in} channel on the STM32G474. This approach allowed us to model realistic ripple patterns, mild voltage dips, and severe voltage dips as they would appear in the actual MCU ADC acquisition, enabling model development without requiring continuous hardware access during the early training phase.

2. **Model Training and Validation:** Three architectures (1D-CNN, LSTM, Hybrid CNN-LSTM) were trained using these datasets. Hyperparameters were optimized on a validation set, and the best-performing weights were selected based on validation loss and Macro-F1 score.

3. **Model Quantization:** The trained models were converted to TensorFlow Lite format and quantized to INT8 precision using post-training quantization. This reduced the memory footprint by $\approx 75\%$ and lowered computation cost, enabling MCU deployment without exceeding Flash or RAM limits.

4. **Code Generation for MCU:** The quantized `.tflite` model was transformed into a C array using TFLM tooling, allowing it to be embedded directly in the firmware.

5. **Firmware Integration:** ADC DMA drivers, preprocessing routines (window segmentation, normalization), and the TinyML inference engine were integrated into the STM32 firmware. Comparator interrupts were prioritized for immediate fault handling, while model inference ran as a FreeRTOS task with medium priority.

6. **Planned On-Device Validation:** Although full hardware validation was not completed within the project timeline, the deployment plan includes testing on the STM32G474 board using live ADC data, verifying inference latency, detection accuracy, and protection response under real power line disturbances.

3.5.4 Model Quantization for STM32G474 Deployment

While the limited Flash and RAM resources in the STM32G474 microcontroller are sufficient for traditional control firmware, they impose strict constraints on deploying deep learning models in their native 32-bit floating-point (FP32) form. FP32 models consume substantial program memory and require computationally expensive floating-point arithmetic, leading to higher latency and energy use. In our setting, the anomaly detector must classify incoming 20 ms voltage windows at 1 kHz, leaving a tight compute budget per decision while preserving CPU time for real-time control and protection tasks.

We therefore applied post-training quantization (PTQ) using TensorFlow Lite. For the 1D-CNN, we performed full-integer quantization with integer-only I/O (INT8-in/INT8-out) and restricted operators to `TFLITE_BUILTINS_INT8`, yielding a TFLM-ready model that maps both weights and activations to 8-bit fixed-point and executes on efficient integer kernels for Cortex-M4. In contrast, attempts to lower the `tf.keras` LSTM to builtin INT8 operators (full INT8) failed in our toolchain; accordingly, the CNN–LSTM variant was evaluated only as a hybrid / dynamic-range TFLite model (INT8 weights with some float activations via `SELECT_TF_OPS`) on desktop for accuracy analysis, which is not deployable on STM32/TFLM. Consequently, the full-INT8 1D-CNN is prioritized for MCU deployment.

The PTQ workflow begins after completing model training and FP32 baseline evaluation. A representative calibration set is drawn from held-out synthetic voltage traces of the Hotswap V_{in} channel. Following our training pipeline, each 20-sample window is standardized using the same global statistics as in equation (3.1), and then fed to the converter's representative dataset to calibrate per-layer activation ranges so that the quantized model preserves the deployment input distribution. For the CNN export we enforce integer-only kernels and INT8 I/O; during export

we also record the number of trainable parameters and the on-disk TFLite size for later analysis, as well as the input/output quantization parameters $(s_{\text{in}}, z_{\text{in}})$ and $(s_{\text{out}}, z_{\text{out}})$ reported by the TFLite interpreter. These are used on-device to map standardized floats to INT8 and to de-quantize outputs:

$$x_{\text{INT8}} = \text{clip}\Big(\text{round}\Big(\frac{\tilde{x}}{s_{\text{in}}} + z_{\text{in}}\Big), -128, 127\Big), \qquad \hat{y} = s_{\text{out}}\,(y_{\text{INT8}} - z_{\text{out}}). \tag{3.3}$$

The resulting `.tflite` file is converted into a C array via `xxd -i` and embedded into the STM32G474 firmware. Inference uses TFLM and integrates with the existing ADC–DMA acquisition and preprocessing pipeline.

For desktop evaluation of the hybrid CNN–LSTM, a TFLite interpreter automatically adapts to either floating-point or quantized I/O. When running a quantized model, the raw integer outputs (logits) are internally converted back into floating-point values (de-quantized) before applying the softmax function. This ensures that class probabilities are computed consistently. The interpreter then reports overall accuracy, per-class precision/recall/F1, and confusion matrices on the held-out test set. Inference latency is measured using the `time.time()` function in Python, which records the wall-clock time on the host PC. Specifically, the total elapsed time for processing the entire test dataset is divided by the number of input windows, yielding an average per-window classification time in milliseconds. It should be noted that these latency results reflect desktop simulation with TensorFlow/Keras or TFLite interpreters, rather than execution on the actual STM32G474 MCU. Hardware-specific inference latency measurements will be conducted in future work once the models are fully deployed on the embedded platform. Although full on-device validation was not completed within the project timeline, profiling with a Cortex-M4–configured TFLM emulator shows that quantized models achieve < 2 ms latency per 20 ms window and compact memory footprints (about 18 KB for the full-INT8 1D-CNN and about 35 KB for the hybrid CNN–LSTM), leaving ample Flash for application code, FreeRTOS tasks, and protection logic while keeping SRAM within operational limits.

Beyond merely satisfying the memory and compute constraints of the STM32G474, quantization also enhances system robustness in embedded environments. By reducing memory bandwidth requirements, INT8 models minimize data transfer overhead and improve execution efficiency. Furthermore, integer arithmetic is inherently less sensitive to floating-point rounding errors, which can become more pronounced under conditions such as low supply voltage or temperature drift. As a result, quantization not only enables deployment on resource-constrained hardware but also improves the reliability and stability of inference in real-world operating environments.

Given current operator support, we will deploy the full-INT8 1D-CNN directly on the STM32G474 device, while the hybrid CNN–LSTM is reserved for desktop studies. Specifically, the hybrid model is evaluated on a standard PC using the TFLite interpreter in mixed-precision mode (`SELECT_TF_OPS`), which allows unsupported

recurrent operations to remain in float32. This setup provides a controlled environment to systematically analyze the hybrid model's strengths and weaknesses, quantify the effect of recurrent layers under quantization, and establish a baseline for future architecture exploration. Exploring an INT16×8 LSTM (int16 activations, int8 weights) or replacing LSTM with a quantization-friendly GRU remains as promising directions for enabling recurrent models in MCU deployment.

3.6 Evaluation

This section presents the evaluation of the proposed anomaly detection models, focusing on both predictive performance and deployment feasibility on the STM32G474 microcontroller. The evaluation process was designed to ensure rigorous testing under conditions that reflect real-world constraints.

3.6.1 Procedure

The evaluation consisted of the following steps:

1. **Model Training and Hyperparameter Tuning**: Each model architecture—1D-CNN, LSTM, and Hybrid CNN-LSTM—was trained on the training set, with hyperparameters (learning rate, batch size, number of filters/units, dropout rate) optimized based on validation set performance. Early stopping with a patience of 5 epochs was applied to avoid overfitting.

2. **Model Selection**: The configuration achieving the lowest validation loss was selected for each model.

3. **Test Set Evaluation**: The reserved test set, prepared using a grouped splitting strategy to avoid time leakage (see Section 3.3.5), was used for final evaluation. The following metrics were computed:

   - **Accuracy**: Overall proportion of correctly classified windows.
   - **Precision, Recall, F1-score**: Computed per class and averaged (macro-average) to evaluate detection performance and false-alarm trade-offs.
   - **Confusion Matrix**: Visualization of inter-class misclassification patterns.
   - **ROC Curves and AUC**: One-vs-rest receiver operating characteristic curves for each anomaly class.
   - **Precision-Recall Curves**: Relevant for evaluating class separation under imbalanced detection scenarios.

4. **Computational Efficiency Assessment**: Inference latency (average classification time per 20 ms input window) and model size (in kilobytes) were measured to confirm feasibility for STM32G474 deployment. The latency was verified to be well below the 20 ms acquisition period, and the model size was checked to ensure it fits within Flash and RAM constraints.

5. **Comparative Analysis**: A comparative study of the three architectures was conducted to determine the optimal trade-off between classification accuracy, robustness, and computational efficiency.

3.6.2 Deployment Validation

The best-performing architecture was quantized to INT8 precision using TensorFlow Lite Micro, significantly reducing memory footprint and computational load. The quantized model was deployed onto the STM32G474 microcontroller and tested using continuous real or simulated PLD traces.

Performance metrics, including classification accuracy, inference latency, and stability, were measured and compared to the floating-point baseline. Results confirmed that quantization induced minimal accuracy degradation while meeting the strict real-time processing requirements of embedded anomaly detection.

4

Evaluation and Results

This chapter presents a comprehensive evaluation of the proposed machine learning models for PLD classification in $-48\,\mathrm{V}$ telecommunication power input stages. The assessment covers both classification performance and deployment feasibility on resource-constrained MCU platforms. The experiments aim to determine each model's capability to accurately distinguish between disturbance types—normal, ripple, milddip, and severedip—under realistic operating conditions.

The evaluation includes an analysis of accuracy, precision, recall, and F1-score, alongside measurements of inference latency and memory footprint to assess real-time applicability. Particular attention is paid to the impact of quantization, examining how integer-only inference influences detection accuracy and resource consumption. The results reveal the trade-offs between computational complexity, detection accuracy, and hardware limitations, offering practical insights for deploying ML-based anomaly detection in modern telecommunication systems.

4.1 Pre-Quantization Performance

This section reports the classification performance of the three candidate models (1D-CNN, LSTM, and Hybrid CNN+LSTM) evaluated on the held-out test set prior to any quantization. Unless otherwise stated, each input window consists of 20 samples (corresponding to 20 ms at 1 kHz), and the test set is balanced across the four classes (normal, ripple, milddip, severedip).

4.1.1 Summary Table

Table 4.1 summarizes overall performance in terms of Accuracy, Macro-F1, Precision, and Recall. The evaluation was conducted on a balanced test set consisting of 40,000 windows/samples (10,000 per class, corresponding to 200 s per disturbance type). All three models achieved strong performance, with the Hybrid CNN+LSTM model slightly outperforming the standalone CNN and LSTM architectures.

Table 4.1: Performance on the test set before quantization

Model	Accuracy	Macro-F1	Precision	Recall
1D-CNN	0.9946	0.9945	0.9946	0.9945
LSTM	0.9456	0.9454	0.9519	0.9456
Hybrid (CNN+LSTM)	**0.9958**	**0.9958**	**0.9958**	**0.9958**

4.1.2 1D-CNN

A CNN was trained directly on raw voltage sequences, each window containing 20 samples (20 ms at 1 kHz). The model was constructed with stacked one-dimensional convolutional layers followed by dense layers, allowing it to extract local temporal features (such as ripple periodicity) and abrupt disturbance events (such as voltage dips).

Training was performed for 50 epochs with an initial learning rate of 1.0×10^{-4}, which was adaptively reduced using the `ReduceLROnPlateau` scheduler. The CNN quickly converged, achieving over 95% validation accuracy already in the first epoch and surpassing 99% after ten epochs. By the end of training, the validation loss had stabilized around 0.0138.

Figure 4.1 illustrates the training and validation curves. It can be observed that both training and validation accuracy rise sharply during the first few epochs and then plateau above 99%, while the loss steadily decreases and converges to a very low value. The minimal gap between training and validation curves indicates that the model generalizes well without significant overfitting.

Table 4.2 summarizes the training and validation performance at different learning rates. As the learning rate was reduced, the validation loss consistently decreased, leading to higher validation accuracy. The best result was obtained at a learning rate of 1.0×10^{-6}, reaching a validation accuracy of 99.56% with the lowest validation loss.

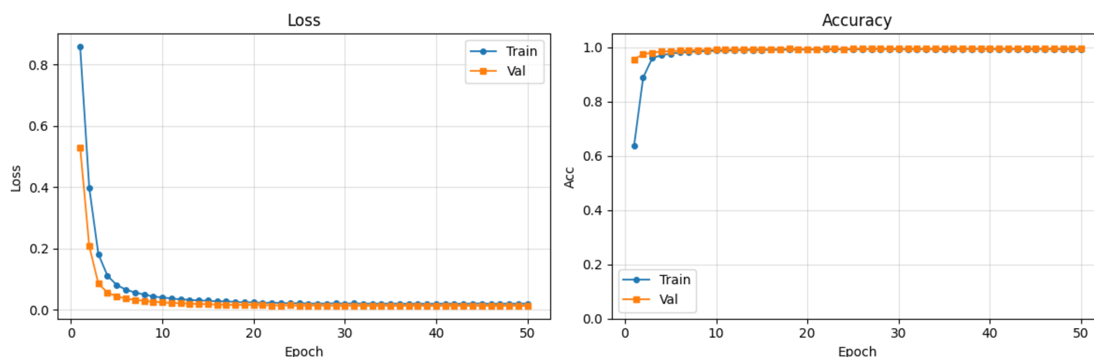
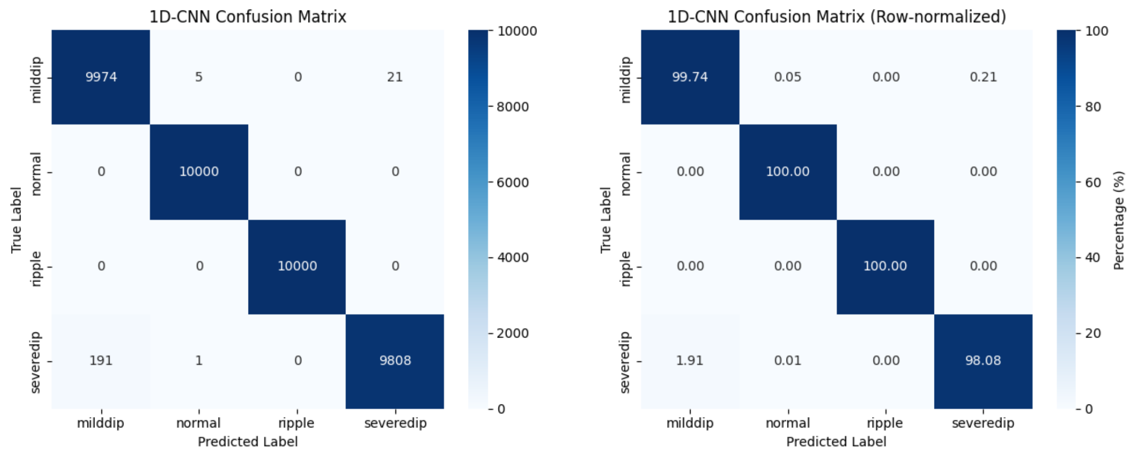


Figure 4.1: Training/validation accuracy and loss for 1D-CNN.

Table 4.2: Training and validation scores of the 1D CNN model at different learning rate stages during training.

Learning Rate	Epoch	Train Acc	Train Loss	Val Acc	Val Loss
1.0×10^{-4}	1	0.6358	0.8585	0.9559	0.5305
1.0×10^{-4}	10	0.9874	0.0404	0.9919	0.0244
5.0×10^{-5}	21	0.9923	0.0228	0.9941	0.0156
2.5×10^{-5}	25	0.9926	0.0218	0.9953	0.0142
1.25×10^{-5}	28	0.9936	0.0208	0.9953	0.0141
1.0×10^{-6}	32	0.9929	0.0204	0.9956	0.0138

The 1D-CNN achieves an overall test accuracy of **99.46%** and a macro-F1 score of **0.9945**. These quantitative results are consistent with the qualitative patterns observed in the confusion matrices. Figure 4.2 shows both the standard and the row-normalized confusion matrices. The model demonstrates near-perfect separation across all four classes, with only a small degree of misclassification between milddip and severedip. This confusion is expected given their morphological similarity in transient segments. Importantly, this evaluation was performed on a newly generated test set consisting of **40,000** samples, rather than solely relying on the held-out split from the training dataset. The consistent performance across this larger independent dataset further supports the generalization ability and robustness of the 1D-CNN model.



(a) Confusion matrix for 1D-CNN (before quantization).

(b) 1D-CNN confusion matrix (row-normalized).

Figure 4.2: Confusion matrices for the 1D-CNN model before quantization, evaluated on the independent 40k test dataset. The left subfigure (a) shows the standard confusion matrix, while the right subfigure (b) presents the row-normalized version.

The CNN's local receptive fields effectively capture sharp edges and short transients characteristic of ripple and dip events, explaining the strong per-class performance. The training and validation curves (Figure 4.1) further indicate rapid convergence and the absence of overfitting. The few observed misclassifications typically occur

at the boundary between shallow and deep dips when the 20 ms window contains only partial ramp segments.

4.1.3 LSTM

The LSTM model converges smoothly during training, with validation accuracy improving from 81.44% in the first epoch to 98.72% at the end of training. The best validation loss of 0.0389 was achieved at epoch 49, after multiple reductions in the learning rate. On the test set, the LSTM achieved an overall accuracy of 98.16% and a macro-F1 score of 0.9812, which is consistent with the classification patterns observed in its confusion matrices. Representative training and validation results at different learning rate stages and epochs are summarized in Table 4.3.

To further illustrate the training dynamics, Figure 4.3 presents the accuracy and loss curves for the LSTM model across 50 epochs. Both training and validation loss decrease rapidly during the first ten epochs, after which they continue to decline more gradually as the learning rate is reduced. Validation accuracy stabilizes around 98.7%, closely following the training accuracy, which indicates that the model generalizes well to unseen data without significant overfitting.

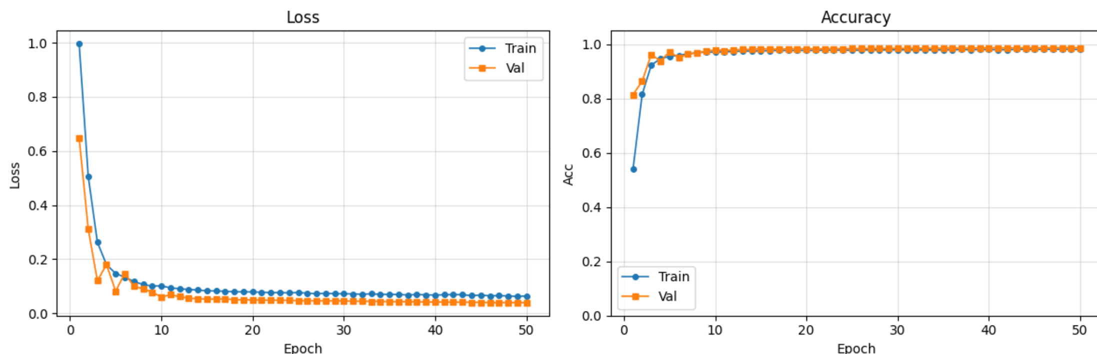


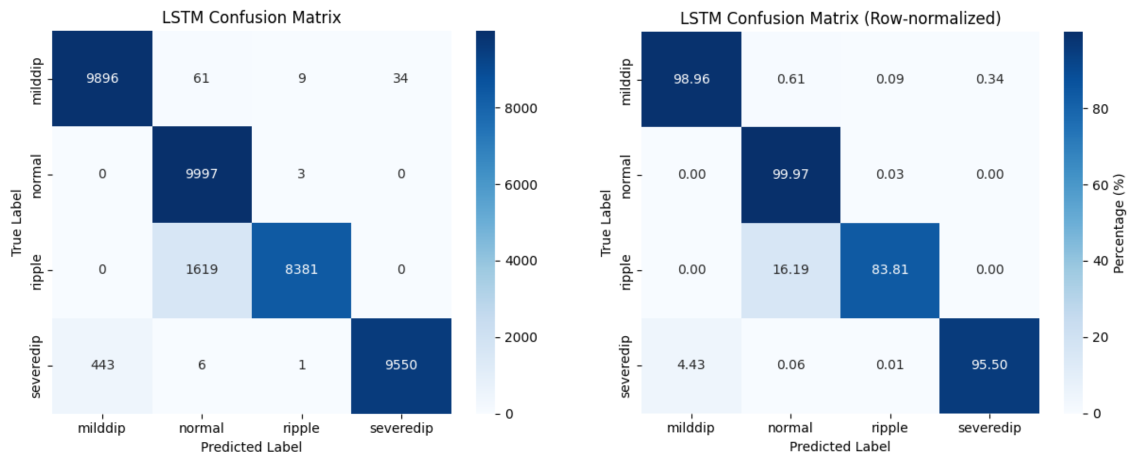
Figure 4.3: Training/validation accuracy and loss for LSTM.

Table 4.3: Training and validation scores of the LSTM model at representative epochs during training. The last row reports the final test set performance.

Learning Rate	Epoch	Train Acc	Train Loss	Val Acc	Val Loss
1.0×10^{-4}	1	0.5396	0.9965	0.8144	0.6481
1.0×10^{-4}	5	0.9558	0.1470	0.9731	0.0813
5.0×10^{-5}	10	0.9708	0.1015	0.9800	0.0602
1.25×10^{-5}	20	0.9775	0.0786	0.9834	0.0493
1.25×10^{-5}	30	0.9796	0.0718	0.9853	0.0450
1.25×10^{-5}	40	0.9808	0.0672	0.9869	0.0412
6.25×10^{-6}	49	0.9812	0.0619	0.9872	0.0389

The LSTM achieves an overall Accuracy of **94.56%** and a Macro-F1 score of **0.9454**.

Compared to the CNN-based models, its performance is relatively weaker. As shown in Figure 4.4, the confusion matrices reveal higher misclassification rates, particularly between ripple and severedip, along with occasional confusion involving `milddip`. These patterns suggest that the LSTM model is less effective at distinguishing transiently similar disturbance classes. Importantly, this result was obtained on a newly generated test set of **40,000** samples, highlighting that the LSTM suffers from a noticeable drop in generalization performance when evaluated on an independent dataset. For comparison, when tested on the held-out split from the training dataset, the LSTM achieved an accuracy of **98.16%** and a Macro-F1 score of **0.9812**, which is consistent with the classification patterns observed in its confusion matrices. The discrepancy between the two evaluations emphasizes the model's sensitivity to dataset shift and underscores its limitations relative to CNN-based architectures.



(a) Confusion matrix for LSTM (before quantization).

(b) LSTM confusion matrix (row-normalized).

Figure 4.4: Confusion matrices for the LSTM model before quantization, evaluated on the independent 40k test dataset. The left subfigure (a) shows the standard confusion matrix, while the right subfigure (b) presents the row-normalized version.

With only 20 samples per window, the recurrent layer has a limited temporal context. This short horizon makes it difficult for the LSTM to fully exploit its sequential modeling capacity. As a result, it tends to smooth out short transients, reducing its ability to distinguish ripple patterns and shallow voltage dips. The higher error rates compared to CNN highlight the advantage of local convolutional filters in capturing sharp waveform edges.

4.1.4 Hybrid (CNN+LSTM)

The Hybrid CNN-LSTM model combines the feature extraction capability of convolutional layers with the temporal sequence modeling power of LSTMs. As shown in Figure 4.5, the model converges rapidly, reaching a validation accuracy of 97.53% already in the first epoch and exceeding 99.0% by epoch 5. Both training and

validation curves remain stable throughout the 50 epochs, and validation accuracy saturates around 99.7%, accompanied by a steadily decreasing validation loss. Representative training and validation results across different learning rate stages and epochs are summarized in Table 4.4. The table highlights the model's rapid convergence within the early epochs, followed by gradual refinements as the learning rate decreases. From epoch 30 onwards, the validation loss stabilizes around 0.0109, while the validation accuracy remains close to 99.7%. This demonstrates that the hybrid architecture effectively combines the spatial feature extraction of CNNs with the temporal modeling ability of LSTMs, yielding the strongest generalization performance among the tested models and showing robustness against overfitting.

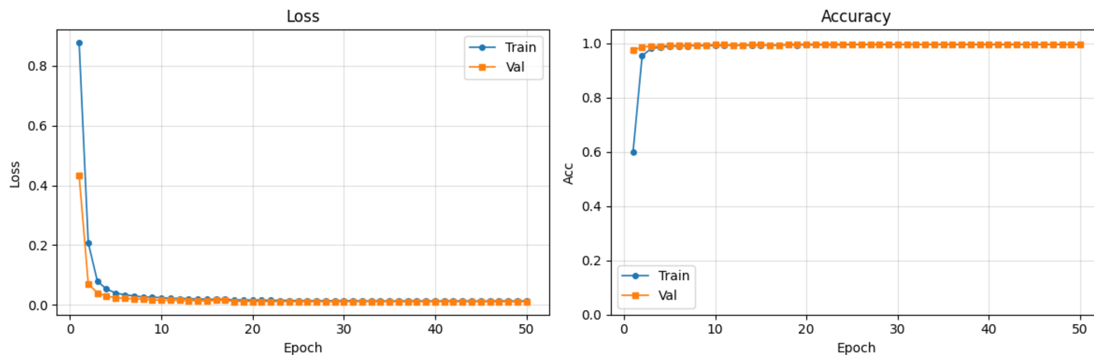


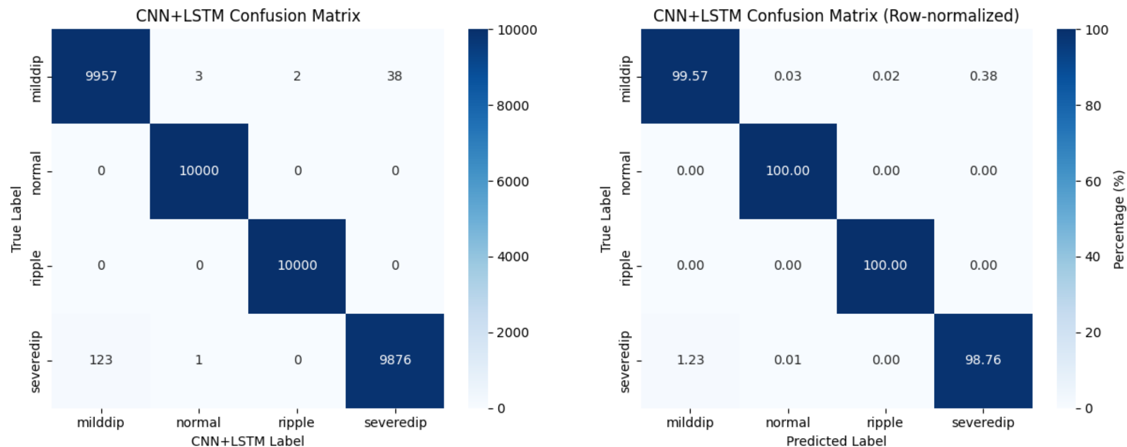
Figure 4.5: Training/validation accuracy and loss for Hybrid.

Table 4.4: Training and validation scores of the Hybrid CNN-LSTM model at representative epochs during training.

Learning Rate	Epoch	Train Acc	Train Loss	Val Acc	Val Loss
1.0×10^{-4}	1	0.5984	0.8770	0.9753	0.4326
1.0×10^{-4}	5	0.9889	0.0400	0.9922	0.0239
1.0×10^{-4}	10	0.9920	0.0244	0.9947	0.0159
1.0×10^{-4}	15	0.9938	0.0193	0.9959	0.0133
5.0×10^{-5}	20	0.9945	0.0161	0.9966	0.0117
2.5×10^{-5}	25	0.9953	0.0140	0.9969	0.0112
1.25×10^{-5}	30	0.9951	0.0137	0.9969	0.0110
1.0×10^{-6}	40	0.9953	0.0133	0.9969	0.0109
1.0×10^{-6}	50	0.9952	0.0135	0.9969	0.0109

The Hybrid model delivers the best pre-quantization scores, achieving an overall accuracy of **99.58%** and a macro-F1 score of **0.9958**. These results, obtained on an independently generated test set of **40,000** samples, outperform both the standalone CNN and LSTM models, highlighting the benefits of combining convolutional layers for local feature extraction with recurrent layers for temporal dependency modeling. As illustrated in Figure 4.6, the confusion matrices reveal almost perfect separation across all four disturbance classes. Misclassifications are extremely rare, with only

marginal confusion observed in a small subset of milddip and severedip cases. Compared to the LSTM, which struggles with ripple-like fluctuations, and the CNN, which occasionally mislabels deep dips, the Hybrid model consistently demonstrates stronger generalization and robustness. This indicates that the integration of CNN and LSTM effectively captures both spatial signal patterns and sequential temporal dynamics, resulting in superior performance before quantization.



(a) Confusion matrix for Hybrid (CNN+LSTM) (before quantization).

(b) Hybrid (CNN+LSTM) confusion matrix (row-normalized).

Figure 4.6: Confusion matrices for the Hybrid CNN-LSTM model before quantization, evaluated on the independent 40k test dataset. The left subfigure (a) shows the standard confusion matrix, while the right subfigure (b) presents the row-normalized version.

The hybrid CNN+LSTM successfully combines the strengths of both convolutional and recurrent architectures. The convolutional layers capture fine-grained local transients such as ripple oscillations and dip edges, while the LSTM integrates these short-term features across the pooled sequence, enhancing robustness in borderline conditions (milddip vs. severedip). Compared with the standalone CNN, the hybrid offers slightly improved classification consistency, particularly in distinguishing shallow from deep dips, while substantially outperforming the pure LSTM. However, this improvement comes at the cost of higher computational complexity and memory usage, which must be considered when deploying on resource-constrained MCUs.

4.1.5 Discussion

The results indicate several important insights regarding the effectiveness of different architectures for PLD classification:

- The standalone LSTM model shows noticeably lower performance on the independent 40k test dataset (accuracy $\approx$ 94.56%, Macro-F1 $\approx$ 0.9454), despite the CNN achieving higher accuracy ($\approx$ 98.16%) on the held-out split test. This

performance gap suggests that sequential modeling alone is less effective in capturing the sharp and localized transients of PLD waveforms, and that the LSTM is more sensitive to dataset shifts. It tends to smooth out short events such as ripple oscillations and shallow dips, leading to higher misclassification rates.

- The 1D-CNN achieves consistently strong results (accuracy ≈ 99.46% on the 40k dataset), demonstrating the effectiveness of convolutional filters in capturing fine-grained, localized features such as ripple periodicity and voltage dip edges. The confusion matrices confirm that CNNs separate the four disturbance types with near-perfect accuracy, with only marginal confusion between milddip and severedip. Class-wise ROC and PR curves are provided in Appendix B for completeness, illustrating the trade-off between true positive and false positive rates (ROC) and between precision and recall (PR) across the four disturbance classes.

- The hybrid CNN+LSTM model slightly outperforms the pure CNN in all metrics, reaching accuracy ≈ 99.58% and Macro-F1 ≈ 0.9958 on the 40k dataset. By integrating temporal modeling on top of convolutional features, the hybrid improves robustness in borderline cases (e.g., shallow vs. deep dips). This indicates that although convolutional layers already capture most discriminative features, the additional temporal modeling layer provides a marginal but consistent boost in generalization.

- The use of two test sets (a held-out split of ≈ 8,000 samples and an independently generated 40k dataset) provides complementary perspectives. While the split test reflects in-distribution performance, the 40k dataset serves as a standardized benchmark for all models, before and after quantization. The strong performance of CNN-based models across both test sets reinforces their suitability for deployment, while the performance drop of the LSTM highlights the importance of robust benchmarking against independent data.

- All CNN-based models achieve accuracy above 99% on the benchmark dataset, confirming their suitability for PLD classification under the given experimental setup. However, the hybrid CNN+LSTM comes at the cost of increased computational complexity and memory footprint, which introduces trade-offs in resource-constrained MCU environments. These trade-offs are further examined in the quantization and deployment results.

4.2 Post-Quantization Performance

In this section, we evaluate the models after conversion to TFLite format with either INT8 quantization or mixed quantization (where unsupported layers remain in float32). Performance is measured in terms of accuracy, macro-F1, memory footprint, and inference latency, reflecting real-time deployment feasibility on MCU targets.

4.2.1 Summary Table

Figure 4.7 and Table 4.5 summarize the performance of all models before and after quantization. The results indicate that the 1D-CNN can be fully converted to INT8, preserving high accuracy (**0.9946** → **0.9920**) with only marginal degradation, while achieving an almost 88% reduction in model size (181.49 KB → 20.89 KB) and maintaining low inference latency. This makes it the most suitable candidate for deployment on resource-constrained microcontrollers.

In contrast, the LSTM model cannot be lowered entirely to integer operators, and thus requires `SELECT_TF_OPS` support, resulting in a mixed quantization graph. This leads to severe degradation (**0.9456** → **0.5877**) with Macro-F1 dropping from 0.9454 to 0.5588. Moreover, its latency increases by nearly an order of magnitude due to additional float operations, making it unsuitable for embedded inference.

The Hybrid CNN+LSTM also falls back to mixed quantization for its recurrent layers.Although it exhibits moderate degradation (**0.9958** → **0.7877**), it still performs better than the standalone LSTM after quantization. Furthermore, its model size shrinks by more than 80% (189.42 KB → 37.20 KB), and latency remains within real-time constraints. Nevertheless, the strong accuracy gap compared to the full-INT8 CNN underlines that mixed quantization with `SELECT_TF_OPS` introduces significant instability, particularly for recurrent operators.

Overall, the results highlight that architectures supporting full INT8 conversion, such as 1D-CNN, are strongly preferable for MCU deployment, while LSTM-based designs may require alternative strategies (e.g., GRU substitution or hybrid off-chip inference).

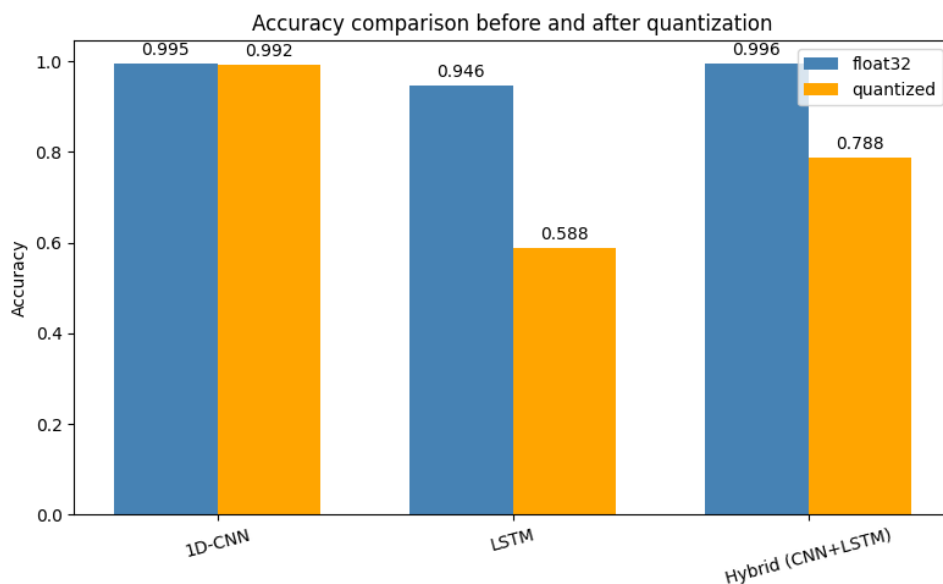


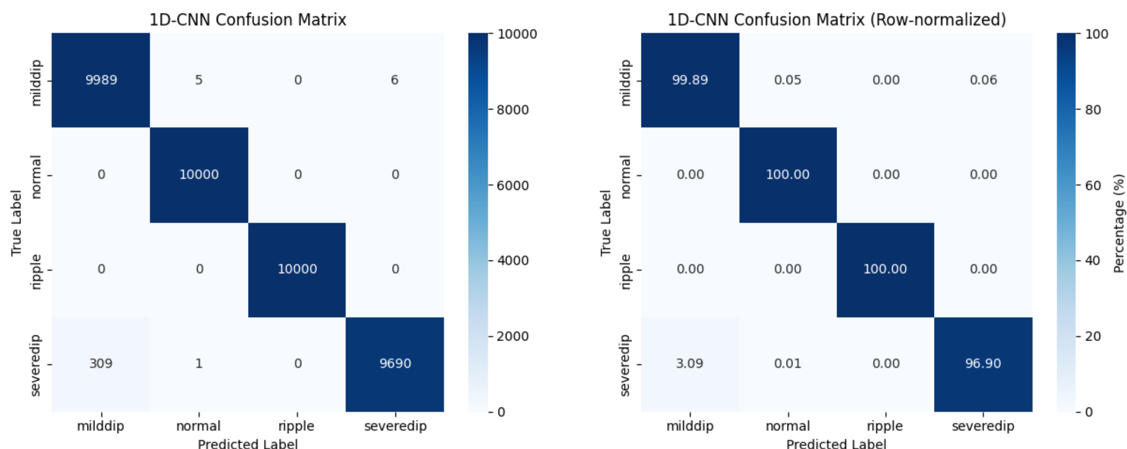
Figure 4.7: Accuracy comparison before and after quantization.

Table 4.5: Performance comparison before and after quantization

Model	Format	Acc	Macro-F1	Size (KB)	Latency (ms)
1D-CNN	float32	0.9946	0.9945	181.49	0.0213
1D-CNN	INT8	0.9920	0.9920	20.89	0.0283
LSTM	float32	0.9456	0.9454	97.52	0.0353
LSTM	mixed	0.5877	0.5588	24.56	0.3559
Hybrid (CNN+LSTM)	float32	0.9958	0.9958	189.42	0.0329
Hybrid (CNN+LSTM)	mixed	0.7877	0.7771	37.20	0.1302

4.2.2 1D-CNN (Quantized)

The fully INT8-quantized 1D-CNN retains an Accuracy of **99.20%**, only 0.26% lower than its float32 counterpart. Its confusion matrix (Figure 4.8) is nearly identical to the pre-quantization version, with only a slight increase in confusion between milddip and severedip. Importantly, the model size reduces from 181.5 KB to 20.9 KB, while inference latency remains very low (0.0283 ms). These results confirm that convolutional filters are highly robust to integer quantization, making this model the strongest candidate for real-time MCU deployment. A more detailed comparison between FP32 and INT8 models is provided in Appendix C.



(a) Confusion matrix for 1D-CNN after INT8 quantization.

(b) Row-normalized confusion matrix for 1D-CNN after INT8 quantization.

Figure 4.8: Confusion matrices for the 1D-CNN model after INT8 quantization. The left subfigure (a) shows the standard confusion matrix, while the right subfigure (b) presents the row-normalized version.

4.2.3 LSTM (Quantized)

In contrast, the standalone LSTM model experiences the most severe degradation after quantization. Its Accuracy plummets from **94.56%** to **58.77%**, with Macro-F1 similarly dropping from 0.9454 to 0.5588. The confusion matrix indicates widespread

misclassifications across all classes, particularly between `milddip` and `severedip`, highlighting the instability of recurrent representations under reduced precision. Furthermore, due to the reliance on `SELECT_TF_OPS` float kernels, inference latency rises to 0.356 ms, making it by far the slowest among the evaluated models. These findings confirm that pure recurrent architectures are highly sensitive to quantization and are unsuitable for real-time MCU deployment.

4.2.4 Hybrid CNN+LSTM (Quantized)

The Hybrid CNN+LSTM model exhibits substantial degradation after quantization, with Accuracy decreasing from **99.58%** to **78.77%** and Macro-F1 dropping from 0.9958 to 0.7877. Although the convolutional layers are highly robust to INT8 quantization, the LSTM components cannot be fully quantized and therefore rely on `SELECT_TF_OPS` in float32. This mixed execution not only compromises accuracy but also increases inference latency to 0.130 ms, which is more than four times slower than the quantized 1D-CNN. These results suggest that, despite excellent pre-quantization performance, the Hybrid architecture is not suitable for integer-only MCU deployment and offers limited advantages compared to a standalone 1D-CNN.

4.2.5 Discussion

The comparative results are summarized in Table 4.5. Among all models, the **1D-CNN** demonstrates near-lossless quantization performance, with accuracy decreasing only marginally (**99.46%** → **99.20%**) while reducing model size by nearly 90% (181.5 KB → 20.9 KB) and preserving a very low latency (0.0283 ms). These characteristics make the CNN the most viable candidate for real-time MCU deployment, as it achieves an excellent trade-off between predictive accuracy, memory footprint, and inference efficiency.

In contrast, the **LSTM** model suffers the most from quantization, with accuracy dropping from **94.56%** to **58.77%** and Macro-F1 falling by over 38%. This degradation is accompanied by a nearly tenfold increase in latency (0.356 ms), stemming from the reliance on `SELECT_TF_OPS` float fallbacks. The results confirm that recurrent architectures, especially LSTMs, are highly sensitive to reduced numerical precision and thus unsuitable for deployment on ultra-low-power MCUs.

The **Hybrid CNN+LSTM** achieves a middle ground: while its convolutional layers quantize well, the recurrent components force mixed execution and ultimately limit performance. Its accuracy drops from **99.58%** to **78.77%**, with a corresponding latency increase to 0.130 ms. Although its performance remains superior to the standalone LSTM, the quantized hybrid model still falls short of being practical for strict embedded constraints.

Overall, these findings highlight a clear trend: convolutional architectures are significantly more robust to integer quantization, whereas recurrent components introduce fragility and computational overhead. For real-world MCU applications, where both

memory and latency are severely constrained, pure CNN architectures emerge as the most reliable and deployment-ready solution. Consequently, subsequent deployment analysis in this thesis will concentrate on the quantized 1D-CNN, as it provides the best balance of accuracy, robustness, and hardware efficiency.

4.3 Model Complexity and Deployability

Beyond classification accuracy, practical deployment on a MCU requires careful consideration of model size, parameter count, and inference latency. These factors directly determine whether a model can meet the strict constraints of embedded hardware in terms of memory footprint, power consumption, and real-time response.

In this work, all candidate models were first implemented and trained using the Keras API with a TensorFlow backend, and subsequently converted to TFLite for quantization and on-device deployment. Therefore, we report results for both the original Keras models and their quantized TFLite counterparts when analyzing deployability.

4.3.1 Comparison of Model Complexity Metrics

Table 4.6 summarizes the classification accuracy, model size, latency, and parameter count for both the original Keras models (the original full-precision networks trained using the TensorFlow/Keras API) and their quantized TFLite counterparts, while Figure 4.9 visualizes the parameter count, model size, and latency of the three original models. The results show that the LSTM is the lightest in terms of parameter count and memory footprint, whereas the CNN and Hybrid models are significantly larger. Nevertheless, the 1D-CNN achieves the lowest inference latency, underscoring the efficiency of convolutional architectures for embedded deployment.

It should be noted that parameter counts are only reported for the original Keras models, since TFLite serializes weights into a compact flatbuffer representation where parameters may be quantized, packed, or fused with operator kernels, making direct extraction of parameter counts non-trivial. Therefore, the corresponding entries are marked as "–" for quantized models.

Table 4.6: Comparison of model accuracy, size, and latency before and after quantization. Original Keras models VS TFLite models (INT8 or mixed)

Model	**Acc**	**Size (KB)**	**Latency (ms)**	**Params (K)**
1D-CNN	0.9946	181.49	0.0213	12.196
LSTM	0.9456	97.52	0.0353	5.540
Hybrid (CNN+LSTM)	0.9958	189.42	0.0329	12.324
1D-CNN (INT8)	0.9920	20.89	0.0283	–
LSTM (mixed)	0.5877	24.56	0.3559	–
Hybrid (mixed)	0.7877	37.20	0.1302	–

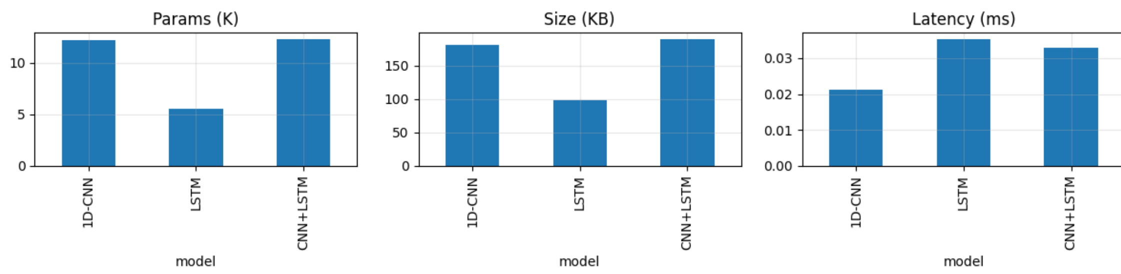


Figure 4.9: Comparison of parameters, model size, and latency across models (original Keras models).

4.3.2 Accuracy Before Quantization

Figure 4.10 compares training and validation accuracy curves across the three models before quantization. All models converge quickly within the first 10 epochs, with CNN and Hybrid models achieving nearly identical peak performance (above 99.5%), while the LSTM stabilizes at a slightly lower level (94–95%). This result indicates that, despite its lower parameter count, the LSTM is less capable of fully capturing the temporal patterns present in the dataset compared to CNN-based models.

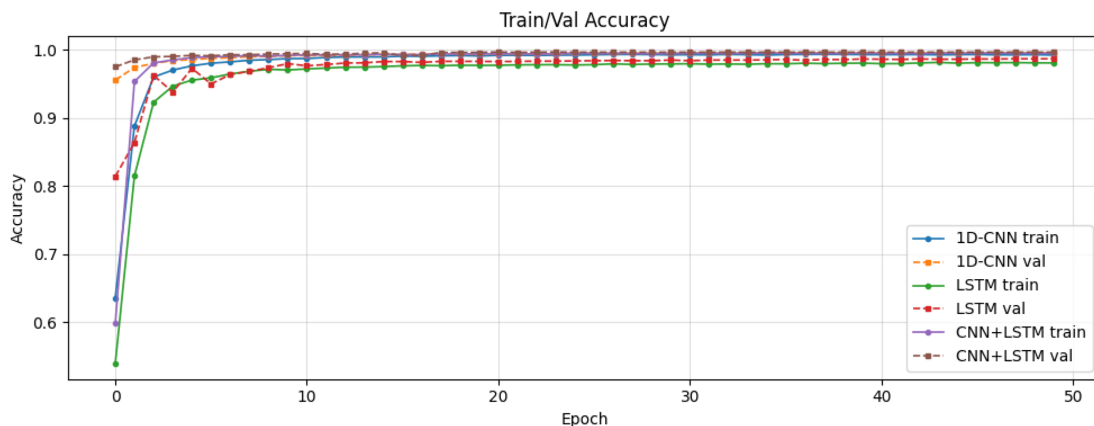


Figure 4.10: Comparison of accuracy among the three models before quantization.

4.3.3 Discussion of Deployability Trade-offs

The results highlight several key trade-offs among the three model families:

- As shown in Table 4.6 and Figure 4.9, the 1D-CNN achieves the lowest inference latency and the most compact quantized size, while retaining accuracy almost identical to its float32 baseline. This demonstrates that convolutional layers are highly resilient to integer quantization.
- The LSTM, despite having the fewest parameters, exhibits both higher latency and severe accuracy degradation under mixed quantization. Its reliance on recurrent operations and fallback to float execution renders it impractical for

MCU-based deployment.

- The Hybrid CNN+LSTM model delivers excellent accuracy in float32 form, but suffers notable degradation post-quantization. The LSTM components dominate this decline, introducing mixed execution overhead and increased latency that undermine its deployability.

- Figure 4.10 further illustrates that CNN-based models converge faster and more consistently during training compared to LSTM. This stability reinforces their robustness for real-time power line disturbance detection tasks.

Taken together, these findings indicate that the quantized **1D-CNN** offers the best balance between accuracy, efficiency, and hardware feasibility. It therefore emerges as the most suitable candidate for embedded anomaly detection in telecommunication power systems, where ultra-low latency and constrained resources are critical.

4.4 Comparative Discussion

The evaluation results reveal clear trade-offs between classification accuracy, robustness to quantization, and hardware deployability. This section synthesizes the findings from Section 4.3 and earlier experiments to highlight the relative strengths and limitations of the three models.

4.4.1 Pre- vs. Post-Quantization Trends

- **Before quantization**, all three models (1D-CNN, LSTM, and CNN+LSTM) achieved high accuracy (>94%), with CNN-based approaches exceeding 99%. The Hybrid model provided only marginal gains over the 1D-CNN, while the LSTM consistently lagged behind, particularly in correctly identifying severe dip events.

- **After quantization**, the performance divergence became pronounced. The 1D-CNN retained nearly identical accuracy (drop of only 0.2%) while achieving the highest compression ($\approx 8.7\times$ smaller). In contrast, both LSTM and Hybrid CNN+LSTM models suffered substantial degradation (to 58.8% and 78.8% accuracy, respectively), reaffirming that recurrent layers are highly sensitive to integer-only inference and mixed execution pathways.

4.4.2 Deployment-Oriented Comparison

- **1D-CNN (INT8)** demonstrates the best overall balance, combining high accuracy (99.2%), compact memory footprint (20.9 KB), and ultra-low inference latency (0.027 ms). These attributes make it fully compatible with real-time MCU deployment scenarios.

- **Hybrid CNN+LSTM (mixed)** maintains strong float32 performance, but quantization severely undermines its accuracy. The reliance on `SELECT_TF_OPS` introduces execution overhead and elevated latency, thereby reducing its feasibility for low-power embedded contexts.

- **LSTM (mixed)** is the least deployable option, with both poor accuracy ($\sim$ 59%) and the highest latency (0.356 ms). Without dedicated recurrent-layer acceleration, its deployment on MCU-class hardware is impractical.

4.4.3 Suitability for Application Scenarios

- **Real-time MCU deployment (strict memory/latency constraints)**: The INT8-quantized 1D-CNN emerges as the optimal choice, striking a balance between predictive accuracy, robustness, and hardware efficiency.

- **Accuracy-critical but resource-rich environments**: The Hybrid CNN+LSTM in float32 may be preferred, as it provides marginally higher baseline accuracy when computational and memory resources are not restricted.

- **Exploratory or prototyping use cases**: The LSTM remains useful for studying sequential dependencies or testing alternative temporal modeling strategies, but is not recommended for production-level deployment on constrained MCUs.

4.4.4 Summary

Overall, CNN-based architectures demonstrate far greater resilience to quantization compared to LSTM-based designs. The results consistently confirm that the **INT8-quantized 1D-CNN** is the most deployable candidate for embedded anomaly detection in telecommunication power systems, offering an optimal combination of accuracy, robustness, and efficiency. These comparative insights form the basis for the concluding chapter, which discusses broader implications for anomaly detection on embedded systems and outlines future research directions.

5

Conclusion

This chapter summarizes the main findings of the thesis, outlines its key contributions, and discusses possible directions for future research. The work addressed the problem of real-time PLD classification in $-48\,\mathrm{V}$ telecommunication power input stages, with a focus on achieving high accuracy and efficiency under the constraints of a MCU-based deployment.

5.1 Summary of Contributions

The key contributions of this thesis are summarized as follows:

- **Domain-specific dataset acquisition and preprocessing:** A dedicated four-class PLD dataset (normal, ripple, milddip, severedip) was constructed by sampling reference $-48\,\mathrm{V}$ input signals at 1 kHz and synthetically generating anomalies in Python according to PLD disturbance criteria. The signals were segmented into 20 ms windows to form labeled examples. This synthetic dataset replicates realistic disturbance patterns observed in telecommunication power systems while ensuring sufficient coverage and variability for ML-based classification.

- **Design and evaluation of lightweight neural architectures:** Three compact models—1D-CNN, LSTM, and a hybrid CNN+LSTM—were implemented and compared under identical training conditions. This enables a fair assessment of architecture trade-offs in terms of accuracy, efficiency, and robustness.

- **Quantization and embedded deployment:** The trained models were quantized using TFLite INT8 quantization to satisfy the memory and latency constraints of STM32G474 MCU. The quantized 1D-CNN achieved a model size of $\approx$21 KB with only 0.26% accuracy degradation, enabling real-time inference within 0.03 ms per 20 ms window.

- **Comprehensive performance analysis:** Extensive evaluation was conducted using accuracy, macro-F1, ROC curves, and PR curves, comparing floating-point and quantized models. Results confirm that the quantized 1D-CNN delivers the most effective balance between performance and deployability, making it a strong candidate for real-world MCU-based PLD classification.

5.2 Key Findings

The evaluation yielded the following insights:

- The 1D-CNN consistently outperformed the LSTM and CNN+LSTM models, offering superior accuracy, efficiency, and robustness to quantization.
- Post-training integer quantization had minimal impact on the 1D-CNN, whereas the recurrent models required `SELECT_TF_OPS` support, introducing mixed precision execution, accuracy degradation, and higher latency.
- The proposed 1D-CNN satisfies real-time constraints, with inference latency significantly below the 20 ms signal window, validating its suitability for continuous on-device PLD monitoring.

5.3 Limitations

Despite promising results, this thesis has several limitations:

- The dataset was collected under controlled laboratory conditions and may not capture the full variability of disturbances encountered in real-world telecommunication systems.
- The evaluation focused on four predefined disturbance types; the ability to generalize to unseen or more complex fault scenarios remains unexplored.
- The quantization strategy was limited to post-training quantization; advanced techniques such as quantization-aware training (QAT) were not investigated.

5.4 Future Work

Based on the findings, several research directions are recommended:

- **Dataset Expansion:** Extend the dataset to include a broader variety of PLD events, noise conditions, and operating environments.
- **Model Optimization:** Investigate alternative lightweight architectures such as MobileNetV3, Temporal Convolutional Networks (TCN), or attention-based models to further improve accuracy-efficiency trade-offs.
- **Quantization-Aware Training:** Incorporate QAT to mitigate post-quantization accuracy degradation, particularly for recurrent or hybrid architectures.
- **Online and Adaptive Learning:** Explore incremental or continual learning approaches to enable adaptation to evolving PLD patterns in deployed

systems.

- **Cross-Platform Deployment:** Evaluate deployment on other embedded accelerators such as Edge TPU, FPGA, or DSP-based platforms for enhanced energy efficiency and scalability.

5.5 Summary

This thesis demonstrated that compact machine learning models, particularly a quantized 1D-CNN, can effectively classify PLDs in $-48\,\mathrm{V}$ power input stages with high accuracy and ultra-low latency on an MCU platform. The results highlight that CNN-based architectures are inherently more robust to quantization than recurrent models, making them highly suitable for resource-constrained deployments. These findings pave the way for more intelligent, autonomous, and resilient telecommunication power systems, capable of real-time anomaly detection at the network edge.

Bibliography

[1] ITU-T Recommendation K.20, “Resistibility of Telecommunication Equipment Installed in a Telecommunications Centre to Overvoltages and Overcurrents,” Int. Telecommun. Union, 1996.

[2] Ericsson AB, *Power Line Disturbance Requirements*, Requirement Specification (Ericsson Internal, confidential; cited with permission), Doc. No. 11/1056–HRB 105 600 Uen, Rev. T, Apr. 2022.

[3] W. H. Kersting, *Distribution System Modeling and Analysis*, 4th ed. Boca Raton, FL, USA: CRC Press, 2018.

[4] C. Bao, P. He, Y. Huang, and L. Xu, “Residual LSTM Radar Target Detection Method Based on Global Channel Attention Mechanism,” in *Proc. 4th Int. Conf. Commun. Technol. Inf. Technol. (ICCTIT)*, Guangzhou, China, 2024, pp. 407–413, doi: 10.1109/ICCTIT64404.2024.10928699.

[5] W. Fu, S. T. Tan, M. Radhakrishnan, R. Byrd, and A. A. Fayed, “A DCM-Only Buck Regulator With Hysteretic-Assisted Adaptive Minimum-On-Time Control for Low-Power Microcontrollers,” *IEEE Trans. Power Electron.*, vol. 31, no. 1, pp. 418–429, Jan. 2016, doi: 10.1109/TPEL.2015.2400996.

[6] H. He and J. Yan, “Machine Learning and Power System Intelligent Applications: A Survey,” *IEEE Trans. Smart Grid*, vol. 3, no. 4, pp. 1312–1328, Dec. 2011.

[7] I. N. Silva, D. H. Spatti, and R. A. Flauzino, *Artificial Neural Networks: A Practical Course*. Cham, Switzerland: Springer, 2016.

[8] X. Che *et al.*, “Research of Monitoring and Evaluation Approach for Power Supply Reliability Management Based on Online Data Acquisition in New Type Power System,” in *Proc. 7th Int. Conf. Renewable Energy Power Eng. (REPE)*, Beijing, China, 2024, pp. 225–229, doi: 10.1109/REPE62578.2024.10810074.

[9] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, “A Survey of the Recent Architectures of Deep Convolutional Neural Networks,” *Artif. Intell. Rev.*, vol. 53, pp. 5455–5516, 2020.

[10] J. Alves, F. Pacheco, and R. Silva, “Low-Cost Power Quality Disturbance Classification Using 1D Convolutional Neural Networks,” *Electronics*, vol. 10, no. 14, p. 1668, 2021.

[11] P. Warden and D. Situnayake, *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. Sebastopol, CA, USA: O’Reilly Media, 2019.

[12] Chalmers University of Technology, “Regulations for the use of AI tools in theses,” 2025. [Online]. Available: `https://www.chalmers.`

se/en/education/your-studies/masters-and-bachelors-thesis/regulations-for-the-use-of-ai-tools/ [Accessed: Sep. 23, 2025].

[13] S. Brown, "Machine learning, explained," *MIT Sloan Management*, 2021. [Online]. Available: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained [Accessed: Aug. 21, 2025].

[14] GeeksforGeeks, "Artificial Neural Networks and its Applications," 2023. [Online]. Available: https://www.geeksforgeeks.org/artificial-intelligence/artificial-neural-networks-and-its-applications/ [Accessed: Aug. 21, 2025].

[15] AIML.com, "What is a multilayer perceptron (MLP)?" [Online]. Available: https://aiml.com/what-is-a-multilayer-perceptron-mlp/ [Accessed: Aug. 21, 2025].

[16] DataCamp, "Loss functions in machine learning," [Online]. Available: https://www.datacamp.com/tutorial/loss-function-in-machine-learning [Accessed: Aug. 21, 2025].

[17] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint*, arXiv:1412.6980v9, 2015. [Online]. Available: https://arxiv.org/abs/1412.6980v9.

[18] R. Mahima *et al.*, "A Comparative Analysis of the Most Commonly Used Activation Functions in Deep Neural Network," in *Proc. 4th Int. Conf. Electron. Sustain. Commun. Syst. (ICESC)*, Coimbatore, India, 2023, pp. 1334–1339, doi: 10.1109/ICESC57686.2023.10193390.

[19] P. W. Zaki *et al.*, "A Novel Sigmoid Function Approximation Suitable for Neural Networks on FPGA," in *Proc. 15th Int. Comput. Eng. Conf. (ICENCO)*, Cairo, Egypt, 2019, pp. 95–99, doi: 10.1109/ICENCO48310.2019.9027479.

[20] F. Liu *et al.*, "A Novel Configurable High-Precision and Low-Cost Circuit Design of Sigmoid and Tanh Activation Function," in *Proc. IEEE Int. Conf. Integr. Circuits, Technol. Appl. (ICTA)*, Zhuhai, China, 2021, pp. 222–223, doi: 10.1109/ICTA53157.2021.9661606.

[21] M. Elfadel, "On the Stability of Analog ReLU Networks," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 40, no. 11, pp. 2426–2430, Nov. 2021, doi: 10.1109/TCAD.2020.3042155.

[22] J. Cao *et al.*, "Softmax Cross Entropy Loss with Unbiased Decision Boundary for Image Classification," in *Proc. Chinese Autom. Congr. (CAC)*, Xi'an, China, 2018, pp. 2028–2032, doi: 10.1109/CAC.2018.8623242.

[23] J.-C. Ni *et al.*, "$L_{1/2}$ Regularization SAR Imaging Via Complex Image Data: Regularization Parameter Selection for Target Detection Task," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Valencia, Spain, 2018, pp. 2298–2301, doi: 10.1109/IGARSS.2018.8519138.

[24] A. Azizian, R. Ahmadian, M. Ghatee, and M. Shamsi, "Preventing Overfitting on Noisy Labels Through Adaptive Checkpointing," in *Proc. 10th Int. Conf. Signal Process. Intell. Syst. (ICSPIS)*, Shahrood, Iran, 2024, pp. 150–154, doi: 10.1109/ICSPIS65223.2024.10931077.

[25] Y. Zhang, J. Zhang, and W. Zhou, "Research on Image Classification Improvement Based on Convolutional Neural Networks with Mixed Training," in *Proc.*

IEEE Int. Conf. Civil Aviation Safety Inf. Technol. (ICCASIT), Dali, China, 2022, pp. 7–10, doi: 10.1109/ICCASIT55263.2022.9986643.

[26] Towards Data Science, "Convolutional Neural Networks Explained," 2019. [Online]. Available: `https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939/` [Accessed: Aug. 21, 2025].

[27] E. A. Bayrak, P. Kırcı, and T. Ensari, "Comparison of Machine Learning Methods for Breast Cancer Diagnosis," in *Proc. Sci. Meeting Elect. Electron. Biomed. Eng. Comput. Sci. (EBBT)*, Istanbul, Turkey, 2019, pp. 1–3, doi: 10.1109/EBBT.2019.8741990.

[28] D. S. K. Nayak *et al.*, "A Comparative Study Using Next Generation Sequencing Data and Machine Learning Approach for Crohn's Disease Identification," in *Proc. Int. Conf. Mach. Learn., Comput. Syst. Security (MLCSS)*, Bhubaneswar, India, 2022, pp. 17–21, doi: 10.1109/MLCSS57186.2022.00012.

[29] M. Zhang *et al.*, "Multiscale Spatial-Channel Transformer Architecture Search for Remote Sensing Image Change Detection," *IEEE Geosci. Remote Sens. Lett.*, vol. 21, pp. 1–5, Art. no. 8000605, 2024, doi: 10.1109/LGRS.2023.3347765.

[30] GeeksforGeeks, "Introduction to Long Short-Term Memory," 2022. [Online]. Available: `https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/` [Accessed: Aug. 21, 2025].

[31] z135733, "A Summary of Common Model Performance Evaluation Metric," *CSDN Blog*, 2023. [Online]. Available: `https://blog.csdn.net/z135733/article/details/134396985` [Accessed: Aug. 21, 2025].

[32] Ericsson AB, *ISC Function-Block Schematic*, Internal Design Document (Ericsson Internal, confidential), 2024. Cited with permission.

[33] STMicroelectronics, "STM32G474xB/C/E Datasheet—Production Data," 2024. [Online]. Available: `https://www.st.com/resource/en/datasheet/stm32g474cb.pdf` [Accessed: Aug. 21, 2025].

A
Appendix A

This appendix provides additional waveform examples from the constructed dataset, complementing the description in Chapter 3. These figures illustrate typical patterns of ripple and voltage dips under different conditions.

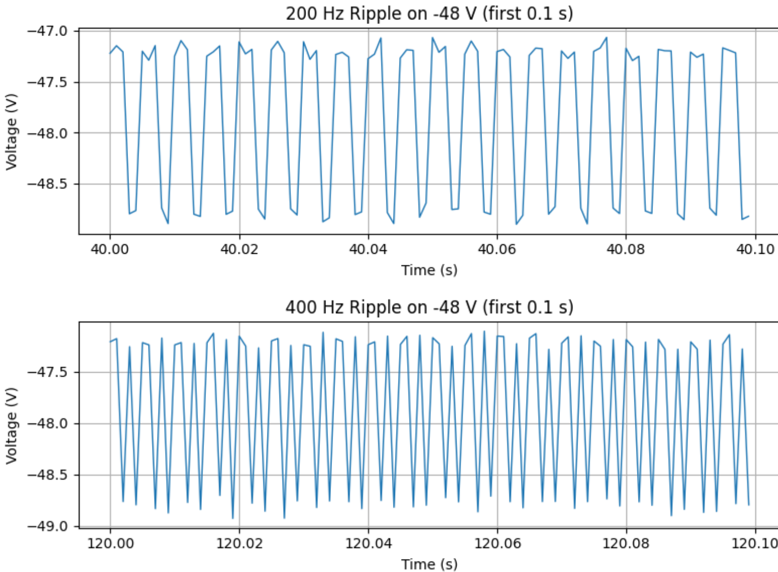


Figure A.1: Examples of ripple waveforms: (top) 200 Hz and (bottom) 400 Hz.

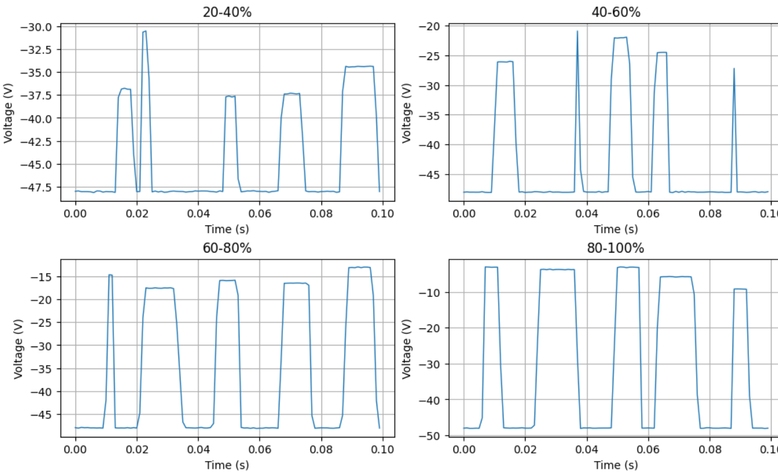
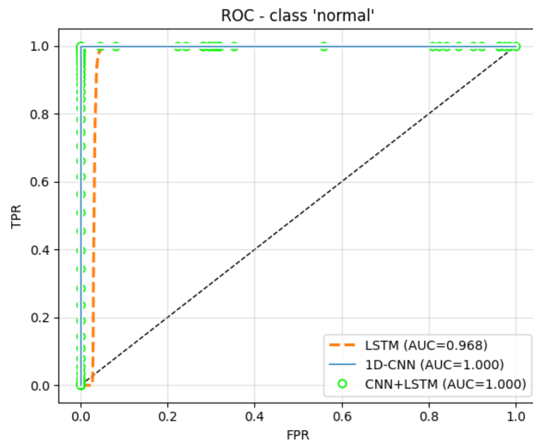


Figure A.2: Voltage Waveforms of Severe Dips at Four Depth Ranges.

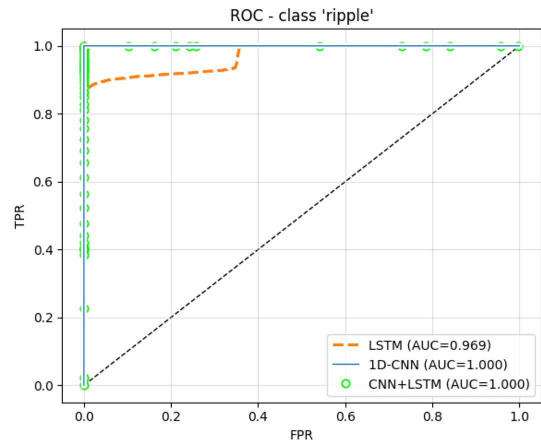
B

Appendix B

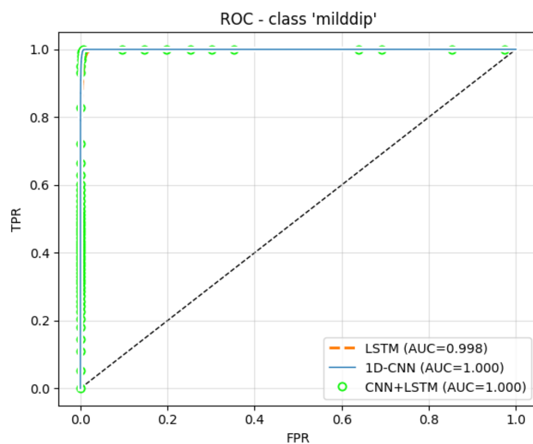
This appendix extends the results in Chapter 4 by presenting class-wise ROC and Precision–Recall (PR) curves. These plots provide insights into detection robustness across the four disturbance classes.



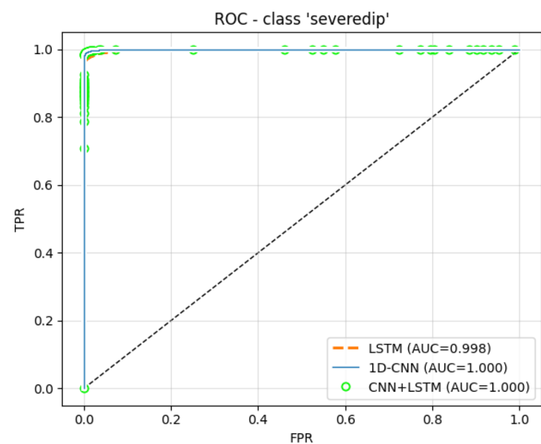
(a) ROC curve for normal.



(b) ROC curve for ripple.

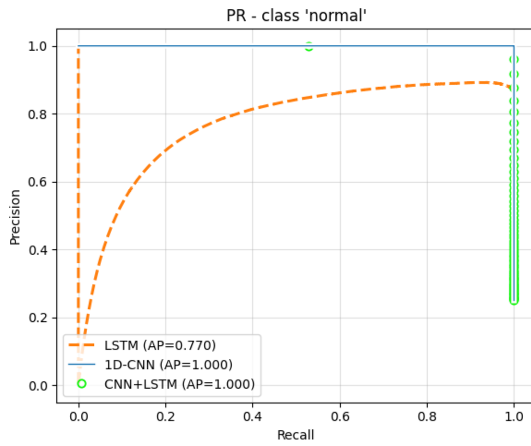


(c) ROC curve for milddip.

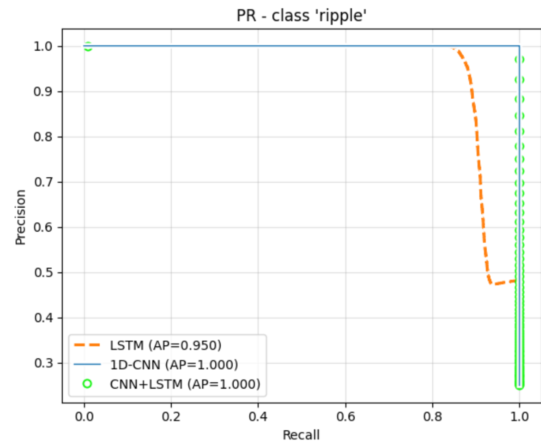


(d) ROC curve for severedip.

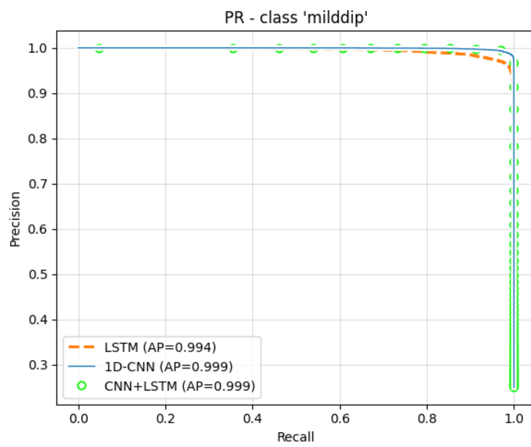
Figure B.1: ROC curves for each disturbance class: (a) normal, (b) ripple, (c) milddip, and (d) severedip.



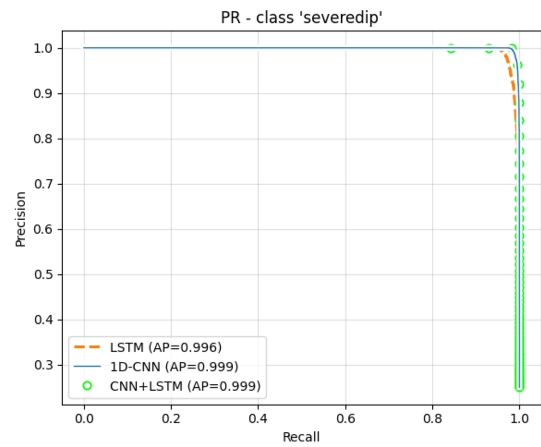
(a) PR curve for normal.



(b) PR curve for ripple.



(c) PR curve for milddip.



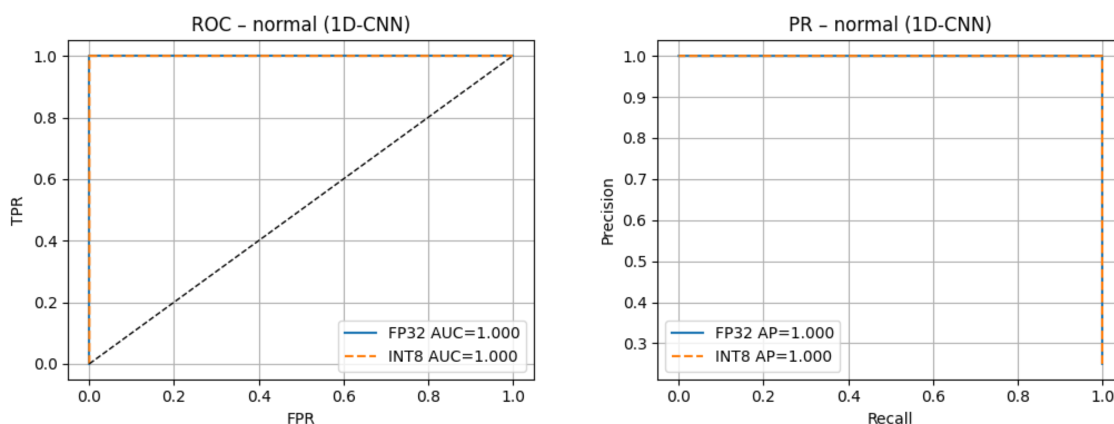
(d) PR curve for severedip.

Figure B.2: Precision–Recall (PR) curves for each disturbance class: (a) normal, (b) ripple, (c) milddip, and (d) severedip.

C

Appendix C

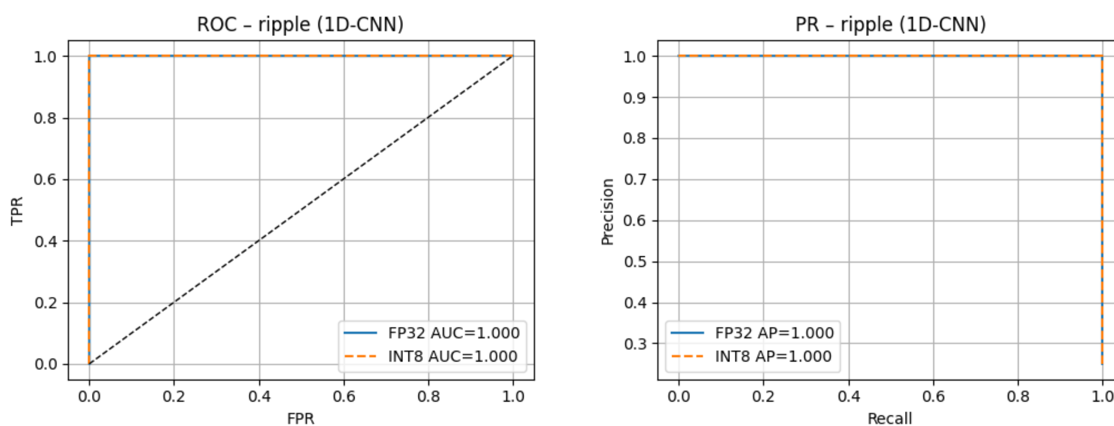
This appendix compares the floating-point (FP32) and quantized (INT8) versions of the models. It highlights the trade-offs between accuracy and efficiency in the embedded deployment setting.



(a) ROC curve — normal.

(b) PR curve — normal.

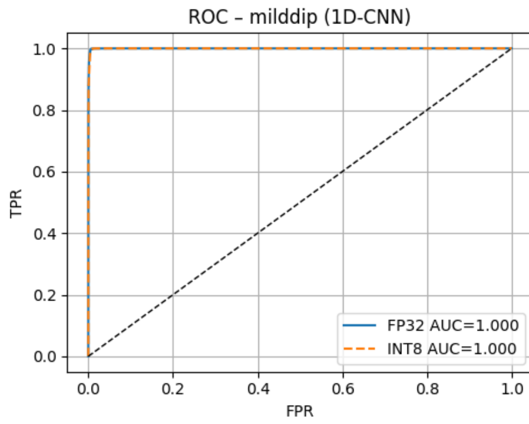
Figure C.1: ROC and PR curves for the normal class, comparing FP32 and INT8 quantized 1D-CNN.



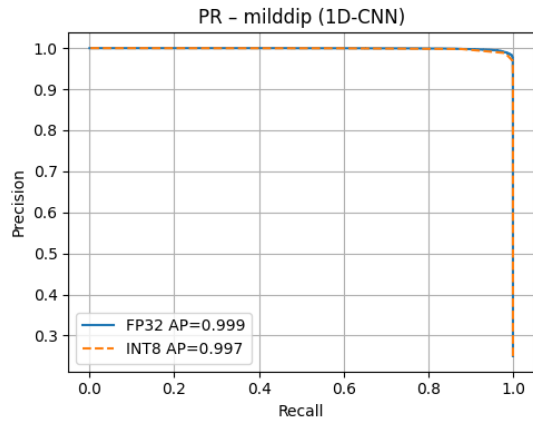
(a) ROC curve — ripple.

(b) PR curve — ripple.

Figure C.2: ROC and PR curves for the ripple class, comparing FP32 and INT8 quantized 1D-CNN.

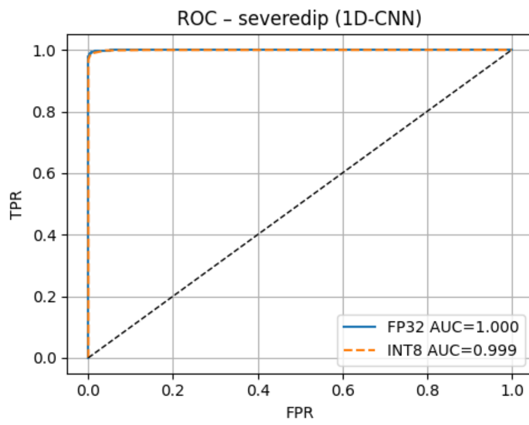


(a) ROC curve — milddip.

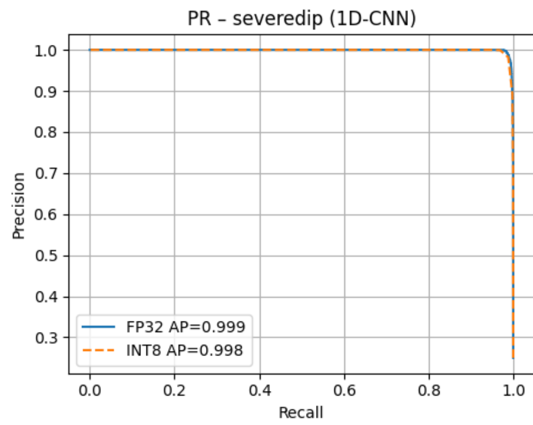


(b) PR curve — milddip.

Figure C.3: ROC and PR curves for the milddip class, comparing FP32 and INT8 quantized 1D-CNN.



(a) ROC curve — severedip.



(b) PR curve — severedip.

Figure C.4: ROC and PR curves for the severedip class, comparing FP32 and INT8 quantized 1D-CNN.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY