REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
الجمهورية الجزائرية الديمقراطية الشعبية

MINISTRY OF HIGHER EDUCATION
AND SCIENTIFIC RESEARCH

HIGHER SCHOOL IN APPLIED SCIENCES

–T L E M C E N–

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
ـ تلمسان ـ

Final year thesis

For the award of the master's Degree

Field of study : Electronics
Specialization : Industrial Informatics

Presented by : Amina HAMRI

Topic

Design and Implementation of a TinyML-Based Power Quality Disturbance Classification System on the STM32H743VIT6

Publicly defended on 30/09/2026 , before a jury composed of:

Mr Sidi Mohammed ABDI	MCA	ESSA-Tlemcen	President
Mr Ghouti ABDELLAOUI	MCA	ESSA-Tlemcen	Thesis Supervisor 01
Mrs Faiza MIMOUNI	MCA	ESSA-Tlemcen	Thesis Co-Supervisor
Mr Ahmed Ali KANOUN	MCB	USTHB	Examiner
Mr Reda BELMERAH	Chief ingineer, IT department	SONALGAZ	Socio-economic partner

Academic Year: 2025/2026


بسم الله الرحمن الرحيم

Dedication

To myself,

for the patience, strength, and courage I have shown throughout this challenging year. A year that brought unexpected difficulties and marked a turning point in my life. Despite everything I went through, I chose not to give up. I kept moving forward, even when it was difficult, and found the courage to complete this thesis. I dedicate this achievement to the person I have become through it all.

To my beloved mother, who, despite her own struggles and the burdens she carried, never stopped supporting me, especially throughout this challenging year. Her love, strength, and sacrifices have been my greatest source of courage. And to my father, you were always there for me, supporting me in my studies journy from the beginning, without your support, I wouldn't be where I am today.
You always taught me that anything is achievable, and now I can prove to you that you were right.

To my grandmother, whose prayers have blessed me throughout my entire journey, bringing me strength and peace.

To my friends, for the memories and support.

Acknowledgement

First and foremost, I thank Almighty God for granting me His blessings, guidance, and patience, and for giving me the courage to successfully complete this work on time.

I would like to extend my sincere gratitude and deep appreciation to my research supervisors, **Mr. Ghouti ABDELLAOUI** *and* **Mrs. Faiza MIMOUNI***, for giving me the opportunity and confidence to develop my research project. I am deeply grateful for their continuous support, valuable guidance, insightful advice, and enriching suggestions throughout the realization of this work. Their encouragement and expertise have been invaluable to the successful completion of this project.*

I also extend my respectful thanks to the honorable members of the jury for accepting to review and evaluate this work, and for their valuable time and insights.

We wish to express my gratitude to the entire department, especially to **Mr. ADJIM Ramz-Eddine Abderrazak***, FabLab Engineer, and* **Mr. KAID Abdelatif***, the laboratory engineer, for their patience, availability, and continuous support throughout our work. Their assistance has been invaluable.*

I would also like to warmly thank **Mr. Achraf SAI** *for his continuous support and encouragement throughout this project. He was always caring and willing to help me deploy my models onto real hardware, something that would have been very difficult to accomplish without him. I am sincerely grateful for his time and support.*

I would also like to express my heartfelt thanks to my kind friend, **Salsabil***, for her constant care, encouragement, and kindness. Her willingness to help, along with her sincere support, has meant a lot to me throughout this journey. I am truly grateful for her presence and for always being there when I needed support.*

I would like to express my sincere gratitude to my dear friends **Ikram** *and* **Hanane** *and to my colleague* **Iyad Atig** *for their valuable help. Their assistance, availability, and contributions were greatly appreciated.*

Finally, we would like to pay tribute to **Mr. ROUISSAT Boucherit***, Director of ESSAT,* **Mr. MEGNAFI Hichem***, Head of the Graduate Department,* **Mr. Ghouti ABDELLAOUI***, again as head of the electronic option industrial computing Department, and* **Mr. KHEFIF Sidi-Mohammed** *for granting us access to the FabLab.*

Abstract

This work presents the development of an intelligent embedded system for power quality disturbance monitoring and classification using artificial intelligence. A synthetic dataset containing 17,000 signals across 17 classes was used to train and compare two deep learning models: CNN-1D and BiLSTM. The models achieved test accuracies of 95.5% and 93%, respectively. The CNN-1D model was successfully deployed and validated on the STM32H743 microcontroller using TinyML, achieving an inference time of 24.873 ms. Noise robustness experiments revealed a significant performance degradation, highlighting the need for further improvements and additional validation using real electrical signals before practical deployment.

Keywords: Power Quality, Artificial Intelligence, Deep Learning, CNN-1D, BiLSTM, TinyML, STM32.

Résumé

Ce travail porte sur le développement d'un système embarqué intelligent pour la surveillance et la classification des perturbations de la qualité de l'énergie électrique à l'aide de l'intelligence artificielle. Une base de données synthétique de 17 000 signaux répartis en 17 classes a été utilisée pour entraîner et comparer deux modèles de Deep Learning : CNN-1D et BiLSTM. Les exactitudes obtenues sur les données de test sont respectivement de 95,5 % et 93 %. Le modèle CNN-1D a été déployé et validé sur le microcontrôleur STM32H743 à l'aide de TinyML, avec un temps d'inférence de 24,873 ms. Les tests de robustesse ont révélé une dégradation des performances en présence de bruit, soulignant la nécessité d'améliorations et d'une validation supplémentaire sur des signaux réels.

Mots-clés : Qualité de l'énergie, Intelligence artificielle, Deep Learning, CNN-1D, BiLSTM, TinyML, STM32.

الملخص

يتناول هذا العمل تطوير نظام مدمج ذكي لمراقبة اضطرابات جودة الطاقة الكهربائية وتصنيفها باستخدام الذكاء الاصطناعي. اعتمدت الدراسة على قاعدة بيانات اصطناعية تضم 17,000 إشارة موزعة على 17 فئة، مع تطوير ومقارنة نموذجي التعلم العميق CNN-1D و BiLSTM . حقق النموذجان دقة تصنيف على بيانات الاختبار بلغت 95.5% و 93% على التوالي. كما تم نشر نموذج CNN-1D والتحقق من تشغيله على المتحكم STM32H743 باستخدام تقنية TinyML ، بزمن استدلال قدره 24.873 ms . أظهرت اختبارات الضوضاء تراجعا ملحوظا في الأداء، مما يؤكد ضرورة تحسين متانة النموذج وإجراء اختبارات إضافية باستخدام إشارات كهربائية حقيقية قبل اعتماده عمليا.

الكلمات المفتاحية: جودة الطاقة الكهربائية، الذكاء الاصطناعي، التعلم العميق، CNN-1D ، BiLSTM ، TinyML ، STM32 .

Contents

List of Figures

List of Tables

Abbreviations List

General introduction **1**

1 State of the Art **4**
1.1 Introduction 4
1.2 Artificial Intelligence for Power Quality Classification 4
1.2.1 Artificial Neural Networks 5
1.2.2 Machine Learning Paradigms 5
1.2.3 Deep Learning 6
1.3 Convolutional Neural Networks 7
1.3.1 CNN Principles 7
1.3.2 One-Dimensional Convolution 8
1.3.3 ReLU Activation Function 8
1.4 Recurrent Neural Networks (RNNs) 9
1.4.1 Long Short-Term Memory (LSTM) 9
1.4.2 LSTM Cell Architecture 9
1.4.3 Bidirectional Long Short-Term Memory (BiLSTM) 9
1.5 Tiny Machine Learning (TinyML) 10
1.5.1 Challenges of Embedded Deep Learning 10
1.5.2 TensorFlow Lite for Microcontrollers 11
1.5.3 STM32H743VIT6 as an AI Platform 11
1.6 Power Quality Disturbances 12
1.7 Related Works 15
1.7.1 CNN: A Consistently High-Accuracy and Efficient Baseline 15
1.7.2 BiLSTM: Included as a Sequential-Modeling Baseline Despite Weaker Reported Accuracy 16
1.7.3 Transformer: Motivated by State-of-the-Art Accuracy Under Realistic Noise Conditions 16
1.7.4 Summary and Motivation for the Selected Architectures 16

1.8 Conclusion 17

2 Methodology and Implementation **18**
2.1 Introduction 18
2.2 Database Presentation 18
2.3 Database Treatment 20
2.3.1 Verification of Class Consistency 20
2.3.2 Splite test/training and Normalisation 22
2.4 Time Series Data 22
2.5 Data Analysis for Feature Scaling Selection 23
2.5.1 Descriptive Statistics 23
2.5.2 Distribution Analysis 25
2.5.3 Outlier Analysis Using Boxplots 26
2.5.4 Statistical Analysis and Justification of the Selected Normalization Method 28
2.6 Data Leakage in Machine Learning 28
2.6.1 Missing Values 29
2.6.2 Time-Series Data 29
2.6.3 Normalization and Standardization 29
2.6.4 Training and Testing Split and normalisation 29
2.7 Normalization Strategies and Their Evaluation 30
2.7.1 First Approach: Feature-wise Min-Max Normalization 30
2.7.2 Second Approach: Global Normalization with Incorrect Dataset Reconstruction 31
2.7.3 Third Approach: Correct Global Min-Max Normalization 32
2.7.4 Fourth approach: RobustScaler normalization 33
2.7.5 Comparison and Selection of the Normalization Strategy 34
2.8 Software tools and API 35
2.8.1 Python 36
2.8.2 Google Colaboratory 36
2.8.3 TensorFlow and Keras 37
2.8.4 X-Cube-AI 37
2.8.5 Validation on target 38
2.9 CNN1D Model Architecture 38
2.9.1 Understanding the building blocks of the CNN-1D network 41
2.9.2 Hyperparameter Selection 42
2.10 Bidirectional Long Short-Term Memory Architecture 42
2.10.1 Understanding the building blocks of the BiLSTM network 43
2.10.2 Hyperparameter Selection 45
2.10.3 Activation functions used in the proposed models 45
2.11 TinyML Deployment 46
2.11.1 Deployment on STM32H743 46
2.11.2 Embedded Inference Validation Pipeline 46
2.12 Acquisition Validation and Signal Conditioning 47
2.12.1 Sampling Frequency and Acquisition Window 47
2.12.2 DAC–ADC Signal Generation and Acquisition 48
2.12.3 Final Acquisition Validation 48
2.13 Noise Robustness Evaluation Prior to Real-World Deployment 48

2.13.1 Initial Approach . . . 48
2.13.2 Identification of the Problem . . . 49
2.13.3 Preliminary Robustness Test . . . 49
2.14 Conclusion . . . 50

3 Results and Discussion **51**
3.1 Introduction . . . 51
3.2 Dataset Verification Results . . . 51
3.3 Training Results . . . 52
3.3.1 Convolutional Neural Network Results . . . 52
3.3.2 Bidirectional Long Short-Term Memory Results . . . 52
3.4 Performance Comparison . . . 53
3.4.1 Overall Model Performance . . . 53
3.4.2 Classification Metrics . . . 53
3.4.3 Interpretation of the Classification Results for the CNN-1D Model . 54
3.4.4 Overall Analysis of the CNN-1D Model . . . 55
3.4.5 Interpretation of the Classification Results for the BiLSTM Model . 56
3.4.6 Overall Analysis of the BiLSTM Model . . . 57
3.5 Confusion Matrix Comparison . . . 58
3.5.1 Confusion Matrix Analysis of the BiLSTM Model . . . 58
3.5.2 Confusion Matrix Analysis of the CNN-1D Model . . . 61
3.5.3 Comparison of the CNN and BiLSTM Confusion Matrices . . . 64
3.6 Deployment Results on STM32H743 . . . 65
3.6.1 Analysis of the CNN-1D Embedded Profiling . . . 65
3.6.2 Analysis of the BiLSTMm Embedded Profiling . . . 70
3.6.3 Memory Resource Comparison and Final Model Selection . . . 72
3.7 Explorations and Discarded Architecture Choices . . . 73
3.7.1 Transformer-Based Classification (Custom Implementation) . . . 73
3.8 Experimental Results . . . 74
3.8.1 Engineering Decision: Delayed Real-Signal Hardware Validation . 74
3.8.2 Limitations of the Artificial-Noise Evaluation . . . 75
3.8.3 Validation Path Toward Real-World Deployment . . . 75
3.9 Conclusion . . . 76

General conclusion **77**

General Conclusion **77**

Bibliography **80**

List of Figures

2.1 Construction of the labeled Power Quality Disturbance dataset from individual CSV files. 21
2.2 Comparison between Gaussian and arcsine distributions. 25
2.3 Comparison of the amplitude distributions of the complete dataset and the pure sinusoidal class. 26
2.4 Boxplot representation and outlier detection based on Tukey's interquartile range (IQR) criterion. 27
2.5 Global Boxplot of the used dataset 27
2.6 Effect of the first approach normalization on a pure sinusoidal signal. 31
2.7 Normalized pure sinusoidal 33
2.8 Effect of RobustScaler normalization on a pure sinusoidal signal. 34
2.9 Deployment and validation workflow of the AI model on the STM32 microcontroller. 36
2.10 X-Cube-AI package in STM32Cube MX 37
2.11 X-Cube-AI anlysis before performing the on-target validation 38
2.12 Architecture of the Proposed CNN-1D Model 40
2.13 Architecture of the proposed BiLSTM Model 43

3.1 BiLSTM Model Confusion Matrix 58
3.2 CNN-1D Model Confusion Matrix 61
3.3 Validation on target results 68
3.4 Memory and Flash resource utilization for the CNN-1D model 69
3.5 STM32Cube.AI memory analysis of the BiLSTM model showing insufficient available RAM 71
3.6 Used Flash and RAM memories 72

List of Tables

1.1 Summary of results motivating the choice of CNN, BiLSTM, and Transformer architectures. 17

2.1 Minimum and maximum values of each class in the SEED-PQD-v1 dataset before normalization. 24
2.2 Comparison of the investigated normalization approaches. 35
2.3 Nature of each block: standard CNN component or design choice specific to this architecture 42
2.4 Nature of each block: standard recurrent-network component or design choice specific to this architecture 44
2.5 Activation functions used in the proposed CNN-1D and BiLSTM models . 46
2.6 Model validation results 47

3.1 BiLSTM Training Results 53
3.2 Summary of the main model performance and embedded deployment results. 53
3.3 Classification performance for each PQD class. 55
3.4 Classification performance for each PQD class using the BiLSTM model. . 56
3.5 Main CNN-1D profiling results on the STM32H743 65
3.6 Layer-by-layer execution time of the CNN-1D model 67
3.7 Comparison of CNN-1D and BiLSTM memory requirements on the STM32H743VIT6 72
3.8 Reported vs. reproduced accuracy for Transformer-based classification . . . 73
3.9 CNN performance on clean and noisy test datasets. 74

Abbreviations List

Acronyme	Signification
AC	Alternating Current.
ADC	Analog-to-Digital Converter.
AI	Artificial Intelligence.
ANN	Artificial Neural Network.
API	Application Programming Interface.
AWGN	Additive White Gaussian Noise.
BiLSTM	Bidirectional Long Short-Term Memory.
CAN	Controller Area Network.
CNN	Convolutional Neural Network.
CPU	Central Processing Unit.
CSV	Comma-Separated Values.
DAC	Digital-to-Analog Converter.
DC	Direct Current.
DMA	Direct Memory Access.
DSP	Digital Signal Processing.
DTCM	Data Tightly-Coupled Memory.
FN	False Negative.
FP	False Positive.
FPU	Floating-Point Unit.
GPU	Graphics Processing Unit.
GRU	Gated Recurrent Unit.
I2C	Inter-Integrated Circuit.
IEEE	Institute of Electrical and Electronics Engineers.
IoT	Internet of Things.
IQR	Interquartile Range.
ITCM	Instruction Tightly-Coupled Memory.
LGAN	Label-Guided Attention Network.
LSTM	Long Short-Term Memory.
MAC	Multiply-Accumulate.
MAE	Mean Absolute Error.
MCU	Microcontroller Unit.
MLP	Multilayer Perceptron.
MPU	Memory Protection Unit.
NaN	Not a Number.

Acronyme	Signification
NPU	Neural Processing Unit.
PQD	Power Quality Disturbance.
PQE	Power Quality Event.
RAM	Random Access Memory.
ReLU	Rectified Linear Unit.
RMS	Root Mean Square.
RMSE	Root Mean Square Error.
RNN	Recurrent Neural Network.
ROM	Read-Only Memory.
SNR	Signal-to-Noise Ratio.
SPI	Serial Peripheral Interface.
SRAM	Static Random Access Memory.
SVM	Support Vector Machine.
SWD	Serial Wire Debug.
TCM	Tightly-Coupled Memory.
TFLite	TensorFlow Lite.
THD	Total Harmonic Distortion.
TinyML	Tiny Machine Learning.
TP	True Positive.
TSC	Time Series Classification.
UART	Universal Asynchronous Receiver-Transmitter.
USB	Universal Serial Bus.
XPQRS	Expert Power Quality Recognition System.

General introduction

According to the IEEE, power quality is described as the concept of powering and grounding electronic equipment in a manner suitable for the operation of that equipment and compatible with the premise wiring system and other connected equipment. In other words, power quality can be defined as the set of electrical conditions within which equipment is expected to operate as intended, without significant loss of performance or reduction in service life.

Power quality disturbances, such as voltage sags, swells, interruptions, harmonics, flicker, and transients, affect a wide range of critical sectors. In industrial systems, they may lead to production losses and equipment malfunction; in healthcare facilities, they can compromise the reliable operation of sensitive equipment; and in renewable energy installations, they may raise concerns regarding grid compliance. Consequently, the development of intelligent systems capable of detecting and classifying these disturbances represents an important multidisciplinary challenge combining power systems, embedded electronics, signal processing, and artificial intelligence.

Recent advances in deep learning and embedded artificial intelligence provide new possibilities for developing monitoring systems capable of automatically identifying power quality disturbances directly from measured waveforms. In particular, Tiny Machine Learning (TinyML) enables trained models to be deployed on resource-constrained microcontrollers, bringing the classification process closer to the measurement source and reducing dependence on external computing platforms.

Problem Statement

Conventional approaches to power quality monitoring generally rely on explicit signal-processing pipelines, computing features such as the Fast Fourier Transform (FFT), Total Harmonic Distortion (THD), or Root Mean Square (RMS) values, followed by threshold-based decision rules. While effective, such approaches require the relevant features and detection criteria to be explicitly designed and adapted to the disturbances under consideration.

Deep learning provides an alternative by allowing classification models to learn discrim-

inative features directly from raw or preprocessed voltage waveforms. Once trained offline, such models can potentially provide a compact classification pipeline suitable for embedded implementation. However, achieving high classification accuracy in software does not necessarily guarantee successful deployment on a resource-constrained microcontroller, where memory usage, computational complexity, and inference latency become important design constraints.

A further challenge concerns the transition from controlled datasets to real measurements. Many power quality classification models are developed and evaluated using synthetic datasets generated under predefined conditions. Real acquired signals, however, may contain measurement noise, sensor inaccuracies, offsets, amplitude variations, and other effects introduced by the acquisition chain.

The challenge addressed in this thesis is consequently not limited to obtaining high classification accuracy. It also involves selecting an appropriate architecture, preparing the data without compromising the physical characteristics of the electrical waveforms, evaluating the feasibility of embedded inference, and identifying the limitations that must be addressed before deployment on real measured signals.

Objectives

The primary objective of this master's thesis is to design and develop an intelligent embedded system for the monitoring and classification of 220 V power quality disturbances using artificial intelligence.
More specifically, this work aims to:

- review the theoretical background and recent literature on power quality disturbance classification, with particular attention to CNN, BiLSTM, and Transformer-based approaches;
- prepare, verify, and statistically analyze a labeled power quality disturbance dataset suitable for supervised learning;
- investigate preprocessing and normalization strategies while preserving the amplitude and morphological characteristics of the electrical waveforms;
- design, train, and evaluate a one-dimensional Convolutional Neural Network (CNN-1D) and a Bidirectional Long Short-Term Memory (BiLSTM) network for power quality disturbance classification;
- investigate a Transformer-based classifier as an additional architecture motivated by results reported in recent literature;
- deploy and validate the retained models on the STM32H743VIT6 microcontroller using TinyML tools and techniques;
- evaluate the trade-offs between classification accuracy and embedded deployment constraints, particularly model size, memory requirements, and inference latency;
- develop and validate an experimental signal generation, conditioning, and acquisition pipeline for future testing with measured signals;
- evaluate the sensitivity of the classification approach to noise and identify the requirements for future real-world validation.

Research Hypotheses

In accordance with the scientific methodology adopted in this work, the following research hypotheses are formulated based on the related works reviewed in Chapter 1 and the objectives of the proposed system:

- **H1:** Based on the results reported in the literature, the CNN-1D architecture is expected to achieve higher classification accuracy than the BiLSTM architecture on the power quality disturbance dataset used in this work.

- **H2:** Due to their different architectural characteristics, the CNN-1D and BiLSTM models are expected to exhibit different memory and computational requirements, influencing their suitability for real-time inference on the resource-constrained STM32H743VIT6 microcontroller.

- **H3:** Models trained exclusively on clean synthetic power quality signals are expected to experience a degradation in classification performance when the input conditions differ from those represented in the training dataset, highlighting the need for validation and potential adaptation using representative real-world measurements before practical deployment.

These hypotheses are investigated through the methodological and experimental framework presented in Chapter 2 and evaluated based on the results discussed in Chapter 3.

Thesis Organization

This thesis is organized into three chapters:

- **Chapter 1:** presents the state of the art related to power quality disturbances and artificial intelligence techniques for their classification. It introduces the main machine learning and deep learning concepts relevant to this work, with particular emphasis on CNN and BiLSTM architectures, TinyML, and embedded AI deployment. Related works involving CNN, BiLSTM, and Transformer-based approaches are subsequently reviewed to motivate the architectures investigated in this thesis.

- **Chapter 2:** presents the methodology and implementation adopted in this work. It describes the SEED-PQD-v1 dataset, its verification and statistical analysis, the investigated preprocessing and normalization strategies, and the design and training of the CNN-1D and BiLSTM models. The chapter also presents the embedded deployment methodology on the STM32H743VIT6, together with the inference pipeline and the experimental signal generation, conditioning, and acquisition framework used to prepare the system for hardware validation.

- **Chapter 3:** presents and discusses the experimental results. The CNN-1D and BiLSTM models are evaluated in terms of classification performance and embedded implementation characteristics. A Transformer-based classifier is also investigated as an additional experiment and compared with the retained approaches. Finally, the chapter examines the robustness of the classification system under noisy input conditions, discusses the limitations encountered during development, and identifies the requirements for future validation using representative real-world measurements.

Chapter 1

State of the Art

1.1 Introduction

This chapter presents the state of the art related to the methods, techniques, and technologies involved in power quality disturbance classification using artificial intelligence and embedded systems. It establishes the theoretical background necessary to understand the approaches adopted in this work and positions the proposed solution with respect to existing research.

The chapter first introduces the main types of power quality disturbances and their characteristics. The application of artificial intelligence to power quality classification is then discussed, followed by an overview of machine learning paradigms and artificial neural networks. Particular attention is subsequently given to the two deep learning architectures investigated in this work, namely one-dimensional Convolutional Neural Networks (CNNs) and Bidirectional Long Short-Term Memory (BiLSTM) networks.

The chapter then addresses the deployment of artificial intelligence models on resource-constrained embedded systems through the concepts of Tiny Machine Learning (TinyML), with a focus on the tools and hardware platform used for embedded inference. Finally, related works on power quality classification are reviewed, including approaches based on CNN, BiLSTM, and Transformer architectures. The chapter concludes with a synthesis of the reviewed studies, highlighting their main contributions and limitations in relation to the objectives of this work.

1.2 Artificial Intelligence for Power Quality Classification

Recently, there has been a growing interest in exploiting the power of learning-based techniques for the classification of PQEs. Convolutional Neural Networks (CNNs) have demonstrated effectiveness in conjunction with other models [1], [2], [3], [4], [5], [6], [7].

Recurrent Neural Networks (RNNs) are also employed in PQE classification [8]. Mishra emphasizes the prevalence of Artificial Neural Networks (ANNs) for PQE classification [9]. Qui et al. integrate time and frequency domain information using multi-fusion CNN [4].

Addressing hidden patterns, a deep autoencoder followed by feature categorization is proposed [10]. Sahani and Dash combine variational mode decomposition with an improved particle swarm optimization technique [11].

Fu et al. introduce PowerCog, utilizing optimized CNN and SVM for precise identification of PQEs in noisy environments [6]. Gu et al. tackle the challenge of the simultaneous occurrence of multiple PQEs, introducing a label-guided attention method (LGAN) with a single dimension CNN [12].

Chiam et al. present two methods for PQE classification [13], [14], yet they do not account for normal signal variations and DC offset in measurements. Additionally, a single-shot power quality disturbance detection approach based on CNN is proposed [5]. Investigating adversarial attacks on CNN-based PQE classifiers, solutions are proposed [7].

1.2.1 Artificial Neural Networks

An Artificial Neural Network is a computational model inspired by the biological nervous system. It consists of interconnected neurons organized into input, hidden, and output layers. ANNs learn by adjusting the weights of the connections between neurons during training to approximate complex nonlinear relationships.
Examples of ANNs include:

- Multilayer Perceptron (MLP)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- LSTM
- GRU
- Transformer

1.2.2 Machine Learning Paradigms

Machine learning can be broadly categorized into different learning paradigms according to the way models learn from available data and feedback. The main paradigms are supervised, unsupervised, and reinforcement learning.

Supervised Learning

Supervised learning is a machine learning paradigm in which a model is trained using a labeled dataset, where each input sample is associated with a known target output (class label or numerical value). During training, the algorithm learns the relationship between the input features and the corresponding labels by minimizing a predefined loss function. Once trained, the model can accurately predict the labels of previously unseen data. Supervised learning is the most widely used approach for classification and regression problems, including power quality disturbance (PQD) classification.

Unsupervised Learning

Unsupervised learning is a machine learning paradigm in which the training data do not contain class labels. The objective is to automatically discover hidden structures, patterns, or relationships within the data without prior knowledge of the expected outputs. Typical unsupervised learning tasks include clustering, dimensionality reduction, anomaly detection, and feature extraction. This approach is particularly useful when labeled datasets are unavailable or costly to obtain.

Reinforcement Learning

Reinforcement learning is a machine learning paradigm in which an intelligent agent interacts with an environment by performing actions and receiving feedback in the form of rewards or penalties. Through repeated interactions, the agent learns an optimal decision-making policy that maximizes the cumulative reward over time. Reinforcement learning is primarily applied to sequential decision-making problems such as robotics, autonomous vehicles, game playing, and industrial process control.

The proposed power quality disturbance classification system is based on the **supervised learning** paradigm. In supervised learning, a model is trained using a labeled dataset in which each input sample is associated with a predefined class label. The learning process aims to establish a mapping between the input voltage waveform and its corresponding disturbance category by minimizing a classification loss function. In this work, a dataset containing **17,000 labeled voltage signals** representing **17 different power quality disturbance classes** was used to train deep learning models, namely a CNN-1D and a BiLSTM network. After training, the models are capable of accurately classifying previously unseen power quality disturbances, making supervised learning the most appropriate machine learning approach for this application.

1.2.3 Deep Learning

Deep learning is a subfield of machine learning that uses artificial neural networks composed of multiple processing layers to automatically learn hierarchical representations from data. Unlike conventional machine learning approaches, which often require manually designed features, deep learning models can learn increasingly complex features directly from raw or minimally processed data through successive layers of nonlinear transformations. This capability has made deep learning particularly effective for complex pattern recognition and classification tasks involving images, audio, and time-series signals.

Deep learning is generally based on artificial neural networks with multiple layers, where each layer transforms the representation produced by the preceding layer. During training, the network parameters are adjusted through backpropagation and an optimization algorithm to minimize a predefined loss function. Depending on the nature of the data and the task, different deep learning architectures can be employed, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformer architectures.

In the context of power quality disturbance classification, deep learning is particularly relevant because electrical disturbances are characterized by temporal variations in the volt-

age waveform. Deep learning models can learn discriminative patterns directly from these waveforms, reducing the need for extensive manual feature extraction.

Feature extraction refers to the process of manually computing meaningful characteristics from raw signals in order to help a model distinguish between different classes. For example, in power quality analysis, common handcrafted features include the root mean square (RMS) value of the voltage, the total harmonic distortion (THD), the peak amplitude, or the frequency spectrum obtained using a Fourier transform. These features are then used as inputs to traditional machine learning algorithms.
In contrast, deep learning models such as CNN-1D and Transformers can automatically learn such relevant patterns directly from the raw voltage signal samples. In this work, CNN-1D and Bidirectional Long Short Term Memory-based architectures are investigated to classify the different power quality disturbance categories from voltage signal samples.

1.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are among the most successful deep learning architectures for automatic feature extraction and pattern recognition. Originally developed for computer vision, CNNs have also been successfully applied to one-dimensional signal classification due to their ability to learn hierarchical features directly from raw or normalized data. A typical CNN is composed of convolutional, activation, pooling, and classification layers that are jointly optimized during training.

CNN are initially developed for computer vision applications such as image classification, object detection, and image segmentation, CNNs have also demonstrated excellent performance in one-dimensional signal processing and time-series classification tasks. [15]

1.3.1 CNN Principles

A typical CNN architecture consists of an input layer followed by several convolutional, activation, pooling, and fully connected layers.[15]
Unlike traditional machine learning methods that rely on handcrafted features, CNNs automatically learn hierarchical feature representations directly from the input data during the training process.
The general mathematical representation of a neural network layer is expressed as

$$Y = f(XW + b) \tag{1.1}$$

where

- X is the input vector;
- W denotes the weight matrix;
- b is the bias vector;
- $f(\cdot)$ represents the activation function;
- Y is the output of the layer.

In CNNs, the convolutional layer extracts local features from the input signal by applying learnable kernels, while nonlinear activation functions improve the network's ability to model complex patterns.
Pooling layers reduce the dimensionality of the extracted feature maps and improve computational efficiency.
Finally, fully connected layers combine the extracted features to perform the final classification task.

1.3.2 One-Dimensional Convolution

Convolutional neural networks are typically classified by the dimensionality of the space over which their kernels move. A **2D convolution**, commonly used for image processing, slides a kernel across two spatial dimensions (height and width) to capture spatial correlations between neighboring pixels, such as edges, textures, or shapes. A **1D convolution**, in contrast, slides a kernel along a single dimension, typically time, making it naturally suited to sequential or temporal data such as voltage and current waveforms. Instead of learning spatial patterns, the 1D kernel learns temporal patterns: how the signal amplitude evolves from one sample to the next.[16]

Each 1D filter specializes in detecting a particular signal pattern, such as abrupt changes, oscillations, or repetitive structures. During training, the kernel coefficients are automatically optimized to maximize classification performance. Compared with handcrafted feature extraction techniques, the convolution operation enables the network to learn task-specific representations directly from the input signal.
The convolution operation can be expressed as:

$$Y = f(Conv1D(X,W)+b) \tag{1.2}$$

where:

- ***X*:** denotes the input signal
- ***W*:** is the convolution kernel
- ***b*:** is the bias term
- ***f()*:** is the activation function

1.3.3 ReLU Activation Function

After convolution, a nonlinear activation function is applied to increase the representational capability of the network. The Rectified Linear Unit (ReLU) is one of the most widely used activation functions because of its computational simplicity and its ability to alleviate the vanishing gradient problem encountered in deep neural networks. ReLU preserves positive values while setting negative values to zero, allowing the network to learn complex nonlinear relationships efficiently.
The ReLU function is defined as

$$f(x) = max(0,x) \tag{1.3}$$

where x represents the input of the activation layer. [17]

1.4 Recurrent Neural Networks (RNNs)

While CNNs are feedforward only, RNNs are designed to handle sequential data by introducing feedback connections, allowing information from previous time steps to influence the current output. This gives RNNs a form of memory, making them well suited for tasks where temporal order matters, such as time series analysis, signal processing, and language modeling.

1.4.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) introduced by Hochreiter and Schmidhuber to address a key limitation of conventional RNNs: their difficulty in learning long-term dependencies, caused by the so-called *vanishing gradient problem*. A standard RNN tends to "forget" information from early time steps as the sequence gets longer, which makes it unreliable for tasks where distant past information still matters.
An LSTM overcomes this limitation by adding an internal memory, called the **cell state**, which runs through the entire sequence and can carry relevant information over long time spans with little loss. Access to this memory is controlled by a set of learned mechanisms called **gates**, which decide at each time step what information should be kept, updated, or discarded. This design allows the LSTM to selectively retain long-term information while still reacting to new, short-term input, making it well suited for tasks involving sequential or time-varying data, such as speech recognition, text analysis, and the classification of time-domain signals.

1.4.2 LSTM Cell Architecture

At each time step, an LSTM cell receives the current input together with two pieces of information carried over from the previous step: the **cell state** (the long-term memory) and the **hidden state** (the short-term output). Three gates regulate how this information is updated:

- **Forget gate**: decides how much of the previous cell state should be kept or discarded.
- **Input gate**: decides how much of the new information from the current input should be added to the cell state.
- **Output gate**: decides how much of the updated cell state should be passed on as the hidden state (the output) for the current time step.

Each gate outputs values between 0 and 1, acting as a filter that controls how much information is allowed to pass through. By combining these three gates, the LSTM cell can preserve important information over long sequences while filtering out irrelevant details, which gives it a clear advantage over a conventional RNN cell.

1.4.3 Bidirectional Long Short-Term Memory (BiLSTM)

Bidirectional Long Short-Term Memory (BiLSTM) is an extension of the conventional LSTM that processes an input sequence in both the forward and backward directions. Instead of relying on a single LSTM layer that reads the sequence from beginning to end, a

BiLSTM combines two independent LSTM layers: one that processes the sequence forward (past to present) and another that processes it backward (future to present). The hidden states produced by both layers are then combined to form the final output representation.

This dual processing allows each time step to be represented using information from both its past and future context, rather than the past alone. As a result, BiLSTMs are particularly effective for sequence classification tasks where the entire signal is available beforehand, since the model can leverage the full sequence to make its decision. Graves and Schmidhuber [18]showed that this bidirectional approach allows the network to better exploit contextual information, leading to improved performance over unidirectional LSTMs in sequence classification tasks.
A BiLSTM consists of two independent LSTM layers:

- One processes the sequence in the forward direction.
- The other processes it in the reverse direction.
- Their hidden representations are subsequently combined to form the output representation.

1.5 Tiny Machine Learning (TinyML)

TinyML, a sub-genre of Edge-AI, is getting more and more attention, especially in the field of IoT and embedded real-time systems. It is the place where the world of deep learning and embedded systems meet each other. The diversity of hardware, frameworks and applications is huge and TinyML systems face different challenges due to their low power characteristic and constrained resources with just a couple of kB RAM and CPU frequency in the lower MHz area.
Since embedded microcontrollers have limited Flash, RAM and computational resources, lightweight neural networks such as MLP and CNN1D are more suitable for TinyML deployment. [19]

1.5.1 Challenges of Embedded Deep Learning

Deploying deep learning models on embedded microcontrollers introduces a set of constraints that are largely absent in conventional deep learning workflows designed for GPUs or cloud servers. These constraints span memory, computation, and power consumption, and they directly influence the choice of model architecture, precision, and optimization strategy.

Memory Constraints

Embedded microcontrollers typically offer only a few tens to a few hundred kilobytes of RAM and a few hundred kilobytes to a few megabytes of Flash memory, in stark contrast to the gigabytes of memory available on GPUs or desktop CPUs. This directly limits both the size of the model that can be stored (weights and biases in Flash) and the amount of intermediate data (activations, buffers) that can be held in RAM during inference. As a result, deep architectures with millions of parameters, such as standard CNN2D or large RNN/LSTM networks, are generally impractical without aggressive compression. Techniques such as quantization (e.g., converting 32-bit floating-point weights to 8-bit integers), pruning, and

knowledge distillation are commonly employed to reduce the memory footprint while preserving acceptable accuracy.

Computational Constraints

Unlike GPUs, which can perform massively parallel operations, microcontrollers rely on CPU cores running at clock frequencies typically between a few MHz and a few hundred MHz, with little to no hardware acceleration for matrix multiplications or convolutions (unless equipped with a dedicated DSP or NPU). This significantly limits the computational throughput available for inference, making complex operations such as large convolutions or attention mechanisms costly in terms of latency. Consequently, lightweight architectures such as MLPs, 1D CNNs, or depthwise-separable convolutions, are favored, as they require fewer multiply-accumulate (MAC) operations while still capturing relevant features from the input signal.

Power Consumption

Since many TinyML applications are designed for battery-powered or energy-harvesting devices, power consumption is often the most critical constraint. Every additional computation or memory access directly translates into energy drawn from a limited power budget, which can determine whether a device operates for days or for years without recharging. Reducing model complexity and inference time not only addresses memory and computational limits but also has a direct and often dominant impact on power efficiency, making energy-aware model design an essential part of the TinyML development pipeline alongside accuracy and latency considerations.

1.5.2 TensorFlow Lite for Microcontrollers

we use TensorFlow to build and train models. A TensorFlow model is essentially a set of instructions that tell an interpreter how to transform data in order to produce an output. When we want to use our model, we just load it into memory and execute it using the TensorFlow interpreter.

However, TensorFlow's interpreter is designed to run models on powerful desktop computers and servers. Since we will be running our models on tiny microcontrollers, we need a different interpreter that's designed for our use case. Fortunately, Tensor-Flow provides an interpreter and accompanying tools to run models on small, lowpowered devices. This set of tools is called ***TensorFlow Lite***. [20]

Before TensorFlow Lite can run a model, it first must be converted into the Tensor-Flow Lite format and then saved to disk as a file. We do this using a tool named the TensorFlow Lite Converter. The converter can also apply special optimizations aimed at reducing the size of the model and helping it run faster, often without sacrificing performance. [20]

1.5.3 STM32H743VIT6 as an AI Platform

In this work we will use the STM32H743VIT6 which is a 32-bit microcontroller from STMicroelectronics built around an Arm Cortex-M7 core, which makes it a strong candidate

for embedded AI and TinyML workloads despite its constrained resource envelope compared to GPU-based platforms.

- **Core and Frequency:** The device integrates an Arm Cortex-M7 core with a double-precision FPU and an L1 cache of 16 Kbytes of data and 16 Kbytes of instruction cache, running at frequencies of up to 480 MHz. It also supports a full set of DSP instructions and includes a Memory Protection Unit (MPU), which together make it well suited to accelerate the multiply-accumulate operations that dominate neural network inference.

- **Memory:** In terms of memory, the STM32H743VIT6 offers up to 2 Mbytes of Flash memory with read-while-write support, and up to 1 Mbyte of RAM.

- **Peripherals and Connectivity:** The microcontroller further provides a dual-mode Quad-SPI memory interface running up to 133 MHz and a flexible external memory controller supporting SRAM, PSRAM, SDRAM/LPSDR SDRAM, and NOR/NAND Flash, which allows the memory footprint to be extended when needed for larger models or datasets. It also integrates communication interfaces such as CAN bus, Ethernet, I2C, UART/USART, SPI, and USB OTG, enabling the transmission of inference results or raw acquired signals (e.g., voltage/current waveforms in a power quality context) to external systems such as a Bluetooth module or a host application.

- **Suitability for Edge AI:** Taken together, the high clock speed, hardware FPU, DSP instruction set, and generous on-chip memory, position the STM32H743VIT6 as one of the more capable platforms in the STM32 family for deploying lightweight neural networks (e.g., MLP, CNN1D) at the edge using frameworks such as X-CUBE-AI and TensorFlow Lite for Microcontrollers, making it suitable for real-time, on-device inference tasks such as embedded power quality disturbance classification.

1.6 Power Quality Disturbances

They are caused by electrical disturbances such as transients, surges, sags and harmonics. The power quality signature, or characteristic, of the disturbance identifies the type of power quality problem. The nature of the variation in the basic components of the sine wave, i.e., voltage, current, and frequency, identifies the type of power quality problem. [21]

In this work, the following power quality disturbances are considered as the **classes used for the classification task**. Each class represents a specific type of waveform disturbance that the developed AI models are trained to identify and distinguish.

Pure Sinusoidal

A pure sinusoidal waveform represents the normal operating condition of the electrical signal, without the presence of power quality disturbances. Ideally, the voltage follows a sinusoidal waveform at the fundamental frequency with constant amplitude and frequency. In the classification task, this class is used as the reference or normal class against which the different power quality disturbances are distinguished.

Voltage Sag

A voltage sag is a short-duration reduction in the RMS voltage to between 0.1 and 0.9 per unit (p.u.) of its nominal value, lasting from 0.5 cycles to 1 minute. It is commonly caused by system faults, motor starting, large load energization, recloser or circuit breaker operation, and capacitor bank switching. Voltage sags primarily affect sensitive electronic equipment such as computers, microcontrollers, and adjustable-speed drives. They are characterized by their magnitude, duration, and phase-angle jump.[21]

Voltage Swell

A voltage swell is a temporary increase in RMS voltage to between 1.1 and 1.8 p.u. for a duration ranging from 0.5 cycles to 1 minute. Voltage swells are generally caused by single line-to-ground faults, switching off large loads, or energizing capacitor banks. The severity of a swell depends on the fault location, system grounding, and network impedance.[21]

Interruptions

An interruption is defined as a reduction of the supply voltage to less than 0.1 p.u. for a duration not exceeding 1 minute. Interruptions are usually caused by power system faults, equipment failures, or protection system operations and may be preceded by a voltage sag.[21]

Harmonics

Harmonics are sinusoidal voltage or current components whose frequencies are integer multiples of the fundamental frequency (50 or 60 Hz). They are mainly generated by nonlinear loads and power electronic devices, causing waveform distortion. Harmonic distortion is commonly quantified using the Total Harmonic Distortion (THD) index or by analyzing the complete harmonic spectrum.

Oscillatory Transient

An oscillatory transient is a short-duration disturbance in which the voltage or current oscillates with rapidly changing polarity. It is characterized by its predominant frequency, magnitude, and duration, and is commonly produced by switching events, capacitor bank energization, or power electronic commutation.[21]

Impulsive Transient

An impulsive transient is a sudden, non-power-frequency, unidirectional change in voltage or current characterized by a very fast rise and decay time. Lightning strikes are among the most common sources of impulsive transients. In the dataset used in this work, this type of disturbance is represented by the class labeled *Transient*, also referred to as a transient/spike, while the oscillatory transient is considered as a separate class.[21]

Voltage Fluctuation (Flicker)

Voltage fluctuations are systematic or random variations in the voltage envelope that generally remain within 0.9–1.1 p.u. Rapid load-current variations produce these fluctuations, which may cause visible light intensity variations known as flicker.[21]

Harmonics with Sag

Harmonics with sag represent a combined power quality disturbance in which harmonic distortion occurs simultaneously with a reduction in the RMS voltage magnitude. The waveform therefore exhibits both a distorted sinusoidal shape caused by harmonic components and a temporary decrease in voltage amplitude. This type of disturbance may occur when nonlinear loads operate during conditions that produce a voltage sag.

Harmonics with Swell

Harmonics with swell represent a combined disturbance characterized by the simultaneous presence of harmonic distortion and a temporary increase in the RMS voltage magnitude. The resulting waveform contains harmonic components while its voltage amplitude exceeds the normal operating level during the disturbance interval.

Flicker with Sag

Flicker with sag is a combined disturbance in which voltage fluctuations occur together with a temporary reduction in the RMS voltage. Consequently, the waveform presents repetitive or irregular amplitude variations superimposed on a voltage sag condition. This class combines the characteristics of voltage flicker and voltage sag.

Flicker with Swell

Flicker with swell is a combined power quality disturbance characterized by voltage fluctuations occurring simultaneously with a temporary increase in the RMS voltage. The waveform therefore exhibits variations in its voltage envelope while its overall magnitude is higher than the nominal operating level.

Sag with Oscillatory Transient

Sag with oscillatory transient is a combined disturbance in which a temporary reduction in the RMS voltage occurs together with a short-duration oscillatory transient. The resulting waveform presents a decrease in voltage magnitude accompanied by a rapidly decaying oscillatory component. Such disturbances can occur following switching events or faults in the electrical system.

Swell with Oscillatory Transient

Swell with oscillatory transient is a combined disturbance characterized by a temporary increase in the RMS voltage together with a short-duration oscillatory transient. The waveform therefore contains an increase in voltage magnitude accompanied by a rapidly decaying oscillatory component.

Sag with Harmonics

Sag with harmonics is a combined disturbance in which a voltage sag occurs in the presence of harmonic distortion. During the disturbance interval, the RMS voltage decreases below its normal value while the waveform remains distorted by harmonic frequency components.

Swell with Harmonics

Swell with harmonics is a combined disturbance characterized by an increase in the RMS voltage occurring in the presence of harmonic distortion. The waveform simultaneously exhibits an increased voltage magnitude and distortion resulting from harmonic components.

Notch

A voltage notch is a periodic disturbance characterized by a short-duration reduction in the instantaneous voltage waveform. It is generally caused by the normal operation of power electronic converters when current is transferred from one phase to another during commutation. Notching may occur repeatedly within each fundamental cycle and introduces high-frequency components into the voltage waveform.

1.7 Related Works

The choice of deep learning architecture for this thesis was guided by a review of recent literature (2021–2025) on Power Quality Disturbance (PQD) and Power Quality Event (PQE) classification. Three architectures were selected for comparison in this work: CNN, BiLSTM, and Transformer, based on the consistent trends and conclusions reported across three representative studies: Machlev et al. [22] (2021), who introduced an open-source PQD dataset generator together with CNN and BiLSTM reference classifiers; Saber et al. [23] (2024), who proposed a Transformer-based framework for PQE classification under realistic noise and measurement-error conditions; and Albalooshi and Qader [24] (2025), who benchmarked an attention-augmented CNN against BiLSTM, GRU, and ResNet50 baselines. The results and conclusions of these studies motivate the architectural choices adopted in this thesis, as detailed below.

1.7.1 CNN: A Consistently High-Accuracy and Efficient Baseline

Across all three studies, CNN-based architectures were reported as the strongest or among the strongest performers, which motivates including CNN as a baseline architecture in this thesis. Machlev et al. [22] reported that their 1-D CNN reference classifier reached 99.28% testing accuracy without noise and 99.75% with AWGN noise (20–50 dB SNR) across sixteen PQD classes, clearly outperforming their own BiLSTM classifier (96.29%/98.14%) on the same dataset, and converging faster during training. Albalooshi and Qader [24] confirmed this trend on a different ten-class dataset: their multi-scale CNN with channel attention achieved an average testing accuracy of 98.11% with noise and 86.61% without noise (peaking at up to 99.49% on specific configurations), outperforming ResNet50 (90.51%/84.03%), BiLSTM (85.49%/82.03%), and GRU (83.77%/83.65%) baselines under the same experimental protocol. The consistency of this result across two independent studies, different class counts, and different dataset sizes indicates that CNNs offer a favorable accuracy-to-complexity trade-off, and supports their selection as a reference architecture for this thesis, particularly given the constraints of eventual embedded deployment.

1.7.2 BiLSTM: Included as a Sequential-Modeling Baseline Despite Weaker Reported Accuracy

BiLSTM was selected for this thesis primarily as a comparison point representative of recurrent, sequence-aware architectures, rather than for its raw accuracy, a choice directly motivated by the results reported in the literature. In Machlev et al. [22], the BiLSTM reference classifier trailed the CNN by roughly 3 percentage points without noise (96.29% vs. 99.28%) and 1.6 points with noise (98.14% vs. 99.75%), and its confusion matrices showed a specific weakness: sag disturbances were frequently misclassified as interruptions in the noiseless case. Albalooshi and Qader [24] reported a similar gap, with their BiLSTM baseline reaching only 82.03% (no noise) and 85.49% (with noise), underperforming even the GRU baseline in the noiseless setting. Both studies attribute this gap to the added optimization difficulty of the internal recurrence and the compactness of the intermediate representations relative to CNNs (Machlev et al. also note the BiLSTM model is roughly 2.4× smaller in size than the CNN, at 215 KB vs. 520 KB). Despite the accuracy gap, BiLSTM was retained in this thesis to evaluate whether its temporal-dependency modeling offers complementary benefits, for instance for disturbances whose distinguishing features are unevenly distributed in time, and to establish, on our own dataset, whether the accuracy gap reported in the literature persists.

1.7.3 Transformer: Motivated by State-of-the-Art Accuracy Under Realistic Noise Conditions

Transformer-based classification was also considered, based on the results reported by Saber et al. [23], who applied a self-attention Transformer directly to raw voltage signals for PQE classification. Under realistic conditions (noise, DC offset, and amplitude/frequency variation), their model reached 91.43% accuracy, trained on a dataset of 100,000 synthetically generated signals. This dataset size is notably larger than the one used in this thesis, a factor whose impact on Transformer performance is examined in Chapter 3.

1.7.4 Summary and Motivation for the Selected Architectures

Table 1.1 summarizes the accuracy results that motivate the three architectures adopted in this thesis.

Table 1.1: Summary of results motivating the choice of CNN, BiLSTM, and Transformer architectures.

Study	Architecture	Classes	Reported Accuracy
Machlev et al. [22] (2021)	1-D CNN	16	99.28% (no noise) / 99.75% (noise)
Machlev et al. [22] (2021)	BiLSTM	16	96.29% (no noise) / 98.14% (noise)
Albalooshi & Qader [24] (2025)	Multi-scale CNN + attention	10	86.61% (no noise) / 98.11% (noise); up to 99.49% peak
Albalooshi & Qader [24] (2025)	BiLSTM (baseline)	10	82.03% (no noise) / 85.49% (noise)
Saber et al. [23] (2024)	Transformer	10	99.81% (ideal) / 91.43% (noise + DC offset + variation)

Two conclusions emerge from this comparison, each directly motivating an architectural choice in this thesis. First, CNNs consistently deliver the best or near-best accuracy across studies and class counts (up to 99.75%), justifying their inclusion as the primary baseline. Second, BiLSTM, while consistently the weakest performer (a gap of 1.6 to 4.6 percentage points below the corresponding CNN in each study), remains a relevant point of comparison for assessing whether temporal-dependency modeling contributes value on our own signals, and is retained for that reason rather than for its standalone accuracy. The Transformer architecture is discussed separately in Chapter 3, where its performance on the smaller dataset used in this thesis is evaluated.

1.8 Conclusion

This chapter has reviewed the theoretical foundations of power quality disturbance classification, from the characterization of PQDs themselves to the artificial intelligence techniques used to detect and classify them, with particular emphasis on CNN and BiLSTM architectures and their deployment constraints on embedded hardware through TinyML. The review of related works further showed that CNNs offer the most reliable accuracy-efficiency trade-off, that BiLSTM remains relevant but generally weaker point of comparison, and that Transformer-based approaches, while promising, are better suited to large-scale datasets. These conclusions directly motivate the choice of CNN and BiLSTM as the two architectures developed and compared in this work. The following chapter details the methodology adopted to prepare the dataset and implement, train, and deploy these two models.

Chapter 2
Methodology and Implementation

2.1 Introduction

This chapter presents the methodology and implementation adopted for the development of the proposed embedded power quality disturbance classification system. The objective is to describe the complete workflow, from dataset preparation and analysis to the design, training, and embedded deployment of the selected deep learning models.

The chapter first introduces the database used in this study and describes the preprocessing steps applied to prepare the signals for classification. Particular attention is given to the characteristics of the time-series data, descriptive and statistical analyses, data leakage prevention, and the evaluation of different normalization strategies. These investigations are used to justify the preprocessing method retained for model development.

The overall development methodology and the main software tools are then presented, followed by the design of the CNN-1D and Bidirectional Long Short-Term Memory (BiLSTM) architectures. Their main building blocks, activation functions, and selected hyperparameters are described to provide a clear understanding of the proposed classification approaches.

Finally, the chapter addresses the transition from model development to embedded implementation through TinyML deployment on the STM32H743 microcontroller. The embedded inference pipeline is presented together with the acquisition and signal-conditioning validation procedure. Particular attention is also given to the synchronization of signal generation and acquisition, as well as to preliminary noise robustness evaluation, in order to establish a reliable experimental framework before evaluating the complete system under real operating conditions.

2.2 Database Presentation

To evaluate the proposed power quality analysis system and to prepare future machine learning models, the **SEED-PQD-v1 (SEED Power Quality Disturbance Dataset v1)** was selected as the reference public dataset.

This dataset was introduced by Khan et al. in the study entitled *"XPQRS: Expert Power Quality Recognition System for Sensitive Load Applications"*, published in the journal **Measurement (Elsevier, 2023)**. It contains simulated voltage waveforms representing the most common power quality disturbances encountered in electrical power systems and is intended for the development, validation, and benchmarking of power quality disturbance recognition

algorithms.

The dataset is generated according to a **50 Hz** power system and sampled at a frequency of **5 kHz**, resulting in **100 samples per waveform**, which corresponds exactly to **one electrical cycle (20 ms)**. Each signal is normalized within the interval **[-1, 1]**, allowing the dataset to be independent of the nominal voltage magnitude and facilitating its use in machine learning applications.

The dataset contains **17 disturbance classes**, with **1000 signals** for each class, resulting in a total of **17,000 labeled waveforms**. Each disturbance category represents either an individual power quality event or a combination of multiple disturbances that may occur simultaneously in practical power systems.
The considered disturbance classes are:

- Pure Sinusoidal
- Voltage Sag
- Voltage Swell
- Voltage Interruption
- Transient
- Oscillatory Transient
- Harmonics
- Harmonics with Sag
- Harmonics with Swell
- Flicker
- Flicker with Sag
- Flicker with Swell
- Sag with Oscillatory Transient
- Swell with Oscillatory Transient
- Sag with Harmonics
- Swell with Harmonics
- Notch

2.3 Database Treatment

Power quality signals often contain outliers, such as transient signals carrying critical diagnostic information. To preserve these morphological features while meeting the input data format requirements of the model architecture, the data processing workflow must ensure strict numerical stability and reshape the data to match the model's expected input dimensions.

Therefore, several data quality verification tests were performed before the training phase. The dataset was examined for missing values, duplicated samples, and infinite values to ensure data consistency and reliability. In addition, the minimum and maximum values of the signals were analyzed to verify the numerical range of the input features and to detect potential abnormal values that could affect the learning process.

These preprocessing validation steps ensured that the dataset was clean, stable, and suitable for subsequent normalization and model training.

2.3.1 Verification of Class Consistency

During the preliminary inspection of the dataset, an additional verification was performed on the disturbance combinations Harmonics with Sag and Sag with Harmonics, as well as Harmonics with Swell and Swell with Harmonics. Although these class names may suggest duplicated disturbances with different naming conventions, a numerical comparison of the corresponding CSV files confirmed that they are not identical. The compared files exhibit different sample values, indicating that they represent distinct waveform realizations rather than duplicated data. Consequently, all four classes were retained as independent disturbance categories in the dataset.

To ensure the integrity of the dataset, the corresponding CSV files were compared using numerical similarity tests as illustrated in **Listing 2.1**, including exact array comparison (`array_equal`), approximate equality (`allclose`), mean absolute error (MAE), root mean square error (RMSE), maximum absolute error, and Pearson correlation coefficient. The obtained results illustrated in **Listing 2.2** confirmed that the compared files are neither identical nor numerically equivalent, thereby validating that these classes correspond to different signal realizations.

```
if a.shape == b.shape:
    difference = a - b

    print("Files are exactly identical:",
          np.array_equal(a, b))

    print("Files are approximately identical:",
          np.allclose(a, b))

    print("Mean absolute error (MAE):",
          np.mean(np.abs(difference)))

    print("Maximum absolute error:",
```

```
14            np.max(np.abs(difference)))
15
16      print("Root mean square error (RMSE):",
17            np.sqrt(np.mean(difference**2)))
18
19      correlation = np.corrcoef(
20          a.flatten(), b.flatten()
21      )[0, 1]
22
23      print("Overall correlation:", correlation)
24
25  else:
26      print("The files do not have the same dimensions.")
```

Listing 2.1: Python code used to compare two signal files

```
Dimensions : (1000, 100) (1000, 100)
Files are exactly identical: False
Files are approximately identical: False
Mean absolute error (MAE): 1.0566432212622328
Maximum absolute error: 3.7600676665833097
Root mean square error (RMSE): 1.3026554975645108
Overall correlation: -0.015890587970072815
```

Listing 2.2: Output of the signal comparison

During the dataset validation process, duplicate sample detection revealed that only the **Pure Sinusoidal** class contains duplicated waveforms. Further inspection showed that the 1000 samples of this class correspond to identical ideal sinusoidal signals, which is consistent with the absence of any disturbance or stochastic variation. Therefore, these duplicates originate from the dataset design rather than from preprocessing errors.

Although the dataset documentation states that all waveforms are scaled between 1 and 1, a numerical inspection of the downloaded CSV files revealed that several disturbance classes exceed this range. In particular, disturbance combinations involving swell, harmonics, and oscillatory transients reach peak amplitudes of approximately ±2.5. Therefore, the actual dataset does not strictly satisfy the normalization range reported in the accompanying documentation.
The Construction of the labeled Power Quality Disturbance dataset is illustrated in **Figure 2.1**

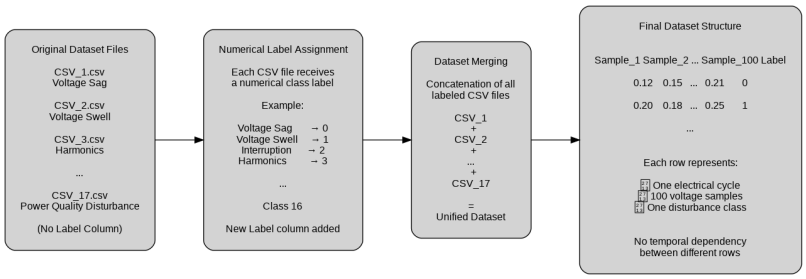


Figure 2.1: Construction of the labeled Power Quality Disturbance dataset from individual CSV files.

2.3.2 Splite test/training and Normalisation

Before training the machine learning models, the dataset was divided into two independent subsets: a training set (80%) and a testing set (20%) using the `train_test_split()` function from the Scikit-learn library. The training set was used to learn the model parameters, while the testing set was reserved exclusively for performance evaluation on previously unseen data.
To ensure a fair and unbiased evaluation, **feature normalization** was performed after the train-test split.
Specifically, the normalization parameters were computed only from the training set using the `fit()` operation and subsequently applied to both the training and testing sets through the transform() operation. This procedure prevents data leakage, where information from the testing set could unintentionally influence the training process, leading to overly optimistic performance estimates. By restricting the normalization process to the training data, the testing set remains completely independent, thereby providing a more realistic assessment of the model's generalization capability on new, unseen signals.

2.4 Time Series Data

A time series is a sequence of observations recorded in chronological order, where each sample represents the value of a variable measured at successive points in time. Unlike conventional tabular data, the temporal ordering of observations is essential because adjacent samples are often correlated. Time series data are commonly encountered in domains such as finance, healthcare, industrial monitoring, telecommunications, and power systems, where signals are acquired at regular sampling intervals.
Depending on the application, time series data can be used for several machine learning tasks:

- **Time Series Forecasting:** Predicts future values of a signal based on its historical observations. Typical applications include electrical load forecasting, weather prediction, and financial market analysis.

- **Time Series Classification (TSC):** Assigns an entire time series to one of several predefined categories. Rather than predicting future samples, the objective is to recognize the underlying pattern represented by the signal. This approach is widely used for fault diagnosis, biomedical signal analysis, and power quality disturbance classification.

- **Time Series Regression:** Estimates one or more continuous numerical variables from a time series, such as predicting the remaining useful life of industrial equipment or estimating a physical quantity from sensor measurements.

- **Time Series Anomaly Detection:** Identifies abnormal or unexpected patterns that deviate significantly from normal operating conditions. This technique is commonly employed for fault detection, predictive maintenance, and cybersecurity.

- **Time Series Segmentation:** Divides a long time series into meaningful temporal regions or events by detecting changes in signal characteristics over time.

In this work, the SEED-PQD-v1 dataset consists of voltage signals sampled at 5 kHz, where each instance corresponds to one electrical cycle (20 ms) composed of 100 consecutive samples. Since the objective is to identify the type of power quality disturbance contained in each signal (e.g., voltage sag, voltage swell, harmonics, or interruption), this study addresses a Time Series Classification (TSC) problem rather than a time series forecasting problem. Each signal is therefore treated as an independent time series associated with a predefined disturbance class.

It should be noted that the different instances (rows) of the dataset are not temporally dependent; each recorded cycle represents an independent observation and does not correspond to a continuous sequence with the previous or following samples. Consequently, the temporal relationships within each individual signal are considered, while no inter-sample dependency exists between different dataset instances.

2.5 Data Analysis for Feature Scaling Selection

Before feeding the raw voltage waveform segments to the classification models (CNN-1D and BiLSTM), a statistical characterization of the dataset was conducted in order to determine the most appropriate scaling method: normalization (Min-Max) or standardization (Z-score), for the input data. This decision was based on an empirical analysis of the data distribution, dispersion, and outlier content, rather than on a default or arbitrary choice, in accordance with standard machine learning preprocessing practice.

According to the dataset documentation, the signal amplitude was originally scaled within the interval [-1, 1].

2.5.1 Descriptive Statistics

Descriptive statistical analysis was performed to characterize the numerical properties and amplitude distribution of the dataset and to verify the consistency of the signals across the 17 power quality disturbance classes.

Per-Class Amplitude Range

A per-class minimum and maximum analysis was first conducted to verify the amplitude range consistancy across disturbance types. While most classes(e.g., Pure sinusuidal, Sag, Interruption) remaind within the nominal [-1, 1] range, several disturbance classes exeeded this boundary, most notably classes combining swell and harmonics content:

Table 2.1: Minimum and maximum values of each class in the SEED-PQD-v1 dataset before normalization.

Class	Min	Max
Pure Sinusoidal	-1.000	1.000
Sag	-1.000	1.000
Swell	-1.800	1.800
Interruption	-1.000	1.000
Transient	-1.039	1.241
Oscillatory Transient	-1.746	1.761
Harmonics	-1.394	1.394
Harmonics with Sag	-1.274	1.270
Harmonics with Swell	-2.186	2.120
Flicker	-1.100	1.098
Flicker with Sag	-1.097	1.093
Flicker with Swell	-1.880	1.892
Sag with Oscillatory Transient	-1.602	1.599
Swell with Oscillatory Transient	-2.460	2.486
Sag with Harmonics	-1.000	1.000
Swell with Harmonics	-1.950	1.950
Notch	-1.000	1.000

This deviation form the nominal ±1 p.u. bound is physically consistent with the difinition of these disturbance classes: a voltage sewell combined with harmonics distortion legitimately produces instantaneous peak amlpitudes exeeding the nominal crest value, since the harmonic components superimpose additively on the fundamental at certain phase angles.

Global Statistics

Considering the dataset in its entirety (all 17 classes, 17,000 signals), **Table 2.1** presents the minimum and maximum amplitude values for each of the 17 disturbance classes. the following global statistics were obtained:

- Global minimum: **-2.460**
- Glabal maximum: **2.486**
- Global mean: **0.0088**
- Global standard diviation: **0.732**

The near-zero global mean confirms the absence of a DC offset bias across the dataset, consistent with the AC nature of the recorded voltage signals.

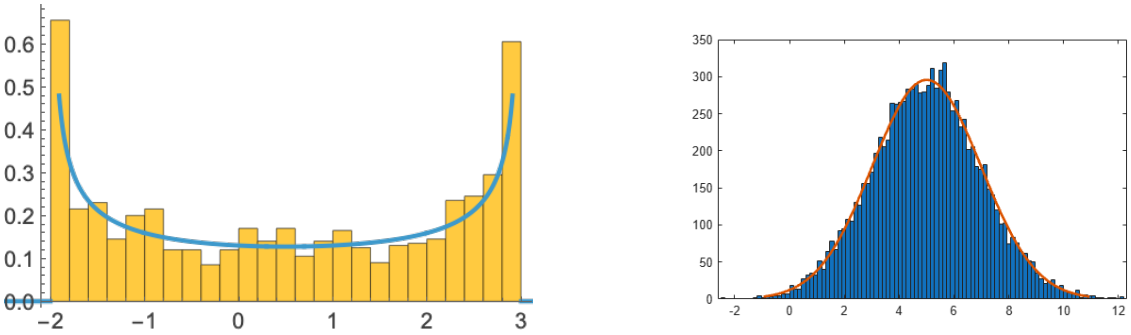
The global standard deviation of 0.732 is slightly higher than the theoretical value of 0.707 expected for a pure unit-amplitude sinusoid. This increase reflects the additional amplitude variability introduced by the different power quality disturbances contained in the dataset.

2.5.2 Distribution Analysis

Distribution analysis was performed to examine the amplitude distribution of the signals and characterize their behavior across the different power quality disturbance classes. Through visual inspection of the amplitude histograms, this analysis provides insight into the statistical characteristics of the dataset and highlights amplitude variations relevant to the subsequent normalization process.

Visual Inspection

Figure 2.2 illustrates the differences between Gaussian and arcsine distributions.



(a) Arcsine distribution.

(b) Gaussian distribution.

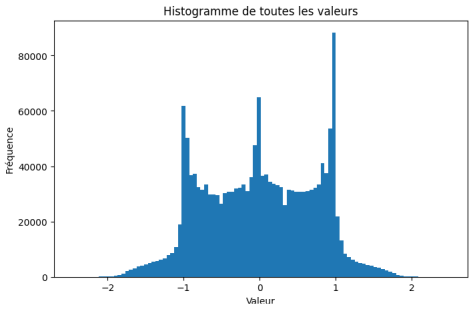
Figure 2.2: Comparison between Gaussian and arcsine distributions.

Figure 2.3 compares the amplitude distribution of the complete dataset with that of the pure sinusoidal class.

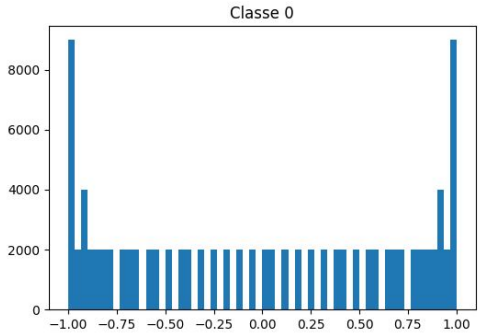
The histogram analysis revealed a trimodal amplitude distribution, as illustrated in **Figure 2.3a**, with pronounced density peaks near -1, 0, and $+1$.

The peaks near ± 1 are consistent with the arcsine-like amplitude distribution of sinusoidal waveforms, as illustrated in **Figure 2.2a**. The central peak may be associated with disturbance classes such as interruptions and voltage sags, which can generate a higher concentration of samples around zero.

Consequently, the trimodal shape reflects the combined amplitude distributions of multiple power quality disturbance classes rather than the distribution of a single sinusoidal waveform, as illustrated in **Figure 2.3b**.



(a) Amplitude histogram of the complete dataset.



(b) Amplitude histogram of the pure sinusoidal class.

Figure 2.3: Comparison of the amplitude distributions of the complete dataset and the pure sinusoidal class.

The histogram of the pure sinusoidal class in **Figure 2.3b** exhibits the characteristic U-shaped amplitude distribution predicted by the arcsine law.
The pronounced peaks near ± 1 occur because a sinusoidal waveform changes more slowly around its maxima and minima than around its zero crossings, leading to a higher concentration of samples near the peak amplitudes.

Normality Testing

The Anderson–Darling test was used to assess whether the data follow a normal distribution. Unlike some other normality tests, it is well suited for large datasets and is particularly sensitive to deviations from normality, particularly at the lower and upper ends of the distribution. [25]

- Statistic: $A^2 = 37.93$
- Critical value (1% significance level): 1.091

Although the Shapiro–Wilk test is one of the most widely used normality tests, it was not applied in this study because many implementations restrict its use to sample sizes of n5000 [25], while the dataset consists of 17,000 observations.

2.5.3 Outlier Analysis Using Boxplots

Prior to model development, the dataset was examined for the presence of outliers, as these observations may affect the statistical characterization of the data and potentially influence the learning process of machine learning algorithms.

An *outlier* is defined as an observation whose value deviates markedly from the majority of the dataset. Such observations may arise from measurement errors, data acquisition faults, or may correspond to legitimate but rare events depending on the application.

To identify potential outliers, the boxplot was employed as a graphical descriptive tool. A boxplot summarizes the distribution of a variable using the five-number summary:

- Minimum
- First quartile (Q_1)

- Median (Q_2)
- Third quartile (Q_3)
- Maximum

The central box represents the interquartile range (IQR), defined as

$$IQR = Q_3 - Q_1. \tag{2.1}$$

According to Tukey's criterion, observations located outside the interval

$$Q_1 - 1.5 \times IQR \leq x \leq Q_3 + 1.5 \times IQR \tag{2.2}$$

are considered potential outliers. Figure 2.4 illustrates the construction of a boxplot and the corresponding outlier detection rule based on the IQR.

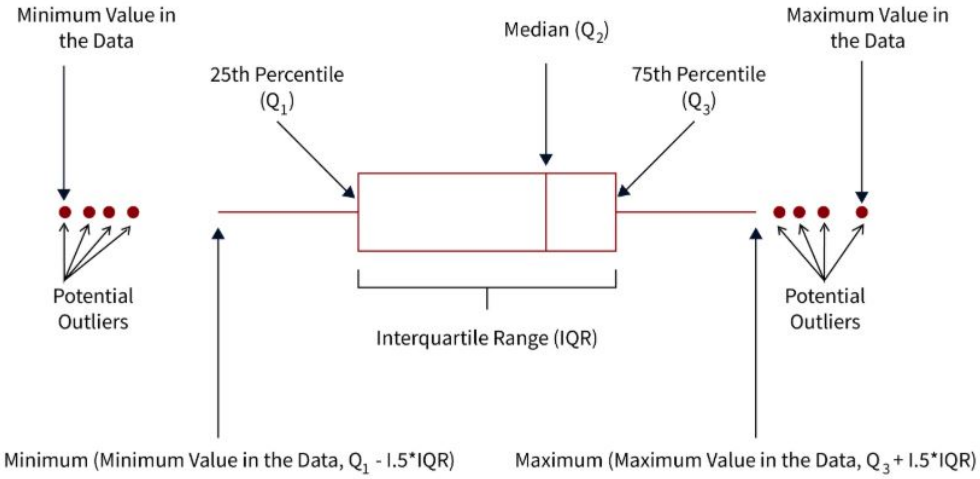


Figure 2.4: Boxplot representation and outlier detection based on Tukey's interquartile range (IQR) criterion.

Observations lying below $Q_1 - 1.5 \times IQR$ or above $Q_3 + 1.5 \times IQR$ are identified as potential outliers.

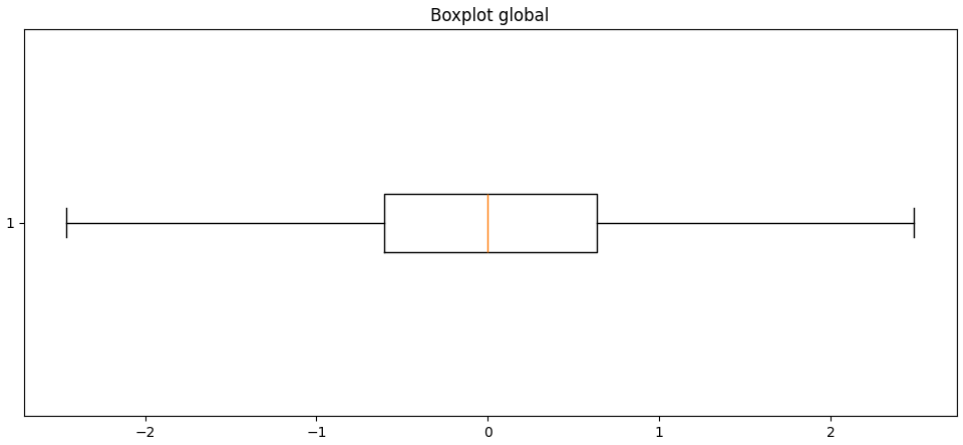


Figure 2.5: Global Boxplot of the used dataset

Figure 2.5 presents the global boxplot obtained by applying Tukey's interquartile range (IQR) criterion to the entire dataset. The boxplot provides a compact summary of the data distribution through the median, quartiles, and whiskers representing the non-outlying observations. No individual data points appear outside the whiskers, indicating that no observations were classified as outliers according to the criterion

$$Q_1 - 1.5 \times IQR \leq x \leq Q_3 + 1.5 \times IQR.$$

This result confirms that the dataset does not contain statistically significant extreme values at the global scale. The absence of outliers suggests that the acquired signals are internally consistent and do not include abnormal observations that would require removal prior to the preprocessing and model training stages.

2.5.4 Statistical Analysis and Justification of the Selected Normalization Method

Since all input features correspond to successive samples of the same voltage waveform, they share the same physical meaning and measurement unit. Therefore, feature standardization to equalize variances is not required.
Moreover, the objective of the present work is to preserve the absolute amplitude relationships of the electrical waveforms, as the magnitude of voltage variations constitutes the primary discriminative information for distinguishing power quality disturbances such as voltage sags, swells, and harmonic events. Standardization rescales each feature according to its mean and standard deviation, thereby modifying its original amplitude scale.
In contrast, Min-Max normalization preserves the relative amplitude relationships while mapping all values into a bounded interval, making it more suitable for neural network training and for maintaining the physical interpretation of the measured signals.

2.6 Data Leakage in Machine Learning

Data leakage is a critical issue in machine learning that occurs when information unavailable during model deployment is inadvertently introduced into the training process. As a result, the model gains access to information that would not be accessible in real-world applications, leading to overly optimistic performance estimates and poor generalization on unseen data. Preventing data leakage is therefore an essential step in the development of reliable and unbiased machine learning models.

Data leakage may occur at different stages of the preprocessing pipeline. In this work, particular attention is given to three common sources of leakage: [26]

- Handling missing values.
- Preprocessing of time-series data.
- Feature scaling through normalization or standardization.

2.6.1 Missing Values

Data leakage may occur when missing values are imputed before splitting the dataset into training and testing subsets. In such a case, the statistics used for imputation (e.g., mean, median, or mode) are computed using information from the entire dataset, including the test set. [26]
Consequently, information from unseen samples influences the training process, resulting in an overly optimistic estimation of the model's performance. To prevent this issue, missing-value imputation must be performed using only the training set, after which the same transformation is applied to the test set. [26]

2.6.2 Time-Series Data

Data leakage may occur in time-series datasets when the chronological order of observations is ignored during data partitioning. Randomly shuffling samples before splitting the dataset allows the model to learn information from future observations that would not be available during real-world prediction, resulting in overly optimistic performance estimates. To avoid this issue, time-series datasets must be divided chronologically, ensuring that the training set contains only past observations while the test set contains future observations.
In the present work, although each sample corresponds to a voltage waveform acquired over time, the dataset is not a continuous time series. Instead, it consists of independent waveform samples representing different power quality disturbance events. Each sample is acquired independently and has no chronological relationship with the preceding or subsequent samples.

Consequently, randomly partitioning the dataset into training and testing subsets does not introduce temporal data leakage, provided that each waveform is assigned entirely to either the training or the testing set.

2.6.3 Normalization and Standardization

Feature scaling may also introduce data leakage if normalization or standardization parameters are computed using the entire dataset before the train-test split. In this situation, the minimum and maximum values (for Min-Max normalization) or the mean and standard deviation (for standardization) are influenced by the test samples, allowing information from the test set to be incorporated into the training process.

To avoid this issue, the scaling parameters must be estimated exclusively from the training data. The fitted transformation is then applied unchanged to the validation and test sets, ensuring that no information from unseen samples is leaked into the training stage.

2.6.4 Training and Testing Split and normalisation

To prevent data leakage, feature scaling was performed only after partitioning the dataset into training and testing subsets. The normalization parameters were estimated exclusively from the training set.
Once the normalization parameters had been determined, the same transformation was directly applied to the testing set without recalculating new minimum or maximum values.

Consequently, the testing samples were transformed using exactly the same scaling function learned from the training data.

It should be noted that the testing data are not necessarily mapped strictly into the interval $[-1, 1]$. If a testing sample contains values outside the range observed during training, the corresponding normalized values may become slightly smaller than -1 or greater than 1. This behavior is expected and confirms that the testing set is processed using the scaling parameters learned from the training set only.

2.7 Normalization Strategies and Their Evaluation

To determine the most appropriate preprocessing technique for the SEED-PQD-v1 dataset, three normalization strategies were investigated. Each method was evaluated not only according to the classification accuracy obtained by the CNN-1D model, but also by verifying the integrity of the generated datasets. The following subsections describe each approach, its implementation, the observed limitations, and the reasons for selecting the final method.

2.7.1 First Approach: Feature-wise Min-Max Normalization

The first normalization strategy was based on computing the minimum and maximum value independently for each feature (sampling instant). Since each signal consists of 100 samples, the algorithm computed 100 minimum values and 100 maximum values from the training set. Each feature was then normalized independently using the following equation:

$$x' = \frac{x - \min(x_j)}{\max(x_j) - \min(x_j)} \times 2 - 1$$

where:

- x is the original sample,
- $\min(x_j)$ and $\max(x_j)$ are the minimum and maximum values of the j^{th} feature computed over the training set,
- x' is the normalized sample.

In this approach, each column of the dataset is scaled independently to the interval $[-1, 1]$. Consequently, the normalization parameters differ from one sampling position to another.

Although this method is mathematically correct and commonly used for tabular datasets, it is less appropriate for power quality time-series signals. Each column corresponds to a specific sampling instant of the waveform rather than an independent physical variable. Therefore, scaling each sampling instant separately modifies the relative amplitude relationships between different temporal positions of the signal.

The effect of the feature-wise normalization was first verified using a pure sinusoidal waveform. The normalized waveform exhibited a noticeable distortion compared with the original signal, demonstrating that the preprocessing modified the signal amplitude profile as illustrated in **Figure 2.6**. Because the same normalization procedure is applied identically to every sample in the dataset, the same transformation is expected to affect all disturbance signals.

Consequently, amplitude-related information, which is essential for distinguishing disturbances such as voltage sag, swell, and interruption, may be partially altered, leading to a degradation in the CNN-1D classification performance.

As a consequence, part of the waveform characteristics that are useful for distinguishing power quality disturbances may be altered. Experimental results confirmed this limitation, as the CNN-1D classifier achieved an accuracy of only **84%**, which was significantly lower than the performance obtained without such distortion.

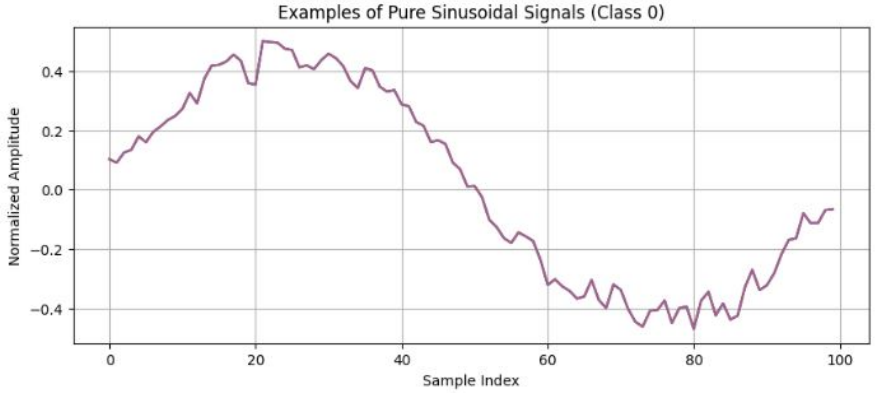


Figure 2.6: Effect of the first approach normalization on a pure sinusoidal signal.

2.7.2 Second Approach: Global Normalization with Incorrect Dataset Reconstruction

The second strategy aimed to normalize the entire training dataset using a single global minimum and a single global maximum computed from the training set. The normalization equation was

$$x' = \frac{x - X_{\min}}{X_{\max} - X_{\min}} \times 2 - 1$$

where $X_{\min}$ and $X_{\max}$ represent the global minimum and maximum values of the training dataset.

This normalization principle is appropriate because every sample is transformed using the same scaling parameters, preserving the relative amplitude relationships among all signal samples.

However, during the reconstruction of the training and testing datasets, an implementation error occurred. The labels were reassigned using

```
train_df["label"] = y_train.reset_index(drop=True)
```

After the dataset was split using `train_test_split()`, both `X_train` and `y_train` retained the same original row indices inherited from the source dataset. For example,

$$X_{\text{train}} = \begin{array}{c|ccc} \text{Index} & x_1 & x_2 & \cdots \\ \hline 6162 & \cdots & \cdots & \cdots \\ 11072 & \cdots & \cdots & \cdots \\ 15624 & \cdots & \cdots & \cdots \end{array}$$

$$y_{\text{train}} = \begin{array}{c|c} \text{Index} & \text{Label} \\ \hline 6162 & 0 \\ 11072 & 3 \\ 15624 & 1 \end{array}$$

However, after applying

```
y_train.reset_index(drop=True)
```

the label vector became

$$y_{\text{train}} = \begin{array}{c|c} \text{Index} & \text{Label} \\ \hline 0 & 0 \\ 1 & 3 \\ 2 & 1 \end{array}$$

while `X_train` still retained its original indices (6162, 11072, 15624, ...).
When executing

```
train_df["label"] = y_train.reset_index(drop=True)
```

Pandas aligns the data according to the row indices rather than their positions. Therefore, it attempts to match the label with index 0 to the feature vector having index 0. Since `train_df` contains rows indexed by 6162, 11072, 15624, and so on, no matching indices are found. Consequently, Pandas fills the unmatched rows with missing values (NaN), producing an invalid dataset.
The verification procedure revealed

- 2717 missing labels in the training dataset,
- 2723 missing labels in the testing dataset.

Because the class labels constitute the ground truth for supervised learning, the generated datasets were considered invalid.

Consequently, this normalization experiment was discarded since the classification results could not be considered reliable.

2.7.3 Third Approach: Correct Global Min-Max Normalization

After identifying the indexing issue, the datasets were reconstructed correctly by resetting the indices of both the normalized feature matrix and the label vector before concatenation.

The normalization remained identical to the previous approach, using one global minimum and one global maximum computed exclusively from **the training dataset**:

$$X_{\text{min}} = \min(X_{\text{train}})$$

$$X_{\text{max}} = \max(X_{\text{train}})$$

followed by

$$x' = \frac{x - X_{\min}}{X_{\max} - X_{\min}} \times 2 - 1$$

The same normalization parameters were subsequently applied to the testing dataset.
This procedure prevents data leakage because no information from the testing set is used during preprocessing.

Unlike the previous implementation, both the normalized features and their corresponding labels shared identical indices before being merged. Consequently, no missing values were observed in either dataset.
The verification step confirmed

- Training dataset: 0 NaN values.
- Testing dataset: 0 NaN values.

Furthermore, this preprocessing strategy preserved the global amplitude relationships of the power quality signals as illustrated in **Figure 2.7**, while ensuring a consistent scaling over the entire dataset.

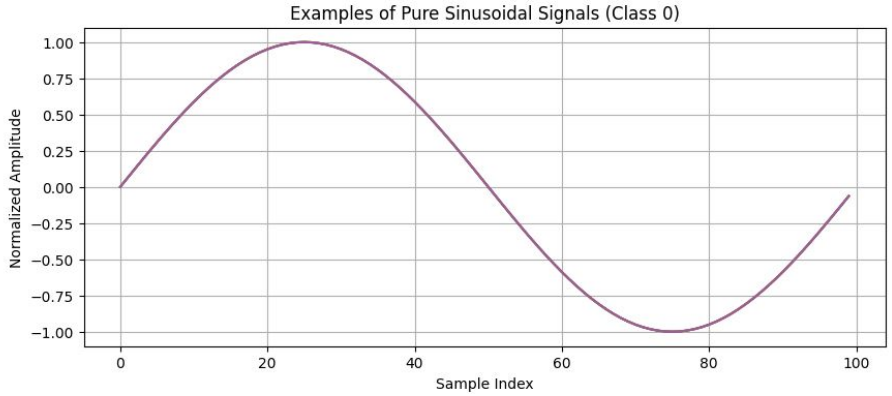


Figure 2.7: Normalized pure sinusoidal

Experimental evaluation demonstrated that this approach achieved the highest classification performance, reaching an accuracy of approximately **94%**.

2.7.4 Fourth approach: RobustScaler normalization

An additional experiment was conducted using RobustScaler to investigate the effect of an alternative normalization strategy on the BiLSTM model. This choice was motivated by the fact that the article [22], which also employs a BiLSTM architecture, does not explicitly specify the normalization technique used, while another study in article [27] applys a Transformer-based model reports the use of RobustScaler for PQD signals. Based on this, RobustScaler was tested in order to evaluate its suitability for the present dataset.

The authors of article [27] justify this choice by the presence of outliers and high-amplitude transient components in PQ signals, for which a robust normalization approach based on the median and interquartile range can reduce the influence of extreme values on the scaling process.

It is also worth noting that the dataset used in [27] was considerably larger, with 10,000 randomly generated samples for each signal category. The dataset was synthetically generated using MATLAB with a fundamental frequency of 50 Hz, a sampling frequency of 3915 Hz. Thus, the effectiveness of RobustScaler reported in [27] was obtained under experimental conditions and a dataset configuration that differed from those considered in the present study.

Following this approach, the same BiLSTM architecture was trained using the RobustScaler-normalized dataset. However, the resulting model achieved an accuracy of **88.7%**, which was lower than the performance obtained with Min–Max normalization. In addition to the classification results, the normalized signals were visually examined.

When the signals corresponding to pure sinusoidal waveforms were plotted after RobustScaler normalization, noticeable distortions of the waveform were observed as illustrated in **Figure 2.8**, whereas the same signals normalized using the Min–Max method preserved a smooth sinusoidal shape as illustrated in **Figure 2.7**. This observation was particularly relevant for the present application, since **the morphological characteristics of the electrical waveforms constitute important information for distinguishing between different PQD classes**.

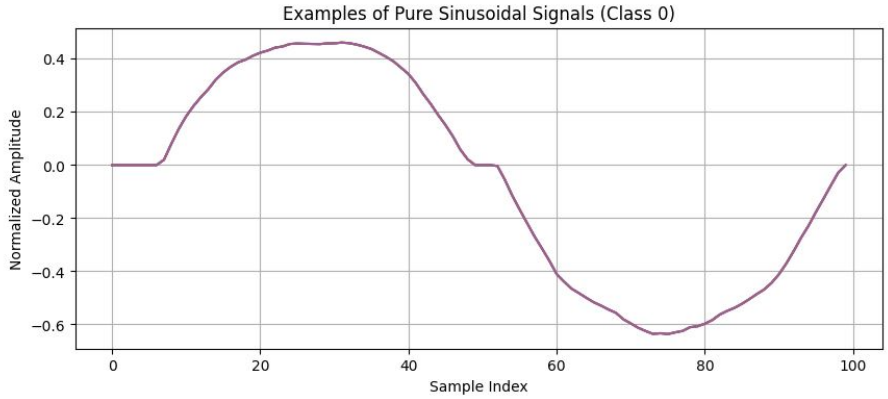


Figure 2.8: Effect of RobustScaler normalization on a pure sinusoidal signal.

Based on these observations, the BiLSTM model was subsequently trained using the Min–Max-normalized dataset under the same experimental conditions. This configuration achieved an accuracy of **92.2%**, representing an improvement of 3.3 percentage points compared with the RobustScaler-based configuration.

2.7.5 Comparison and Selection of the Normalization Strategy

The four investigated approaches were compared in terms of signal integrity, dataset validity, and classification performance. **Table 2.2** summarizes the experimental findings obtained with the CNN-1D and BiLSTM models.

Table 2.2: Comparison of the investigated normalization approaches.

Approach	**Signal Integrity**	**Dataset Validity**	**Classification Accuracy**
Feature-wise Min-Max	Relative amplitude relationships altered	Valid	84% (CNN-1D)
Global Min-Max with incorrect reconstruction	Relative amplitude relationships preserved by scaling	Invalid (missing labels)	Not reliable
Correct Global Min-Max	Relative amplitude relationships preserved	Valid (0 NaN values)	$\approx$ 94% (CNN-1D) 92.2% (BiLSTM)
RobustScaler	Waveform distortion observed	No invalidity reported	88.7% (BiLSTM)

The comparative evaluation highlights the importance of selecting a normalization strategy that preserves the characteristics of power quality signals while ensuring dataset integrity. The feature-wise Min-Max approach altered the relative amplitude relationships within the waveforms, resulting in reduced CNN-1D classification accuracy. The second approach employed an appropriate global scaling principle but produced invalid datasets due to an indexing error during label reconstruction.

The corrected global Min-Max approach addressed these limitations by preserving relative amplitude relationships and ensuring correct alignment between features and labels. It achieved approximately 94% accuracy with the CNN-1D model.

Furthermore, the additional experiment conducted with the BiLSTM architecture demonstrated that global Min-Max normalization achieved 92.2% accuracy, compared with 88.7% using RobustScaler, representing an improvement of 3.5 percentage points. Visual inspection also revealed waveform distortion following the tested RobustScaler implementation, whereas global Min-Max normalization preserved the sinusoidal waveform shape.

Therefore, considering dataset integrity, waveform preservation, and the classification results obtained with both architectures, the corrected global Min-Max approach was selected as the final preprocessing method.

Therefore, this normalization strategy was adopted for all subsequent experiments presented in this work.

2.8 Software tools and API

This section presents the overall methodology followed in this work, from the development and training of the artificial intelligence model to its deployment and validation on the target STM32 microcontroller. It also introduces **the different frameworks**, **software environments**, and **tools** used throughout the process, including Python and Google Colab for

model development and training, and STM32CubeMX with X-CUBE-AI for model conversion, code generation, and embedded deployment. The complete workflow is illustrated in **Figure 2.9**.

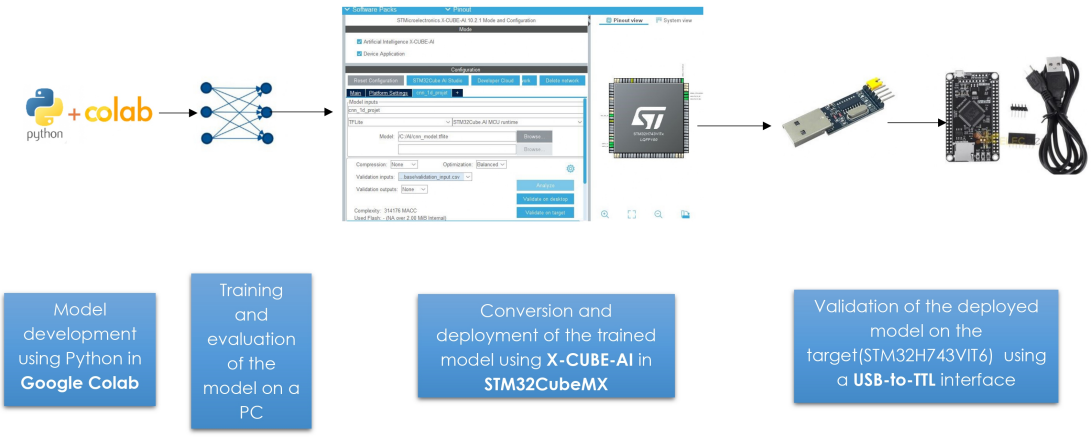


Figure 2.9: Deployment and validation workflow of the AI model on the STM32 microcontroller.

2.8.1 Python

Python was used as the main development environment for data preprocessing, deep learning model development, training, and evaluation. The implementation relied mainly on **TensorFlow** and **Keras** for model construction and training, while **Scikit-learn** was used for preprocessing and evaluation. **NumPy, Pandas,** and **Matplotlib** were also used for numerical processing, dataset manipulation, and visualization.

Python enabled rapid experimentation with different model architectures and hyperparameters before selecting the final models for deployment on the STM32 microcontroller.

2.8.2 Google Colaboratory

To run our notebook we will use a tool called Colaboratory, or *Colab* for short. Colab is made by Google, and it provides an online environment for running AI models. It's provided for free as a tool to encourage research and development in machine learning. [20]

Traditionally, you needed to create a notebook on your own computer. This required installing a lot of dependencies, such as Python libraries, which can be a headache. It was also difficult to share the resulting notebook with other people, since they might have different versions of the dependencies, meaning the notebook might not run as expected. In addition, machine learning can be computationally intensive, so training models might be slow on your development computer. [20]

Colab allows you to run notebooks on Google's powerful hardware, at zero cost. You can edit and view your notebooks from any web browser, and you can share them with other people, who are guaranteed to get the same results when they run them. You can even

configure Colab to run your code on specially accelerated hardware that can perform training more quickly than a normal computer. [20]

We will be creating our model inside of a Colab notebook, which permits us to do awesome things to visualize our data during development. This includes displaying graphs that show our model's accuracy and convergence.

2.8.3 TensorFlow and Keras

TensorFlow is a set of tools for building, training, evaluating, and deploying machine learning models. Originally developed at Google, TensorFlow is now an open source project built and maintained by thousands of contributors across the world. It is the most popular and widely used framework for machine learning. Most developers interact with TensorFlow via its Python library. [20]

TensorFlow does many different things. In this chapter we'll use **Keras**, TensorFlow's *high-level API* that makes it easy to build and train deep learning networks. We'll also use **TensorFlow Lite**, a set of tools for deploying TensorFlow models to mobile and embedded devices, to run our model on-device. [20]

2.8.4 X-Cube-AI

X-CUBE-AI is an STM32Cube MX expansion package as illustrated in **Figure 2.10** that enables the integration of trained artificial intelligence models into STM32 microcontroller applications. Automatically converting pretrained artificial intelligence algorithms into C code and generates the required libraries for integration into the STM32CubeMX project. During this process, the neural network is optimized to reduce its memory footprint and computational requirements while preserving its functionality. The generated AI library can then be executed by the STM32 microcontroller, allowing the trained model to perform inference directly on the embedded target.

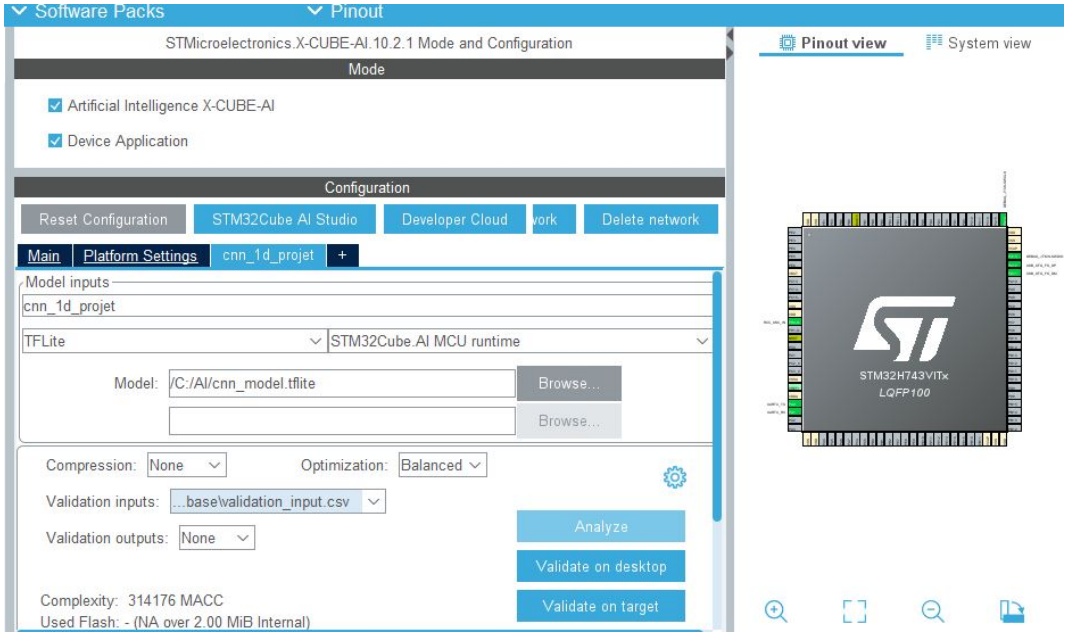


Figure 2.10: X-Cube-AI package in STM32Cube MX

2.8.5 Validation on target

Before performing the on-target validation, the trained CNN model was analyzed using STM32CubeMX and the X-CUBE-AI software package as illustrated in **Figure 2.11**. This analysis provides an estimation of the computational and memory requirements of the neural network and verifies how the model can be mapped onto the target STM32 microcontroller. The analysis also decomposes the neural network into its supported computational operators and reports the Flash and RAM requirements of the generated AI application.

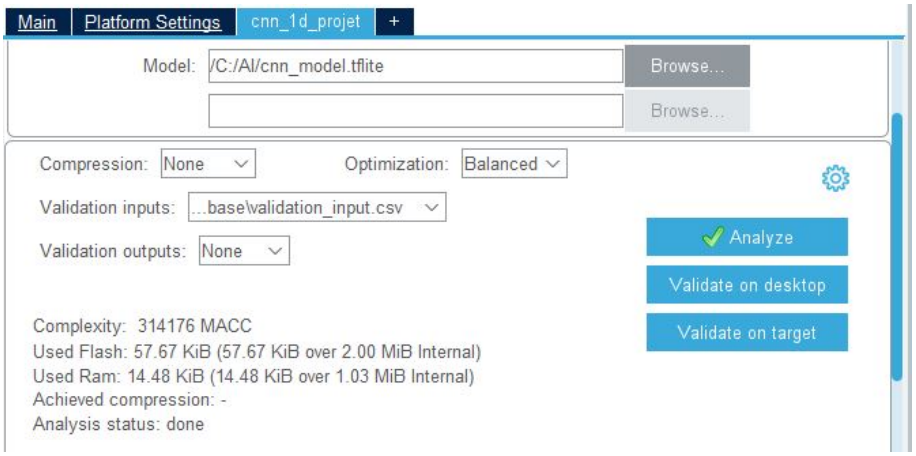


Figure 2.11: X-Cube-AI anlysis before performing the on-target validation

The main results obtained from the model analysis are summarized as follows:

- **Model format:** The trained network was converted into a TensorFlow Lite (`.tflite`) model, which was then imported into STM32CubeMX for analysis.
- **Network operations:** X-CUBE-AI decomposed the neural network into its elementary operations, including convolution, pooling, nonlinear activation, element-wise multiplication, and dense operations, representing the main computational workload required for one inference.
- **Flash memory consumption:** The estimated Flash memory consumption.
- **RAM consumption:** The estimated RAM requirements.
- **Resource utilization:** The estimated resource utilization of the generated AI application.

2.9 CNN1D Model Architecture

The first network included in this study is a one-dimensional Convolutional Neural Network (CNN-1D), which is widely used for processing sequential data such as time-series signals. The proposed network consists of three 1-D convolutional layers with Rectified Linear Unit (ReLU) activation functions, followed by batch normalization and max-pooling layers to extract hierarchical features and improve the generalization capability of the model. A Global Average Pooling layer is then applied, followed by a dropout layer to reduce overfitting. The architecture of the network is shown in **Figure 2.12**.

Finally, a fully connected layer with a Softmax activation function is used for the classification of the 17 power quality disturbance classes. The architecture of the proposed CNN-1D model is presented in **Figure 2.12**. The convolutional layers use 16, 32, and 64 filters with kernel sizes of 7, 5, and 3, respectively, while the max-pooling layers use a pool size of 2.

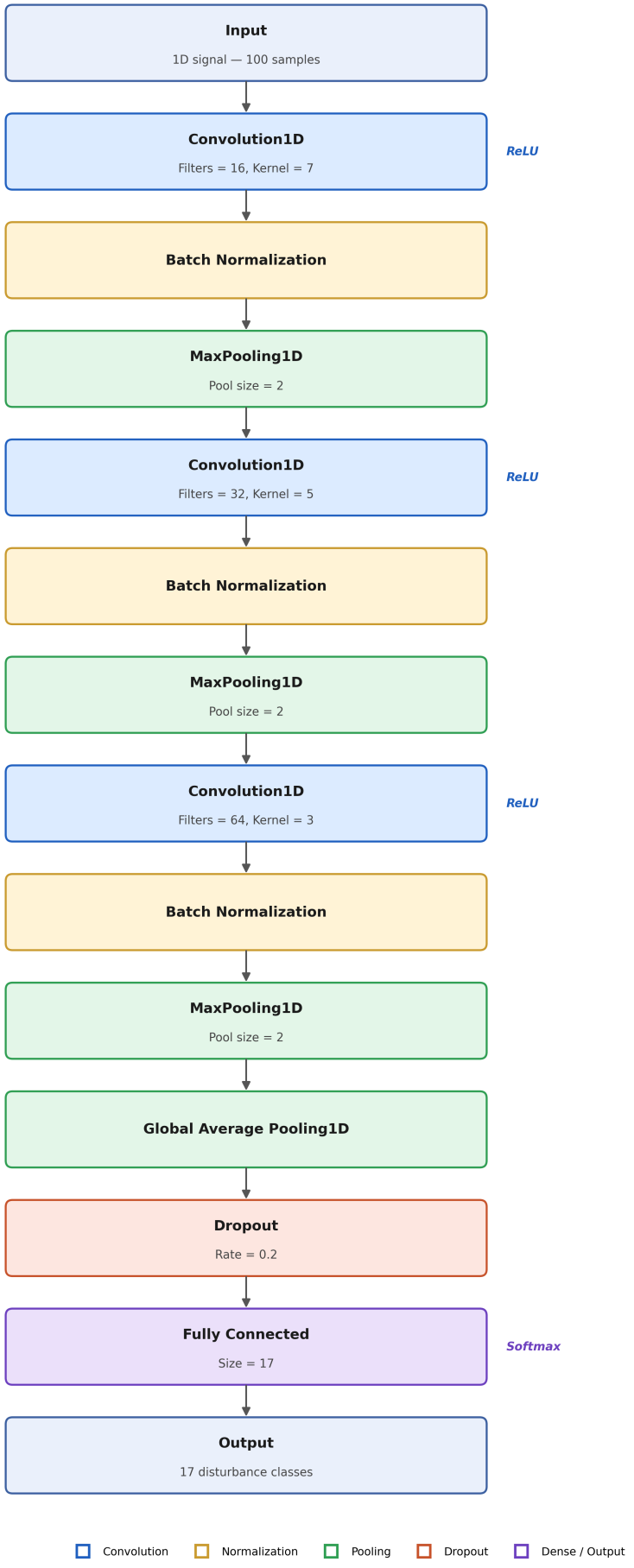

Input
1D signal — 100 samples
Convolution1D
Filters = 16, Kernel = 7
ReLU
Batch Normalization
MaxPooling1D
Pool size = 2
Convolution1D
Filters = 32, Kernel = 5
ReLU
Batch Normalization
MaxPooling1D
Pool size = 2
Convolution1D
Filters = 64, Kernel = 3
ReLU
Batch Normalization
MaxPooling1D
Pool size = 2
Global Average Pooling1D
Dropout
Rate = 0.2
Fully Connected
Size = 17
Softmax
Output
17 disturbance classes
Convolution
Normalization
Pooling
Dropout
Dense / Output

Figure 2.12: Architecture of the Proposed CNN-1D Model

2.9.1 Understanding the building blocks of the CNN-1D network

A convolutional neural network (*CNN*) is built by stacking simple processing "blocks". Each block transforms the signal in a precise way. This section describes, without mathematical formalism, the role of each block used in the proposed architecture (**Table 2.3**).

Convolution (Convolution1D)

This is a small "pattern detector" (a filter) that slides along the signal, from beginning to end, and responds strongly when it recognizes a particular shape (a sharp rise, an oscillation, a cut, etc.). Each filter learns to recognize a different pattern: the more filters used, the more different patterns the network can recognize at once.

Normalization (Batch Normalization)

After each convolution layer, the resulting values can become very large or very small, which makes training unstable. Normalization rescales these values to a reasonable, consistent range — similar to recalibrating measurements taken with different instruments before comparing them. This speeds up and stabilizes training.

Pooling (MaxPooling1D)

This layer summarizes the information by keeping only the strongest value in each small portion of the signal, then "shrinks" the length of the signal (here, by half). This lets the network focus on what matters, reduces the amount of computation, and makes the network less sensitive to a small time shift in the signal.

Dropout

During training only, this layer randomly "switches off" part of the neurons at each iteration. This prevents the network from "memorizing" the training examples by heart, and forces it to learn features that are genuinely useful and generalize well to new signals.

Dense layer (Fully Connected)

This is the final decision layer: it connects all the features extracted by the previous layers to produce, for each possible class, a score reflecting the probability that the signal belongs to that class.

Output (Softmax)

This last step turns the scores from the dense layer into probabilities that add up to 100 %. The class with the highest probability is taken as the network's final decision (for example: "Sag", "Harmonics", "Normal", etc.).

Table 2.3: Nature of each block: standard CNN component or design choice specific to this architecture

Block	Status	What is a choice
Convolution	Standard block, present in almost every CNN	Number of filters (16/32/64) and kernel size (7/5/3)
Normalization	Very common standard technique, but not mandatory	Its presence and its position in the network
Pooling	Standard block, present in the vast majority of CNNs	Size of the reduction window (here 2)
Dropout	Standard regularization technique	The rate (here 0.2) and where it is placed
Dense (Fully Connected)	Standard final classification block	Its size, here fixed at 17 (number of classes in the problem)
Softmax	Standard for multi-class, mutually exclusive classification	No real choice: imposed as soon as the classes are mutually exclusive

In summary **Table 2.5** summarizes that the **types of layers** used (convolution, normalization, pooling, dropout, dense, softmax) are **standard** components, found in almost every CNN architecture regardless of the application domain. On the other hand, **the number of layers, their order, the number of filters, the kernel sizes, the pooling window sizes, and the dropout rate** are **hyperparameters specific to the proposed architecture**: they were chosen (or empirically tuned) based on the nature of the signal being processed and the number of classes to distinguish, and are not dictated by any universal rule.

2.9.2 Hyperparameter Selection

The hyperparameters of the proposed CNN-1D model were selected experimentally to achieve a good trade-off between classification performance and computational complexity for future deployment on the STM32H743 microcontroller. During the model development, only one hyperparameter was modified at a time, while the remaining parameters were kept unchanged. After each modification, the model was trained and its performance was evaluated using the training and validation accuracy and loss curves. This iterative process made it possible to identify the combination of hyperparameters that provided the best classification performance while maintaining a lightweight architecture suitable for embedded implementation.

2.10 Bidirectional Long Short-Term Memory Architecture

The proposed BiLSTM architecture consists of three stacked Bidirectional Long Short-Term Memory (BiLSTM) layers followed by a fully connected output layer. The BiLSTM layers are used to extract and learn temporal features from the input power quality signals in both forward and backward directions. The extracted features are then fed to a fully connected layer with a Softmax activation function to classify the signals into the 17 power

quality disturbance classes. The architecture of the proposed BiLSTM model is presented in **Figure 2.13**.

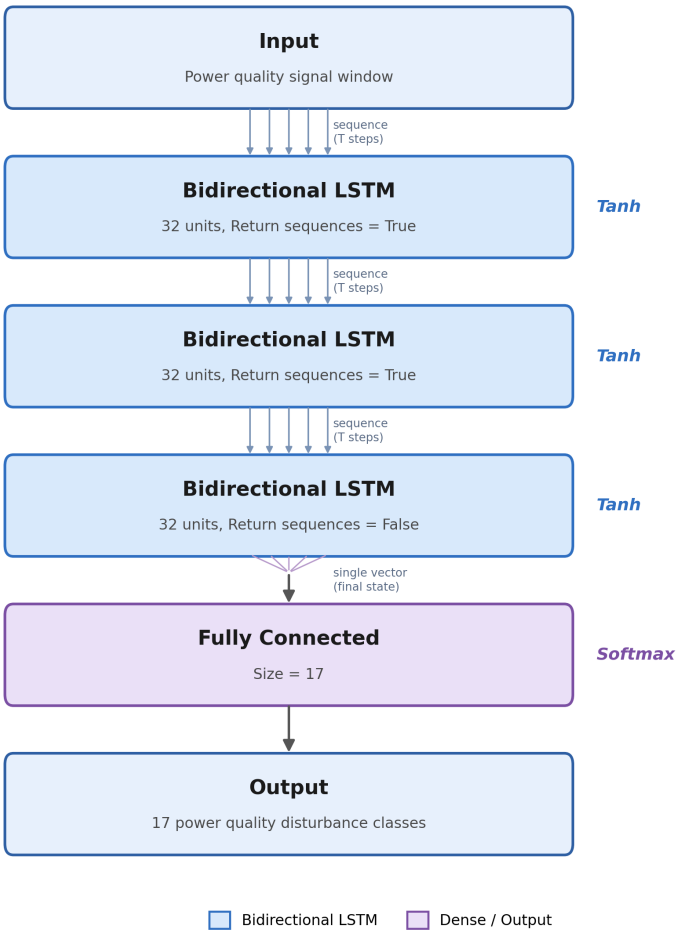


Figure 2.13: Architecture of the proposed BiLSTM Model

2.10.1 Understanding the building blocks of the BiLSTM network

A recurrent neural network based on Long Short-Term Memory (*LSTM*) units is built to read a signal step by step, keeping a "memory" of what it has already seen so it can capture how the signal evolves over time. This section describes, without mathematical formalism, the role of each block used in the proposed BiLSTM architecture (**Table 2.4**).

Bidirectional LSTM

A standard LSTM reads the signal only in one direction (from the first sample to the last), so at any given instant it only "knows" what happened before. A *bidirectional* LSTM runs two LSTMs in parallel on the same input (one reading the signal forward, the other reading it backward), and combines their outputs [18]. This allows the network, at every point in the signal, to take into account both what came before *and* what comes after, which is particularly useful for detecting disturbances whose signature depends on the whole shape of the event (e.g. the moment a sag begins can only be recognized with certainty once its end has also been "seen").

Tanh activation

Inside each LSTM unit, the hyperbolic tangent function is used to regulate the values passed from one time step to the next, keeping them bounded within a fixed range. This is the standard activation used in LSTM cells; it helps the network preserve stable, well-scaled information over long sequences instead of letting values grow or shrink uncontrollably.

Return sequences

At every time step, an LSTM layer produces an output. The *return sequences* parameter decides whether the layer passes on **all** of these time-step outputs, or only the **last** one. When set to `True`, the full sequence is preserved so that the next layer can, in turn, read it step by step. When set to `False`, only the final output is kept, condensing the entire sequence into a single vector, this is required just before the classification stage, since a dense layer expects a fixed-size input, not a sequence.

Dense layer (Fully Connected)

This is the final decision layer: it connects the single feature vector produced by the last BiLSTM layer to produce, for each possible class, a score reflecting the probability that the signal belongs to that class.

Output (Softmax)

This last step turns the scores from the dense layer into probabilities that add up to 100 %. The class with the highest probability is taken as the network's final decision (for example: "Sag", "Harmonics", "Normal", etc.).

Table 2.4: Nature of each block: standard recurrent-network component or design choice specific to this architecture

Block	Status	What is a choice
Bidirectional LSTM	Standard block, present in most sequence-modeling networks	Number of layers (here 3) and number of units per layer (here 32)
Tanh activation	Standard, default activation inside an LSTM cell	No real choice: imposed by the LSTM cell's internal design
Return sequences	Standard mechanism for stacking recurrent layers	Which layers propagate the full sequence (True) and which return only the final state (False)
Dense (Fully Connected)	Standard final classification block	Its size, here fixed at 17 (number of classes in the problem)
Softmax	Standard for multi-class, mutually exclusive classification	No real choice: imposed as soon as the classes are mutually exclusive

In summary: the **types of layers** used (bidirectional LSTM, dense, softmax) and the **activation functions** (Tanh, Softmax) are **standard** components, found in almost every BiLSTM-based classification network regardless of the application domain. On the other hand, **the number of stacked BiLSTM layers, the number of units per layer, and which layers return full sequences versus a single final state** are **hyperparameters specific to**

the proposed architecture: they were chosen (or empirically tuned) based on the nature of the signal being processed and the number of classes to distinguish, and are not dictated by any universal rule.

2.10.2 Hyperparameter Selection

The hyperparameters of the proposed BiLSTM model were selected experimentally to achieve a good trade-off between classification performance and computational complexity. During the model development, only one hyperparameter was modified at a time, while the remaining parameters were kept unchanged. After each modification, the model was trained and its performance was evaluated using the training and validation accuracy and loss curves. This iterative process made it possible to identify the combination of hyperparameters that provided the best classification performance while keeping the architecture reasonably lightweight.

2.10.3 Activation functions used in the proposed models

Both proposed architectures rely on a small set of standard activation functions, each chosen for a specific role within the network rather than arbitrarily. This section explains why each one was used and to what extent it represents a "standard" choice versus one imposed by the architecture itself.

ReLU (CNN-1D convolutional layers)

The Rectified Linear Unit (ReLU) has become the default activation function for convolutional layers because it is computationally simple and helps avoid the vanishing gradient problem that affects saturating functions such as sigmoid or tanh in deep networks. It is not strictly mandatory, alternatives such as LeakyReLU, ELU, or Swish could technically be used, but ReLU remains the overwhelming standard choice for CNNs unless there is a specific reason to do otherwise.

Tanh and Sigmoid (BiLSTM layers)

Inside each LSTM cell, the hyperbolic tangent (tanh) function is used to regulate the candidate and hidden states, keeping their values bounded between -1 and 1, while the sigmoid function is used internally in the forget, input, and output gates to produce values between 0 and 1 that act as "opening percentages" controlling how much information is kept or discarded at each time step. Unlike ReLU in CNNs, this is not simply a common convention: tanh and sigmoid are part of the original LSTM cell design (Hochreiter & Schmidhuber, 1997) and are used by default in standard deep learning frameworks such as Keras/TensorFlow. Replacing them would essentially change the internal behavior of the LSTM cell itself, so they are best described as imposed by the architecture rather than a tunable choice. [28]

Softmax (output layer, both models)

The softmax activation is used in the final dense layer of both the CNN-1D and the BiLSTM models to convert the raw output scores into probabilities that sum to 100 %, allowing the class with the highest probability to be selected as the network's decision. This choice is not tied to the CNN or BiLSTM architecture itself, but to the nature of the classification problem: since the 17 power quality disturbance classes are mutually

exclusive (a signal belongs to exactly one class), softmax is the standard activation for this type of multi-class, single-label classification. Had the problem been multi-label instead, a sigmoid output would have been used in place of softmax. [16]

Table 2.5: Activation functions used in the proposed CNN-1D and BiLSTM models

Activation	Used in	Role	Status
ReLU	CNN-1D convolutional layers	Introduces non-linearity while avoiding vanishing gradients	Standard default, but replaceable
Tanh / Sigmoid	BiLSTM cells (internal)	Regulate cell state and gate values	Imposed by the LSTM cell design
Softmax	Output layer (both models)	Converts scores into class probabilities	Standard, imposed by the multi-class nature of the problem

2.11 TinyML Deployment

Once training is complete, the model is exported in its native Keras format (`model.save()`) and then converted to TensorFlow Lite using the standard `TFLiteConverter`.

2.11.1 Deployment on STM32H743

The resulting `.tflite` file is imported into STM32CubeMX through the `X-CUBE-AI` package, with the target platform set to `stm32h743VIT6`. Before any code generation, the `analyze` step is run: it parses the network graph statically, without executing it, and reports:

- the memory footprint the model will require once deployed — Flash for the weights plus the AI runtime library, and RAM for the activation buffers;
- the computational cost of the network, expressed in MACC (multiply-accumulate operations), broken down layer by layer;
- the relative contribution of each layer to both the compute load and the ROM footprint, which highlights which layers dominate the inference cost.

No accuracy figures are produced at this stage, since no data is actually run through the network, `analyze` is a static estimation step, used to confirm the model is compatible with the target's Flash/RAM budget before moving on to functional validation.

2.11.2 Embedded Inference Validation Pipeline

validation is carried out in two successive stages.

Validate on desktop

The generated C model is first exercised on the host PC rather than on the microcontroller. Input samples are fed in parallel to the original TFLite model and to the C code generated by ST Edge AI Core, running side by side on the same machine. Their outputs are compared sample by sample, yielding equivalence metrics (accuracy, RMSE, MAE, cosine similarity, confusion matrix). As illustrated in **Table 2.6**, this step does not validate real hardware behavior; it only confirms that the C code generated from the `.tflite` file is numerically faithful to the original model, before any embedded execution takes place.

Table 2.6: Model validation results

Output	acc	rmse	mae	l2r	mean	std	nse	cos
X-cross #1	100.00%	0.00000000	0.00000000	0.00000000	-0.000000	0.000000	1.000000	1.000000

Embedded Inference Validation on Target

The firmware is then flashed onto the STM32H743 board, and a CSV file containing the actual input samples is streamed to the microcontroller so that inference runs directly on the target hardware, using the real embedded runtime and memory layout rather than a desktop simulation.

This step requires a genuine bidirectional data channel between the host PC and the microcontroller: input vectors have to be streamed in, and the corresponding predictions read back, for every sample in the CSV file. The probe used in this setup is a stand-alone ST-Link/V2, which only implements SWD, a debug and programming interface, not a general-purpose data link. Unlike Nucleo boards, whose integrated ST-Link exposes a virtual COM port alongside SWD over the same USB cable, this custom board's probe provides no such channel. A **CH340 USB-to-UART adapter**, wired to the microcontroller's UART pins, is therefore used to expose the serial port that STM32Cube AI Studio needs for the validate-on-target data exchange, while the ST-Link remains dedicated exclusively to flashing and debugging.

The output of this stage reports the same set of metrics as the desktop validation, but now measured on real hardware: memory actually occupied on the target (Flash and RAM, matching or refining the static `analyze` estimate), classification accuracy, the confusion matrix over all classes, and the per-sample inference latency.

2.12 Acquisition Validation and Signal Conditioning

After the deployment of the CNN on the STM32H743VIT6, the acquisition chain was implemented and validated to provide input signals compatible with the model requirements. The validation focused on the sampling configuration, the DAC–ADC acquisition process, and the synchronization between signal generation and acquisition.

2.12.1 Sampling Frequency and Acquisition Window

The acquisition system was configured with a sampling frequency of 5 kHz. Since the signals considered in the study have a fundamental frequency of 50 Hz, one electrical cycle corresponds to:

$$T = \frac{1}{50} = 20 \text{ ms}.$$

At a sampling frequency of 5 kHz, this duration corresponds to:

$$N = 5000 \times 0.02 = 100 \text{ samples}.$$

Therefore, the acquisition of 100 consecutive samples provides exactly one 50 Hz electrical cycle, which is consistent with the input dimension of the proposed CNN.

2.12.2 DAC–ADC Signal Generation and Acquisition

To validate the acquisition chain before using externally acquired electrical signals, the DAC of the STM32H743VIT6 was used to generate a controlled test waveform. The generated signal was then acquired through the ADC.

The ADC was configured with a 16-bit resolution and operated in DMA mode. A timer trigger was used to ensure a sampling frequency of 5 kHz. The resulting 100 samples were then provided to the signal-processing and classification pipeline.

This configuration establishes the following acquisition sequence:

DAC signal generation → ADC acquisition → DMA transfer → 100 samples → CNN input

2.12.3 Final Acquisition Validation

The final acquisition configuration was verified by monitoring the acquired samples and confirming that a complete 100-sample signal window could be obtained at the required sampling frequency. This validation confirmed the compatibility between the acquisition configuration and the CNN input requirements.

The acquisition stage therefore provides the interface between the physical signal acquisition system and the embedded classification pipeline. Further evaluation using non-ideal and real-world signals is considered separately to assess the robustness and generalization of the trained model.

2.13 Noise Robustness Evaluation Prior to Real-World Deployment

2.13.1 Initial Approach

The initial development and validation of the proposed models were conducted using an idealized PQD dataset composed of clean signals. This choice provided a controlled environment for first validating the architecture and its ability to distinguish the different PQD classes without introducing additional acquisition-related variables.

The use of an ideal dataset was also motivated by the difficulty of obtaining a sufficiently large real-world PQD database containing all the characteristics required by the proposed system. In particular, no suitable dataset was identified that simultaneously provided the required PQD classes together with realistic sensor noise, DC offset, ADC-related effects,

amplitude and gain variations, signal distortions, and other imperfections associated with a practical measurement chain.

Furthermore, constructing such a database experimentally would require the acquisition and controlled generation of high-power electrical signals under different PQD conditions. This introduces important hardware, safety, and experimental constraints. Therefore, the clean dataset was retained as the initial basis for model development and performance evaluation.

2.13.2 Identification of the Problem

The models achieved a high accuracy on the clean test dataset. Although this results confirm that the models can effectively distinguish the considered PQD classes under controlled conditions, it does not by itself demonstrate robustness under practical acquisition conditions.

A potential *domain shift* or *distribution shift* exists between the signals used during training and the signals that may be encountered during deployment. The training data represent an idealized signal distribution, whereas signals acquired through a sensor and ADC-based measurement chain may contain several additional disturbances and non-idealities.

Consequently, high classification accuracy on clean data cannot be considered sufficient evidence of reliable real-world generalization. This raised the following validation question:

Can the models maintain thier classification performance when the input signals are affected by disturbances that were not present during training?

Before proceeding directly to hardware acquisition, a preliminary robustness test was therefore introduced.

2.13.3 Preliminary Robustness Test

Since publicly available noisy PQD datasets were not directly compatible with the experimental configuration of the proposed system, a noisy version of the original dataset was constructed by introducing controlled Gaussian noise.

Using the original dataset rather than replacing it with another database ensured that the main signal characteristics remained unchanged, including:

- signal frequency;
- sampling frequency;
- number of samples per signal;
- signal duration;
- PQD class distribution.

Therefore, the experiment was designed to isolate the effect of noise as much as possible, without simultaneously introducing differences in sampling frequency, signal duration, or input structure.

Three Signal-to-Noise Ratio (SNR) levels were considered:

- 40 dB;

- 30 dB;
- 20 dB.

The CNN model was evaluated on the clean and noisy datasets without modifying its architecture or learned parameters.

2.14 Conclusion

This chapter established the methodology used to develop the proposed power quality disturbance classification system, covering the main steps from data preparation and model development to embedded implementation on the STM32H743. The adopted approach provides a consistent framework for training the CNN-1D and BiLSTM models and integrating them into the target embedded platform. The acquisition and preliminary robustness tests also provided the basis for evaluating the system under conditions closer to practical operation by adding gaussien noise.

Based on this implementation, the next chapter presents and discusses the obtained results, with particular emphasis on the performance and embedded characteristics of the proposed models.

Chapter 3
Results and Discussion

3.1 Introduction

This chapter presents and discusses the results obtained from the proposed power quality disturbance classification system. The evaluation covers both the machine learning performance of the CNN-1D and BiLSTM models and their implementation on the STM32H743 microcontroller.

The models are first evaluated using training results, classification metrics, and confusion matrices to assess their ability to distinguish between the different disturbance classes. Their embedded deployment is then analyzed in terms of computational and memory requirements. Finally, the limitations encountered during the development process and the experimental choices made to address them are discussed.

3.2 Dataset Verification Results

The dataset verification procedure described in Chapter 2 confirmed the numerical consistency of the prepared SEED-PQD-v1 dataset before model training. The inspection showed that the dataset did not contain problematic missing or infinite values, while the duplicated samples identified in the Pure Sinusoidal class were attributed to the original construction of the dataset rather than to an error introduced during preprocessing.

The amplitude-range analysis also confirmed that several disturbance classes exceed the nominal interval of $[-1, 1]$, particularly classes involving swell, harmonics, and oscillatory transients. These values were therefore retained because they represent characteristics of the original disturbance signals rather than invalid numerical observations.

Regarding preprocessing, the comparison of the normalization strategies presented in Chapter 2 showed that the selected **global Min–Max normalization** preserved the relative amplitude relationships and the main morphological characteristics of the electrical waveforms. In contrast to the feature-wise Min–Max approach, which altered the waveform profile, the global transformation applies the same scaling parameters to all sampling instants and therefore maintains the amplitude information required to distinguish disturbances such as voltage sag, swell, interruption, harmonics, and transient events.

The verification results therefore support the use of the selected global Min–Max normalization as the preprocessing method for the subsequent CNN-1D and BiLSTM experiments.

The classification performances obtained with the resulting datasets are presented in the following sections.

3.3 Training Results

This section presents and discusses the training results obtained for the two proposed models.

3.3.1 Convolutional Neural Network Results

The initial CNN-1D model achieved a classification accuracy of 94%. Several experiments were subsequently conducted to improve the classification performance while maintaining a lightweight architecture suitable for future deployment on the STM32H743 microcontroller. Different architectural and training parameters were modified and evaluated individually and in combination.

The first experiments focused on the feature aggregation and convolutional layers. The Flatten layer was replaced by a Global Average Pooling 1D layer, while different convolutional configurations were also investigated, including the use of SeparableConv1D layers. The combination of these modifications was then evaluated. The architecture was subsequently simplified by reducing the number of convolutional blocks and adjusting the number of filters. The number of neurons in the fully connected layer was also reduced from 64 to 32. Finally, the learning rate was optimized to improve the training process and classification performance.

Among the different configurations investigated, the architecture combining conventional Conv1D layers with a Global Average Pooling 1D layer provided the best performance, the Early Stopping mechanism restored the model weights corresponding to epoch **67**, which achieved an accuracy of 96.9%. Therefore, this configuration was selected as the final CNN-1D model. The final convolutional architecture, including the number of filters and kernel sizes, is presented in **Figure 2.12**. The improvement from 94% to 96.9% (The best validation accuracy during training) demonstrates that the architectural modifications, particularly the replacement of the Flatten layer by Global Average Pooling 1D, contributed to improving the classification performance while maintaining a relatively lightweight model.

3.3.2 Bidirectional Long Short-Term Memory Results

During the initial training phase, the Early Stopping mechanism restored the model weights corresponding to epoch 146, which achieved an accuracy of 94.4% (The best training configuration). In order to reduce the training time and the number of epochs required for convergence, the training procedure was subsequently modified. With the modified training strategy, the best model weights were restored from epoch 118, resulting in an accuracy of 91.1%. Although the modified configuration reduced the number of training epochs, it also led to a decrease in classification performance.

The relatively large number of epochs required by the BiLSTM model illustrated in **Table 3.1**, can be explained by the nature of the architecture and the sequential characteristics

of the input signals. The model must progressively learn temporal dependencies through several stacked recurrent layers, which can require more training iterations than a simpler architecture. In addition, the use of a learning-rate reduction strategy allows the optimizer to progressively refine the model parameters when the validation loss reaches a stage of stability, which can extend the convergence process.

Table 3.1: BiLSTM Training Results

Number of Epochs	Accuracy
146	94.4%
118	91.1%

The Early Stopping mechanism is therefore useful for retaining the best-performing weights while preventing unnecessary additional training.
In Keras, `restore_best_weights=True` restores the weights corresponding to the best value of the monitored metric, while `ReduceLROnPlateau` reduces the learning rate when the monitored metric stops improving.

3.4 Performance Comparison

This section presents a comparative evaluation of the CNN-1D and BiLSTM models, focusing on their overall performance, classification metrics, and confusion matrices to identify their strengths and limitations in power quality disturbance classification.

3.4.1 Overall Model Performance

Before analyzing the class-wise classification metrics, the main performance results of the investigated architectures are summarized in **Table 3.2**. The reported values correspond to different stages of the evaluation process and should therefore be distinguished between validation, test, and embedded deployment results.

Table 3.2: Summary of the main model performance and embedded deployment results.

Model	Best Validation	Test	STM32 Feasibility
CNN-1D	96.9%	95.5%	Feasible
BiLSTM	94.4%	93%	Limited by RAM
Transformer	68.7%	–	Not pursued

3.4.2 Classification Metrics

Classification metrics provide a quantitative evaluation of the model's ability to correctly identify the different Power Quality Disturbance classes. In this study, three main metrics are considered: **precision**, **recall**, and **F1-score**. Since the dataset contains 17 classes with an equal number of test samples, these metrics provide a detailed evaluation of the classification performance for each disturbance type.

Precision

Precision measures the proportion of samples correctly classified as a given class among all samples predicted as that class. It therefore indicates the model's ability to avoid false positive predictions. A high precision means that when the model predicts a particular PQD class, the prediction is likely to be correct. Precision is defined as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.1}$$

where:

- TP: The number of true positive predictions
- FP: The number of false positive predictions.

Recall

Recall measures the proportion of samples belonging to a given class that are correctly identified by the model. It therefore evaluates the model's ability to detect all samples belonging to a particular PQD class. A low recall indicates that some samples of the considered class are incorrectly classified as other disturbance types. Recall is defined as:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.2}$$

where:

- FN: represents the number of false negative predictions.

F1-score

The F1-score provides a combined measure of precision and recall. It is particularly useful for evaluating whether the classifier maintains a good balance between avoiding false positive predictions and correctly detecting samples of each class. The F1-score is calculated as the harmonic mean of precision and recall:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.3}$$

A value close to 1 indicates strong classification performance, whereas a value closer to 0 indicates poor performance.

3.4.3 Interpretation of the Classification Results for the CNN-1D Model

The classification report for the CNN-1D model in **Table 3.3**, contains 3,400 test samples, with 200 samples for each of the 17 PQD classes. The model achieves an overall accuracy of 95%. The macro-averaged precision, recall, and F1-score are 0.96, 0.95, and 0.96, respectively. The weighted averages present the same values because all classes contain an equal number of test samples. These results indicate that the classifier provides a strong and relatively balanced performance across the different disturbance categories.

Table 3.3: Classification performance for each PQD class.

PQD Class	Precision	Recall	F1-score	Support
Pure Sinusoidal	0.98	1.00	0.99	200
Sag	0.99	0.97	0.98	200
Swell	0.99	0.99	0.99	200
Interruption	0.98	1.00	0.99	200
Transient	0.88	0.98	0.93	200
Oscillatory Transient	0.81	0.90	0.85	200
Harmonics	0.98	1.00	0.99	200
Harmonics with Sag	0.97	0.94	0.95	200
Harmonics with Swell	0.99	0.99	0.99	200
Flicker	0.94	0.98	0.96	200
Flicker with Sag	0.96	0.94	0.95	200
Flicker with Swell	0.96	0.98	0.97	200
Sag with Oscillatory Transient	0.98	0.89	0.93	200
Swell with Oscillatory Transient	0.99	0.85	0.91	200
Sag with Harmonics	0.92	0.92	0.92	200
Swell with Harmonics	0.97	0.95	0.96	200
Notch	0.98	0.95	0.97	200
Macro Average	**0.96**	**0.95**	**0.96**	**3400**
Weighted Average	**0.96**	**0.95**	**0.96**	**3400**

3.4.4 Overall Analysis of the CNN-1D Model

Overall, the results demonstrate that the model performs very well across the 17 PQD classes, achieving 95% overall accuracy and a macro F1-score of 0.96.
The highest performance is observed for:

- Pure Sinusoidal
- Swell
- Interruption
- Harmonics
- Harmonics with Swell

All of which achieve F1-scores of approximately 0.99.
In contrast Oscillatory Transient presents the greatest classification difficulty, with an F1-score of 0.85.
The relatively lower recall values observed for:

- Sag with Oscillatory Transient: (0.89)
- Swell with Oscillatory Transient: (0.85)

Also indicate difficulties in recognizing some combined disturbances involving **oscillatory transients**.

Therefore, the classification results suggest that the model is highly effective for most PQD categories, while combined and oscillatory disturbances remain the main sources of classification errors. The confusion matrix analysis presented in the following section can provide further information about the specific classes that are being confused with each other.

3.4.5 Interpretation of the Classification Results for the BiLSTM Model

The classification report for the BiLSTM model in **Table 3.4**, contains 3,400 test samples, with 200 samples for each of the 17 PQD classes. The model achieves an overall accuracy of 93%. The macro-averaged precision, recall, and F1-score are 0.94, 0.93, and 0.93, respectively. The weighted averages present the same values because all classes contain an equal number of test samples. These results indicate that the BiLSTM model provides good and relatively balanced classification performance across the different disturbance categories.

Table 3.4: Classification performance for each PQD class using the BiLSTM model.

PQD Class	Precision	Recall	F1-score	Support
Pure Sinusoidal	1.00	1.00	1.00	200
Sag	0.99	0.99	0.99	200
Swell	0.97	1.00	0.99	200
Interruption	0.98	1.00	0.99	200
Transient	0.80	0.98	0.88	200
Oscillatory Transient	0.91	0.89	0.90	200
Harmonics	0.93	0.99	0.96	200
Harmonics with Sag	0.91	0.82	0.87	200
Harmonics with Swell	0.98	0.93	0.95	200
Flicker	0.97	1.00	0.99	200
Flicker with Sag	0.96	0.90	0.93	200
Flicker with Swell	0.96	0.93	0.94	200
Sag with Oscillatory Transient	0.94	0.91	0.92	200
Swell with Oscillatory Transient	0.97	0.90	0.93	200
Sag with Harmonics	0.81	0.86	0.83	200
Swell with Harmonics	0.86	0.90	0.88	200
Notch	0.97	0.89	0.93	200
Macro Average	**0.94**	**0.93**	**0.93**	**3400**
Weighted Average	**0.94**	**0.93**	**0.93**	**3400**

Macro average: gives every class exactly the same importance, regardless of how many samples that class contains.

$$\text{Macro Average F1-score} = \frac{F1_1 + F1_2 + \cdots + F1_{17}}{17}$$

Weighted average: considers the number of samples in each class.

$$\text{Weighted Average F1-score} = \frac{N_1 F1_1 + N_2 F1_2 + \cdots + N_{17} F1_{17}}{N_1 + N_2 + \cdots + N_{17}}$$

3.4.6 Overall Analysis of the BiLSTM Model

Overall, the results demonstrate that the BiLSTM model performs well across the 17 PQD classes, achieving 93% overall accuracy and a macro F1-score of 0.93.
The highest performance is observed for:

- Pure Sinusoidal
- Sag
- Swell
- Interruption
- Flicker

Which achieve F1-scores between 0.99 and 1.00. These results indicate that: **The BiLSTM model can effectively recognize several individual PQD patterns.**

In contrast, the lowest performance is observed for:

- Sag with Harmonics: Which achieves an F1-score of 0.83.
- Harmonics with Sag: (0.87)
- Transient: (0.88)
- Swell with Harmonics: (0.88)

The Transient class: presents a particularly low precision of 0.80 despite its high recall of 0.98, indicating that the model successfully detects most actual transient signals but also incorrectly assigns some samples from other classes to this category.

The results also show that combined disturbances involving harmonics present greater classification difficulty. In particular, Sag with Harmonics and Harmonics with Sag obtain lower F1-scores than most individual disturbance classes.

Similarly, the oscillatory-transient-related classes show relatively lower recall values, including:

- Oscillatory Transient (0.89)
- Sag with Oscillatory Transient (0.91)
- Swell with Oscillatory Transient (0.90)

This suggests that the combination of multiple disturbance characteristics makes some PQD categories more difficult for the BiLSTM model to distinguish.

Therefore, the classification results indicate that the BiLSTM model provides strong performance for most individual PQD categories, while combined disturbances, particularly those involving harmonics and oscillatory transients, remain the main sources of classification errors. The confusion matrix analysis presented in the following section can provide further information about the specific classes that are being confused with each other.

3.5 Confusion Matrix Comparison

While the classification report shows the performance of each class, the confusion matrices in **Figure 3.1** and **Figure 3.2** shows which classes are most often confused by the CNN-1D and the BiLSTM model. The off-diagonal values reveal three main groups of errors, with each group showing a different reason why some classes are difficult for the model to distinguish.

3.5.1 Confusion Matrix Analysis of the BiLSTM Model

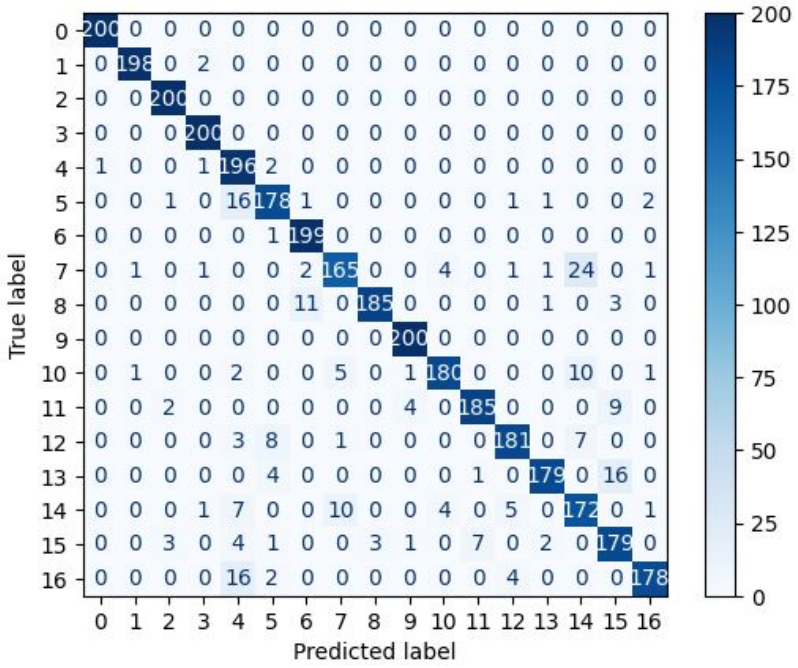


Figure 3.1: BiLSTM Model Confusion Matrix

Confusion Between Harmonics-Sag Classes

The main confusion observed in the confusion matrix is between the two combined disturbance classes:

- Harmonics with Sag: 24 samples are incorrectly classified as Sag with Harmonics.
- Sag with Harmonics: 10 samples are incorrectly classified as Harmonics with Sag.

These two classes have similar characteristics because both contain harmonics and sag disturbances. Therefore, the BiLSTM model may have difficulty distinguishing which disturbance is dominant in some samples.

The confusion is not completely symmetric, since 24 samples of Harmonics with Sag are classified as Sag with Harmonics, while only 10 samples are classified in the opposite direction. This indicates that the model tends to predict Sag with Harmonics more often when the characteristics of the two disturbances are similar.

This result shows that combined disturbances are more difficult to classify than individual disturbances. The presence of both sag and harmonic components in the same signal makes the distinction between these two classes more difficult for the BiLSTM model.

Confusion Between Transient-Related Classes

A second group of errors is observed among the transient-related classes. The main confusions are:

- Oscillatory Transient: 16 samples are incorrectly classified as Transient.
- Notch: 16 samples are incorrectly classified as Transient.
- Swell with Oscillatory Transient: 4 samples are incorrectly classified as Oscillatory Transient.

These classes can be difficult to distinguish because they contain short-duration changes in the electrical signal. The main observations are:

- Oscillatory Transient and Notch are sometimes classified as the general Transient class.
- The model may detect the presence of a short disturbance but have difficulty identifying its exact type.
- Oscillatory Transient contains an oscillating component, while Notch has a short dip in the waveform.
- When these specific characteristics are not clearly detected, the model tends to classify the signal as Transient.

These results indicate that short-duration disturbances are more difficult for the BiLSTM model to distinguish. In particular, the model can detect the presence of a transient event, but it may have difficulty identifying the specific type of transient disturbance.

Confusion Between Swell and Flicker-Related Classes

A third group of errors is observed among the combined classes involving Swell and Flicker. The main confusions are:

- Swell with Oscillatory Transient: 16 samples are incorrectly classified as Swell with Harmonicse.
- Flicker with Swell: 9 samples are incorrectly classified as Swell with Harmonics.
- Swell with Harmonics: 7 samples are incorrectly classified as Flicker with Swell.
- Harmonics with Swell: 11 samples are incorrectly classified as Harmonics.

These errors can be explained by the similarity between some of the combined disturbance signals. In particular:

- The Swell component can be clearly detected because it affects the overall amplitude of the signal.
- However, the additional disturbance, such as harmonics, flicker, or oscillatory transient, can be more difficult to identify.
- As a result, the model sometimes correctly recognizes the main Swell characteristic but incorrectly identifies the additional disturbance.
- This is particularly visible in the confusion between Swell with Harmonics and Flicker with Swell.

These results indicate that the BiLSTM model can recognize the main amplitude variation caused by Swell, but it may have more difficulty identifying the additional disturbance present in the same signal. Therefore, combined classes with multiple disturbance characteristics remain more challenging for the model.

Errors Related to Sag with Harmonics

The *Sag with Harmonics* class shows errors with several different classes rather than with only one class. The main misclassifications are:

- Sag with Harmonics: 10 samples are incorrectly classified as Harmonics with Sag.
- Sag with Harmonics: 7 samples are incorrectly classified as Transient.
- Sag with Harmonics: 5 samples are incorrectly classified as Sag with Oscillatory Transient.
- Sag with Harmonics: 4 samples are incorrectly classified as Flicker with Sag.
- Sag with Oscillatory Transient: 7 samples are incorrectly classified as Sag with Harmonics.

These errors can be explained by the fact that *Sag with Harmonics* is a combined disturbance containing more than one characteristic. In particular:

- It can be confused with *Harmonics with Sag* because both classes contain sag and harmonic components.
- It can also be confused with *Sag with Oscillatory Transient* because both contain a sag component.
- The confusion with *Flicker with Sag* can also be related to the presence of the common sag characteristic.
- Some samples are classified as *Transient*, showing that the additional changes in the waveform can sometimes be interpreted as a transient disturbance.

Therefore, the *Sag with Harmonics* class represents another challenging category for the BiLSTM model. Its errors are distributed among several combined disturbance classes, especially those that share the same sag or harmonic characteristics.

Synthesis

Overall, the confusion matrix shows that most classification errors are concentrated in a few groups of similar or combined disturbances. The main observations are:

- Confusion between *Harmonics with Sag* and *Sag with Harmonics*.
- Confusion between short-duration transient-related classes, particularly *Oscillatory Transient*, *Notch*, and *Transient*.
- Confusion between some combined classes involving *Swell*, *Flicker*, and other disturbances.
- *Sag with Harmonics* is also confused with several other combined disturbance classes.

These results show that the BiLSTM model performs very well for several individual disturbance classes, while combined disturbances are generally more difficult to distinguish. However, the confusion matrix alone cannot determine the exact reason for every misclassified sample. Some errors may be related to the similarity between classes, while others may result from variations in the signals, model limitations, or other factors.

Therefore, the main conclusion is that the remaining errors are mainly observed in classes with similar or combined signal characteristics, rather than being evenly distributed across all PQD classes.

3.5.2 Confusion Matrix Analysis of the CNN-1D Model

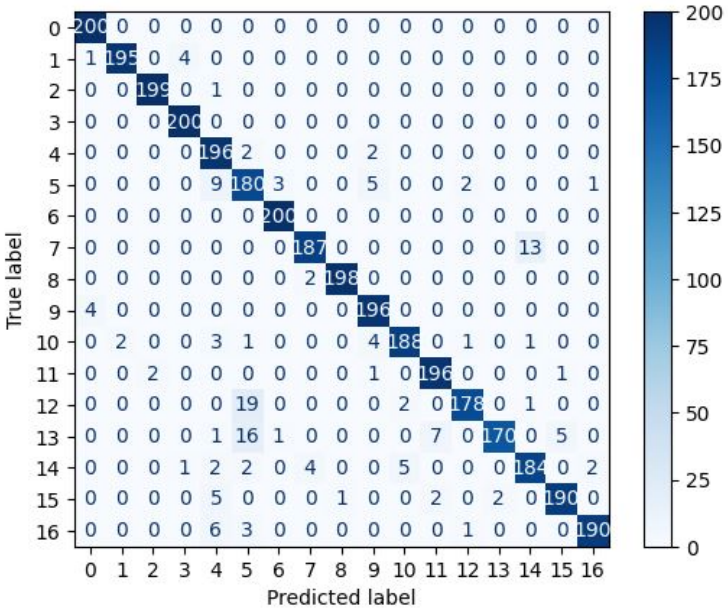


Figure 3.2: CNN-1D Model Confusion Matrix

Overall, the confusion matrix shows that the CNN model performs well across the 17 PQD classes, with 3247 correctly classified samples out of 3400, corresponding to an overall accuracy of 95.5% (The final held-out test accuracy). Most classes have high numbers of correctly classified samples, as shown by the strong values along the diagonal.

Confusion Between Transient-Related Classes

The most important group of errors is observed among the transient-related classes. The main misclassifications are:

- Sag with Oscillatory Transient: 19 samples are incorrectly classified as Oscillatory Transient.
- Swell with Oscillatory Transient: 16 samples are incorrectly classified as Oscillatory Transient.
- Oscillatory Transient: 9 samples are incorrectly classified as Transient.
- Notch: 6 samples are incorrectly classified as Transient.
- Oscillatory Transient: 5 samples are incorrectly classified as Flicker.

These errors indicate that the CNN model has more difficulty distinguishing some short-duration disturbances from each other. In particular:

- Both *Sag with Oscillatory Transient* and *Swell with Oscillatory Transient* are sometimes classified as *Oscillatory Transient*.
- Some *Oscillatory Transient* samples are classified as the general *Transient* class.
- Some *Notch* samples are also classified as *Transient*.

These results suggest that the CNN model can detect the presence of a short-duration disturbance but may have difficulty identifying the exact type when several disturbance characteristics are present in the same signal. This represents the most important source of errors in the CNN confusion matrix.

Confusion Between Harmonics with Sag and Harmonics with Swell

Another important confusion is observed between the two combined harmonic classes:

- Harmonics with Sag: 13 samples are incorrectly classified as Harmonics with Swell.
- Harmonics with Swell: 2 samples are incorrectly classified as Harmonics with Sag.

Both classes contain harmonics, but they differ in the additional amplitude disturbance, namely Sag or Swell. The CNN model correctly identifies the harmonic component in many cases, but some samples are incorrectly classified according to the type of amplitude variation.

The higher number of errors from *Harmonics with Sag* to *Harmonics with Swell* (13 samples) shows that this confusion is not completely symmetric. However, the confusion matrix alone cannot determine the exact reason for this difference.

Therefore, these results indicate that distinguishing between combined classes that share a common harmonic component can be more difficult for the CNN model.

Dispersed Errors from Sag with Harmonics

The *Sag with Harmonics* class also shows errors with several different classes. The main misclassifications are:

- Sag with Harmonics: 5 samples are incorrectly classified as Flicker with Sag.
- Sag with Harmonics: 4 samples are incorrectly classified as Harmonics with Sag.
- Sag with Harmonics: 2 samples are incorrectly classified as Transient.
- Sag with Harmonics: 2 samples are incorrectly classified as Oscillatory Transient.
- Sag with Harmonics: 2 samples are incorrectly classified as Notch.

These errors show that the *Sag with Harmonics* class does not have only one dominant source of confusion. Some of its errors occur with other classes that share a common disturbance characteristic.

- The confusion with *Harmonics with Sag* is understandable because both classes contain sag and harmonic components.
- The confusion with *Flicker with Sag* may be related to the common Sag component.
- The remaining errors are smaller and are distributed among several transient-related classes.

Therefore, *Sag with Harmonics* is one of the more difficult combined classes for the CNN model, with errors distributed across several categories.

Confusion Among Swell-Related Combined Classes

Some additional errors are observed among the combined classes involving Swell. The main examples are:

- Swell with Oscillatory Transient $\rightarrow$ Flicker with Swell: 7 samples.
- Swell with Oscillatory Transient $\rightarrow$ Swell with Harmonics: 5 samples.
- Swell with Harmonics $\rightarrow$ Transient: 5 samples.
- Swell with Oscillatory Transient $\rightarrow$ Oscillatory Transient: 16 samples.

These results show that the CNN model can sometimes correctly detect the main Swell characteristic but incorrectly identify the additional disturbance. For example, a signal containing both Swell and an oscillatory transient may sometimes be classified as *Oscillatory Transient* or another combined class.

This suggests that the presence of multiple disturbance characteristics makes some combined PQD classes more difficult to distinguish. However, these errors are less concentrated than the main confusion observed among the transient-related classes.

Synthesis

Overall, the confusion matrix shows that the CNN model achieves strong classification performance, with an overall accuracy of 95.5%. Most individual PQD classes are recognized correctly, while the remaining errors are mainly concentrated in combined and transient-related disturbances.
The main observations are:

- The largest group of errors involves transient-related classes, particularly *Sag with Oscillatory Transient*, *Swell with Oscillatory Transient*, *Oscillatory Transient*, and *Notch*.
- *Harmonics with Sag* and *Harmonics with Swell* are also sometimes confused because both classes contain harmonic components.
- *Sag with Harmonics* shows errors with several different combined disturbance classes.
- Some confusion is also observed among the combined classes involving Swell.

These results indicate that the CNN model performs particularly well when the disturbance has clear and distinct characteristics. In contrast, classification becomes more difficult when several disturbances are combined or when different classes contain similar signal characteristics.

It is important to note that the confusion matrix shows which classes are confused, but it does not provide the exact reason for every misclassification. Some errors may be related to similarities between classes, while others may result from variations in the signals, limitations of the model, or other factors. Therefore, the observed patterns should be considered possible explanations rather than definite causes for every individual error.

3.5.3 Comparison of the CNN and BiLSTM Confusion Matrices

The confusion matrices of the CNN and BiLSTM models show similar general behavior, but the distribution of errors differs between the two models.

Similarities

- Both models correctly classify most samples, with high values along the main diagonal.
- Both models perform better on individual disturbances than on some combined disturbances.
- Both models show confusion between *Harmonics with Sag* and *Sag with Harmonics*.
- Transient-related classes, particularly *Transient*, *Oscillatory Transient*, and *Notch*, are also sources of errors for both models.

Differences

- The CNN has fewer total errors and achieves a higher accuracy (95.5%) than the BiLSTM (93%).
- The CNN errors are mainly noticeable in transient-related and some combined classes.
- The BiLSTM shows stronger confusion between *Harmonics with Sag* and *Sag with Harmonics*, as well as between some transient-related classes.
- The two models do not always misclassify the same samples, showing that they learn different characteristics from the PQD signals.

Overall, both models show similar classification difficulties, mainly for combined and similar disturbances. However, the CNN provides better overall separation between the classes, resulting in fewer classification errors than the BiLSTM.

3.6 Deployment Results on STM32H743

This section presents the deployment results of the CNN-1D and BiLSTM models on the STM32H743, focusing on their embedded performance, memory requirements, and inference time to evaluate their suitability for real-time power quality disturbance classification.

3.6.1 Analysis of the CNN-1D Embedded Profiling

The 1D Convolutional Neural Network (`cnn-1d`) was deployed and validated directly on the STM32H743 microcontroller using the X-Cube-AI package. The target validation provides information about the tensor dimensions, computational complexity, inference latency, and numerical agreement between the reference model and its embedded implementation. The main profiling results are summarized in **Table 3.5**.

Table 3.5: Main CNN-1D profiling results on the STM32H743

Parameter	Result
Validation samples	3,400
Input tensor	$[1, 100, 1, 1]$
Input size	400 bytes (`float32`)
Output tensor	$[1, 1, 1, 17]$
Average inference time	24.873 ms
CPU cycles	1,591,873 cycles
Computational workload	314,176 MACC
Efficiency	5.07 cycles/MACC

The reported tensor dimensions require clarification:

- **Original model input:** $(100, 1)$, corresponding to 100 temporal samples and one input channel.
- **Cube.AI representation:** $[1, 100, 1, 1]$. The additional dimension of size 1 is an internal four-dimensional representation used by the AI runtime and does not change the original CNN-1D architecture.

- **Output:** $[1, 1, 1, 17]$, corresponding to the 17 output classes.

The most important performance indicators can be interpreted as follows:

- **Average inference time: 24.873 ms.** This is the average time required by the STM32H743 to process **one complete input window of 100 samples** and produce the classification result.

- **Inference throughput:** The corresponding theoretical sequential throughput is:

$$\frac{1}{24.873 \times 10^{-3}} \approx 40.2 \text{ inferences/s.}$$

  Thus, the microcontroller can perform approximately 40 classifications per second, with each classification corresponding to one 100-sample input window.

- **CPU cycles: 1,591,873 cycles.** This represents the number of processor cycles required to execute one inference and provides a hardware-oriented measure of the computational cost of the model.

- **Computational workload: 314,176 MACC.** MACC stands for Multiply-Accumulate and represents one of the fundamental operations performed by convolutional neural networks $ab + c$.

  – Multiplication : ab

  – Addition/accumulation : $+c$

  The reported value indicates that approximately 314 thousand MACC operations are required for one input window.

- **Efficiency: 5.07 cycles/MACC.** This value represents the average number of CPU cycles required for one MACC operation:

$$\text{Cycles/MACC} = \frac{1,591,873}{314,176} \approx 5.07.$$

  Therefore, approximately 5.07 CPU cycles are required on average for each MACC operation. This metric is useful for comparing the computational efficiency of different models or deployment configurations.

Real-Time Interpretation

The relationship between the acquisition parameters and the inference latency is particularly important for the embedded application. The CNN-1D model was trained using 100-sample windows at a sampling frequency of 5 kHz. Consequently:

- **Sampling frequency:** $F_s = 5$ kHz = 5000 Hz.

- **Number of samples per input:** 100 samples.

- **Duration of one input window:**

$$T_{\text{window}} = \frac{100}{5000} = 20 \text{ ms.}$$

- **Inference time:** 24.873 ms per 100-sample window.

- **Theoretical inference rate:** approximately 40.2 windows/classifications per second.

Therefore, one signal window is acquired in **20 ms**, while its classification requires approximately **24.873 ms** when processed sequentially. This means that the inference stage is slightly slower than the arrival of successive non-overlapping 100-sample windows.

Consequently, the current implementation provides rapid embedded classification. For continuous real-time operation, DMA-based double buffering can maintain uninterrupted data acquisition.

The sampling frequency must also remain consistent with the training configuration. Since the model was trained with $F_s = 5$ kHz, each input of 100 samples represents 20 ms of the electrical signal, corresponding to one 50-Hz fundamental cycle:

$$\frac{5000}{50} = 100 \text{ samples/cycle.}$$

Maintaining the same sampling frequency during deployment therefore preserves the temporal representation learned by the CNN-1D model.

Layer-by-Layer Profiling

The layer-by-layer profiling shows that the computational cost is strongly dominated by the convolutional stages. The main results are presented in **Table 3.6**.

Table 3.6: Layer-by-layer execution time of the CNN-1D model

Layer	Time (ms)	Share	CPU cycles
Conv2D (`ai_node_0`)	2.240	9.0%	143,367
Conv2D (`ai_node_4`)	9.466	38.1%	605,809
Conv2D (`ai_node_8`)	11.143	44.8%	713,159
Three convolutional layers	**22.849**	**91.9%**	–

Although X-Cube-AI reports these operations as `Conv2D`, they correspond to the convolutional operations of the original CNN-1D model. The four-dimensional representation used by the AI runtime explains both the `Conv2D` notation and the input tensor shape $[1, 100, 1, 1]$. The profiling results show that:

- The three convolutional layers account for **91.9%** of the total inference time.

- Their combined execution time is **22.849 ms** out of the total 24.873 ms.

- **Batch Normalization, activation, pooling, Dense, and Softmax** operations account for only the remaining **8.1%** of the inference time.

The increasing computational cost of the deeper convolutional layers is related to the increase in the number of feature maps throughout the network. As a result, these layers perform a larger number of convolutional operations and become the main computational bottleneck of the embedded implementation.

Target Validation and Numerical Agreement

The target validation was performed to verify that the CNN-1D model deployed on the STM32H743 produces outputs consistent with those of the reference model. This validation does not measure the classification accuracy of the 17 power quality classes; instead, it evaluates whether the embedded implementation preserves the numerical outputs of the original model.



Figure 3.3: Validation on target results

The obtained results illustrated in **Figure 3.3** are summarized as follows:

- **Cross-validation agreement: 100.00%.** This indicates that the outputs generated by the embedded implementation are fully consistent with the outputs of the reference model according to the validation criterion used by X-Cube-AI.
- **RMSE (Root Mean Square Error):** 8.6×10^{-8}**.** RMSE measures the average magnitude of the difference between the reference outputs and the outputs generated on the STM32H743. A value close to zero indicates that the two sets of outputs are numerically very similar. The obtained value is extremely small, showing negligible numerical error.
- **MAE (Mean Absolute Error):** 1.5×10^{-8}**.** MAE measures the average absolute difference between the reference and embedded outputs. As with RMSE, a value close to zero indicates a high numerical agreement. The obtained value confirms that the differences between the two implementations are negligible.
- **Cosine Similarity: 1.000000.** Cosine Similarity measures the similarity between the directions of the reference and embedded output vectors. Its value ranges from -1 to 1, where a value of 1 indicates identical vector directions. The obtained value of 1.000000 therefore indicates an almost perfect correspondence between the two output vectors.

The combination of these metrics demonstrates that the CNN-1D model is faithfully reproduced on the STM32H743. The RMSE and MAE values are extremely close to zero, while the Cosine Similarity is equal to 1.000000. Therefore, the embedded implementation introduces only negligible numerical differences compared with the reference model.

It is important to distinguish this target validation from the classification accuracy reported during model evaluation. The **100.00% cross-validation agreement does not mean that the CNN-1D achieves 100% classification accuracy**. Instead, it means that the embedded model produces outputs that agree with the reference model. The actual classification accuracy of the CNN-1D model remains the value obtained during the independent test-set evaluation.

Memory and Flash Resource Utilization

The resource consumption of the deployed CNN-1D model was evaluated using the X-Cube-AI analysis tool as illustrated in **Figure 3.4**. The reported values represent the total memory and Flash requirements of the generated AI application on the STM32H743.

```
** RT AI runtime objects (kernels+infrastructure)
 Summary - "stm32h7" target
 ---------------------------------------------------
              FLASH (ro)      %*   RAM (rw)       %
 ---------------------------------------------------
 RT total         17,376   29.7%      4,844   32.7%
 ---------------------------------------------------
 TOTAL            58,408             14,828
 ---------------------------------------------------
 *  rt/total
```

Figure 3.4: Memory and Flash resource utilization for the CNN-1D model

The total resource consumption is summarized as follows:

- **Total Flash:** 58,408 bytes, corresponding to approximately **57.04 KiB**.
- **Total RAM:** 14,828 bytes, corresponding to approximately **14.48 KiB**.

These values represent the total memory footprint required by the generated AI application, including the neural network data and the X-Cube-AI runtime components.

Considering the memory resources of the STM32H743VIT6, which provides **2 MB of Flash (2048KiB)** and **1 MB of RAM (1024KiB)**, the CNN-1D deployment occupies only a small portion of the available memory:

- **Flash utilization:** approximately $57.04/2048 \times 100 \approx \mathbf{2.8\%}$.
- **RAM utilization:** approximately $14.48/1024 \times 100 \approx \mathbf{1.4\%}$.

Therefore, the deployed CNN-1D has a relatively low memory footprint on the STM32H743VIT6. A large portion of the available Flash and RAM remains free for other components of the embedded system, such as signal acquisition, preprocessing, communication interfaces, application code, and additional processing functions. This confirms that the memory requirements of the CNN-1D are well within the hardware resources of the target microcontroller.

3.6.2 Analysis of the BiLSTMm Embedded Profiling

Initial Deployment Using STM32Cube.AI

After training and evaluating the deep learning model, the next step was to deploy the selected model on the **STM32H743VIT6** microcontroller. **STM32Cube.AI** was initially selected because it provides tools for analyzing and converting neural network models for STM32 devices.
The initial deployment procedure followed these steps:

- The trained model was exported in **TensorFlow Lite (TFLite)** format.
- The resulting `.tflite` file was imported into **STM32Cube.AI** for model analysis and conversion.
- STM32Cube.AI was expected to generate the optimized C code required for neural network inference on the STM32H743VIT6.
- The generated code would then be integrated into the STM32 application to perform the classification directly on the microcontroller.

This approach was initially successful for models containing supported operations. However, when the recurrent model was analyzed, several compatibility problems appeared during the conversion process.

Compatibility Issues During Model Deployment

The conversion errors were related to specific operations and graph structures that were not supported by the selected **X-CUBE-AI/ST Edge AI** conversion workflow. Therefore, the issue should not be considered as a limitation of the BiLSTM architecture itself, but rather as an **operator-level compatibility issue**.
The main problematic elements identified in the exported TFLite graph were:

- **Dynamic control-flow operations:** operations such as `WHILE` and `Loop` can be generated when recurrent computations are implemented.
- **Flex/TensorList operations:** operations such as `FlexTensorListReserve` and `FlexTensorListStack` can be introduced when TensorFlow represents sequential data using dynamic-length lists.
- **`ReverseSequence`:** this operation may appear in bidirectional recurrent architectures because the sequence is processed in both forward and backward directions.
- **Dynamic tensor dimensions:** the conversion process requires sufficiently static tensor dimensions. Models containing undefined or dynamic input dimensions may therefore fail during conversion.

These problems can occur in different neural network architectures whenever their exported computational graph contains unsupported operations. In the present case, they appeared because the standard Keras implementation of the BiLSTM generated a graph containing several dynamic sequence and control-flow operations.

Solution Approach

To overcome the compatibility issues encountered during the STM32Cube.AI conversion, an alternative deployment approach was adopted. The main steps of the proposed solution are summarized as follows:

- **Use of the `Gather` operation:** the `Gather` operation was introduced before exporting the model to TFLite to obtain a compatible model representation.
- **Application Template:** the STM32Cube.AI Application Template was enabled to integrate the TFLite Micro runtime and the model into the STM32 project.
- **Use of the TFLite Micro runtime:** when enabling **Application Template**, instead of using the standard "STM32Cube.AI MCU runtime", the TFLite model was executed using the "TFLite Micro runtime".

Trained BiLSTM → Gather-Based modification → TFLite Export → TFLite Micro Integration → STM32H743VIT6 → Embedded Classification

After introducing the `Gather` operation, the modified model was evaluated in Google Colab to verify that the graph modification did not affect its classification performance. The model achieved the same accuracy, confirming that the modification did not result in any accuracy degradation.

This approach resolved the operator-compatibility issues encountered with the standard STM32Cube.AI conversion workflow and produced a model representation compatible with the selected TFLite Micro deployment approach. However, compatibility with the embedded runtime does not by itself guarantee that the model can be executed within the available hardware resources. Therefore, the memory requirements of the resulting BiLSTM model were subsequently analyzed to determine its practical deployment feasibility on the **STM32H743VIT6**.

Memory Resource Analysis

After adopting the proposed deployment approach, the memory requirements of the BiLSTM model were analyzed as illustrated in **Figure 3.5** and **Figure 3.6** to evaluate its feasibility on the **STM32H743VIT6**.

```
Analyzing model
C:/Users/NTC/STM32Cube/Repository/Packs/.../AI/bilstm_gather.tflite
Flash:  1942148
Ram:  1367552
Analyze complete on AI model
Required Ram or Flash size for bilstm is bigger than available Ram or
Flash.
```

Figure 3.5: STM32Cube.AI memory analysis of the BiLSTM model showing insufficient available RAM

Used Flash: 1.85 MiB (0.00 B over 2.00 MiB Internal)
Used Ram: 1.30 MiB (**1.30 MiB over 1.03 MiB** Internal)
Analysis status: done

Figure 3.6: Used Flash and RAM memories

Those analysis provided the following results:

- **Flash memory:** the model requires approximately **1.85 MiB** of Flash memory, while approximately **2.00 MiB** of internal Flash is available. Therefore, the Flash requirement remains within the available capacity.
- **RAM:** the estimated RAM requirement is approximately **1.30 MiB**, whereas only **1.03 MiB** of internal RAM is available for the considered configuration. The required RAM therefore exceeds the available capacity.
- **Deployment limitation:** the analysis consequently indicates that the BiLSTM model cannot be fully accommodated in the available internal RAM using the analyzed configuration, despite the Flash requirement being within the available Flash capacity.

The analysis therefore highlights that **RAM is the main memory constraint** for the deployment of the BiLSTM model on the STM32H743VIT6. This result was taken into consideration when selecting and evaluating the final embedded deployment strategy.

3.6.3 Memory Resource Comparison and Final Model Selection

Following the embedded profiling of both models, a comparison of their memory requirements was performed to determine their feasibility for deployment on the STM32H743VIT6. The results are summarized in **Table 3.7**.

Table 3.7: Comparison of CNN-1D and BiLSTM memory requirements on the STM32H743VIT6

Memory Resource	**CNN-1D**	**BiLSTM**	**Available**
Flash memory	57.04 KiB	1.85 MiB	2.00 MiB
RAM memory	14.48 KiB	1.30 MiB	1.03 MiB
Deployment feasibility	**Feasible**	**Not feasible**	–

As shown in **Table 3.7**, the CNN-1D model requires significantly fewer memory resources than the BiLSTM model. Its Flash and RAM requirements remain well within the available hardware capacities, allowing sufficient memory to be reserved for other embedded functions, including signal acquisition, preprocessing, and communication.

In contrast, although the BiLSTM model fits within the available Flash memory, its RAM requirement of approximately 1.30 MiB exceeds the available 1.03 MiB. Consequently, the BiLSTM model cannot be deployed on the STM32H743VIT6 using the current hardware and software configuration.

Based on these results, the BiLSTM model was excluded from further hardware validation due to its excessive RAM requirements. The CNN-1D model was therefore retained for the experimental implementation. Accordingly, the following experimental results focus exclusively on the CNN-1D model and its validation on the physical STM32H743VIT6 hardware.

3.7 Explorations and Discarded Architecture Choices

Beyond the final model selection presented in the previous sections, an additional optimization strategies were investigated during the development of the embedded classification pipeline. Although not retained in the final implementation, these explorations are reported here for completeness, as they contributed to narrowing the design space and justified some of the final architectural decisions.

3.7.1 Transformer-Based Classification (Custom Implementation)

Motivation

The literature reports strong results for Transformer-based architectures on comparable classification tasks. Saber et al. [23] (2024) report a pretrained Transformer model reaching 99.81% accuracy under ideal conditions, and 91.43% accuracy under more realistic conditions including noise, DC offset, and parameter variation. Given these results, a Transformer-based classifier was considered as a candidate architecture for the embedded classification pipeline. **Table 3.8** shows the reported vs. reproduced accuracy for Transformer-based classification

Table 3.8: Reported vs. reproduced accuracy for Transformer-based classification

Source / Configuration	Accuracy
Saber et al. [23] – ideal conditions	99.81%
Saber et al. [23] – noise + DC offset + variation	91.43%
Custom implementation (this work)	68.7%

Outcome

Rather than reusing a pretrained model, the Transformer architecture was implemented and trained from scratch on the available dataset. The resulting classifier reached only 68.7% accuracy, well below both values reported by Saber et al. [23], and below the accuracy already achieved with the CNN-1D and BiLSTM architectures developed in this work.

Analysis

This gap is consistent with the data requirements typically associated with Transformer architectures. Unlike convolutional networks, self-attention layers carry comparatively little built-in inductive bias regarding local structure, and generally require either large training datasets or transfer learning from a pretrained model to reach competitive performance. The results reported by Saber et al. [23] were obtained with a pretrained model, whereas the

present implementation was trained from scratch on a comparatively limited dataset, without access to the original pretraining procedure, hyperparameters, or preprocessing pipeline. This difference in training conditions likely accounts for the observed accuracy gap.

Decision

Given that the reproduced accuracy (68.7%) was already well below the level required for reliable classification, and below the performance of the retained CNN-1D model, this architecture was not pursued further. In particular, it was not implemented or evaluated on the target STM32H743VIT6 hardware, since the model did not reach an acceptable accuracy level in software.

3.8 Experimental Results

The obtained TFLite classification accuracies are summarized in Table 3.9.

Table 3.9: CNN performance on clean and noisy test datasets.

Test Dataset	Float32 TFLite Accuracy
Clean	95.5%
40 dB	26.02%
30 dB	24.53%
20 dB	11.54%

The results show a substantial degradation in classification performance when artificial noise is introduced. The accuracy decreases from **95.5%** for the clean dataset to **26.02%**, **24.53%**, and **11.54%** for SNR levels of 40, 30, and 20 dB, respectively.

Experimentally, these results demonstrate that the current CNN is highly sensitive to the specific noise conditions introduced in this evaluation. The accuracy reduction therefore constitutes an important warning regarding the robustness of the model outside its original clean-data distribution.

However, these results should not be interpreted as a direct prediction of the CNN performance on real signals. The experiment evaluates the response to artificially generated Gaussian noise and does not reproduce the complete statistical behavior of a physical acquisition system.

3.8.1 Engineering Decision: Delayed Real-Signal Hardware Validation

The robustness results motivated the decision not to proceed immediately to real-world hardware acquisition using the current CNN. This decision follows a progressive validation strategy:

Clean Dataset → Artificially Noisy Dataset → Real Acquired Signals → Real-Signal Hardware Validation

The objective of this intermediate step was to identify potential robustness limitations before allocating additional experimental time and hardware resources to real signal acquisition.

Importantly, this decision does not imply that the CNN is unsuitable for real-world deployment. Rather, the noisy-data experiment revealed that the current training conditions may not provide sufficient robustness to input disturbances, indicating that additional validation and potentially additional training strategies are required.

3.8.2 Limitations of the Artificial-Noise Evaluation

The artificial-noise experiment provides a useful preliminary robustness assessment, but it cannot reproduce all the imperfections of a real measurement chain.

In an actual acquisition system, the measured signal may be affected by several factors, including:

- sensor noise;
- DC offset;
- ADC quantization;
- ADC conversion errors;
- gain and amplitude variations;
- sensor and conditioning-circuit distortions;
- electromagnetic interference;
- frequency and sampling variations;
- nonlinearities of the acquisition chain.

The Gaussian noise introduced in the present experiment represents only one possible source of disturbance. Consequently, the noisy dataset should be considered a **preliminary robustness test** rather than a complete representation of real-world acquisition conditions.

Another limitation concerns the availability of suitable real PQD datasets. Existing datasets may use different sampling frequencies, signal durations, or numbers of samples per electrical cycle. Such differences make direct evaluation difficult when the proposed system is specifically designed around a sampling frequency of 5 kHz and an input window of 100 samples.

3.8.3 Validation Path Toward Real-World Deployment

The adopted validation strategy follows a progressive path from controlled conditions toward real-world deployment:

Ideal Data → Problem Identification → Noise Robustness Test → Performance Degradation → Deployment Decision → Robustness Improvement → Real-World Validation

The experiments demonstrate that the CNN achieves high accuracy on clean data but experiences a significant performance degradation under the artificial noise conditions tested. This finding does not determine its future performance on real signals, but it provides an important indication that robustness must be further investigated before final embedded deployment.

Thus, the use of controlled noisy data constitutes a practical intermediate validation stage that allows potential generalization limitations to be identified before conducting more demanding real-world hardware experiments.

3.9 Conclusion

This chapter presented and discussed the experimental results obtained for the proposed power quality disturbance classification system, covering model training, classification performance, embedded deployment, and robustness evaluation.

The CNN-1D model achieved a best validation accuracy of **96.9%** , outperforming the BiLSTM model, which achieved **94.4%** validation accuracy. The confusion matrix analysis revealed that both models classified most individual disturbances accurately, while combined disturbances, particularly those involving harmonics and oscillatory transients, remained more challenging. An additional Transformer-based experiment achieved only **68.7%** accuracy and was consequently not retained for embedded implementation.

Regarding embedded deployment, the CNN-1D model was successfully validated on the STM32H743 microcontroller, requiring approximately **57.04 KiB of Flash** and **14.48 KiB of RAM**, with an average inference time of **24.873 ms**. The embedded implementation also demonstrated excellent numerical agreement with the reference model, achieving a cross-validation agreement of **100%**. However, the inference time exceeded the 20 ms acquisition window, indicating that further optimization is necessary for continuous real-time classification of successive non-overlapping windows. In contrast, the BiLSTM deployment was limited by its estimated RAM requirement of approximately **1.30 MiB**, exceeding the available **1.03 MiB**.

Finally, the preliminary robustness evaluation revealed a significant degradation in CNN classification accuracy under artificial Gaussian noise, decreasing from **95.5%** on clean signals to **26.02%**, **24.53%**, and **11.54%** at SNR levels of 40, 30, and 20 dB, respectively. These findings highlight the need for more representative training data, improved robustness, and further validation using real signals acquired through the complete measurement chain. Overall, the results demonstrate the feasibility of deploying the CNN-1D architecture on the selected microcontroller while identifying the remaining challenges related to real-time execution and generalization under practical operating conditions. These findings provide the basis for the general conclusion and future perspectives of this work.

General Conclusion

This thesis investigated the development of an intelligent embedded system for power quality disturbance (PQD) classification using deep learning and the STM32H743VIT6 microcontroller. The main objective was to evaluate the classification performance of different neural network architectures while considering the practical constraints associated with their deployment on a resource-constrained embedded platform. Particular attention was given to the relationship between classification accuracy, memory consumption, inference time, and robustness under non-ideal acquisition conditions.

The proposed methodology involved the preparation and statistical analysis of the SEED-PQD-v1 dataset, the investigation of preprocessing and normalization strategies, the development and comparison of CNN-1D and BiLSTM models, and the evaluation of their embedded deployment feasibility. An additional Transformer-based architecture was also investigated. Finally, a preliminary robustness experiment was conducted to assess the sensitivity of the retained CNN model to artificial measurement noise.

The dataset verification and preprocessing stages highlighted the importance of preserving the physical characteristics of electrical waveforms. The selected global Min–Max normalization maintained the relative amplitude relationships between samples, which are essential for distinguishing power quality disturbances such as voltage sag, swell, interruption, harmonics, and transient events. This preprocessing strategy provided a consistent basis for training and evaluating the proposed classification models.

The experimental results obtained throughout this work allow the three research hypotheses formulated in the General Introduction to be addressed as follows.

Evaluation of the Research Hypotheses

- **H1: Classification Performance of CNN-1D and BiLSTM**

  The first hypothesis proposed that the CNN-1D architecture would achieve higher classification accuracy than the BiLSTM architecture on the dataset used in this study.

  The obtained results support this hypothesis for the investigated model configurations. The CNN-1D achieved a best validation accuracy 93%, respectively, for the BiLSTM model. The analysis of classification reports and confusion matrices also revealed that both models successfully recognized most individual disturbances, whereas certain combined disturbances, particularly those involving harmonics and oscillatory transients, remained more challenging.

These findings demonstrate that the CNN-1D achieved higher classification accuracy under the experimental conditions considered in this work. However, this conclusion is specific to the dataset, preprocessing strategy, and model configurations investigated and should not be generalized to all CNN and BiLSTM architectures.

- **H2: Embedded Deployment and Hardware Resource Constraints**

  The second hypothesis concerned the differences in memory and computational requirements between CNN-1D and BiLSTM architectures and their influence on deployment feasibility on the STM32H743VIT6 microcontroller.

  The embedded profiling results support this hypothesis with respect to memory requirements and deployment feasibility. The CNN-1D required approximately 57.04 KiB of Flash and 14.48 KiB of RAM, allowing its successful deployment and target validation. Its average inference time was measured at 24.873 ms, with a computational workload of 314,176 MACC operations. The embedded implementation also achieved 100% cross-validation agreement with the reference model according to the STM32Cube.AI validation criterion, demonstrating numerical consistency between the two implementations. This agreement should not be confused with classification accuracy.

  In contrast, although the BiLSTM compatibility issues were addressed through an alternative TFLite Micro deployment approach, the resulting model required approximately 1.85 MiB of Flash and 1.30 MiB of RAM. Its RAM requirement exceeded the approximately 1.03 MiB available in the analyzed configuration. Consequently, the BiLSTM model could not be deployed using the available internal memory and was excluded from further hardware validation.

  The results therefore demonstrate that the two investigated models have substantially different memory requirements, which directly influenced their suitability for the target microcontroller. However, a direct comparison of their inference latency on the physical STM32 could not be performed because the BiLSTM model was not successfully deployed.

  Furthermore, the CNN inference time of 24.873 ms exceeded the 20 ms duration of one 100-sample input window at a sampling frequency of 5 kHz. Thus, although its deployment and numerical validation were successful, further optimization is required to guarantee continuous classification of every successive non-overlapping window without accumulating processing delays.

- **H3: Generalization and Robustness Under Real-World Conditions**

  The third hypothesis concerned the potential degradation of classification performance when a model trained on clean synthetic signals is exposed to different input conditions.

  The preliminary noise experiments revealed a significant sensitivity of the CNN-1D model to artificial Gaussian noise, providing initial evidence of limitations in its gen-

eralization capability. However, the performance of the model on real acquired signals has not yet been evaluated.

Consequently, this hypothesis is only partially supported by the present experiments and requires further investigation using representative real-world measurements. The improvement of the CNN-1D model will therefore be addressed in future work after constructing a dataset acquired directly from the intended measurement system.

Future Perspectives and Proposed Improvements

Based on the limitations identified in this work, the following improvements are proposed for future development.

Construction of a Real-World PQD Dataset

The first step will consist of constructing an experimental PQD dataset using the intended acquisition system. The acquired signals could be characterized according to several parameters, including:

- Signal-to-Noise Ratio (SNR);
- DC offset;
- signal amplitude and gain;
- ADC noise and quantization;
- sensor characteristics and signal distortion;
- sampling frequency and acquisition-chain effects.

This dataset will provide representative signals for evaluating and improving the CNN-1D model under actual acquisition conditions.

Improvement of the CNN-1D Model

Once the real-world dataset is available, the CNN-1D model could be improved through:

- Evaluation of the current model on real acquired signals;
- Identification of the most frequently misclassified PQD classes;
- Retraining or fine-tuning using representative real-world data;
- Adjustment of the preprocessing strategy if necessary;
- Data augmentation based on the imperfections observed in the real measurements;
- Evaluation of the improved model using accuracy, precision, recall, F1-score, and confusion matrices.

The objective is to improve the model's robustness and generalization capability under practical operating conditions.

Final Hardware Validation

After improving the CNN-1D model, the complete system should be validated using real acquired signals. This evaluation will focus on:

- Classification accuracy on real PQD signals;
- Inference latency and real-time performance;
- Flash and RAM consumption;
- Continuous acquisition and processing;
- Overall system stability and reliability.

This final stage will allow the practical performance of the proposed embedded classification system to be assessed under representative operating conditions.

Bibliography

[1] J. Li, H. Liu, D. Wang, and T. Bi, *Classification of Power Quality Disturbance Based on S-Transform and Convolution Neural Network*, Frontiers in Energy Research, vol. 9, p. 708131, 2021.

[2] M. A. Rodriguez, J. F. Sotomonte, J. Cifuentes, and M. Bueno-Lopez, *Power Quality Disturbance Classification via Deep Convolutional Autoencoders and Stacked LSTM Recurrent Neural Networks*. In *2020 International Conference on Smart Energy Systems and Technologies (SEST)*, IEEE, pp. 1–6, 2020.

[3] N. Mohan, K. Soman, and R. Vinayakumar, *Deep Power: Deep Learning Architectures for Power Quality Disturbances Classification*. In *2017 International Conference on Technological Advancements in Power and Energy (TAP Energy)*, IEEE, pp. 1–6, 2017.

[4] W. Qiu, Q. Tang, J. Liu, and W. Yao, *An Automatic Identification Framework for Complex Power Quality Disturbances Based on Multi-Fusion Convolutional Neural Network. IEEE Transactions on Industrial Informatics*, vol. 16, no. 5, pp. 3233–3241, 2019.

[5] C. Iturrino-García, G. Patrizi, A. Bartolini, L. Ciani, L. Paolucci, A. Luchetta, and F. Grasso, *An Innovative Single Shot Power Quality Disturbance Detector Algorithm. IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–10, 2022.

[6] L. Fu, K. Yan, and T. Zhu, *PowerCog: A Practical Method for Recognizing Power Quality Disturbances Accurately in a Noisy Environment. IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3105–3113, 2021.

[7] J. Tian, B. Wang, J. Li, and Z. Wang, *Adversarial Attacks and Defense for CNN-Based Power Quality Recognition in Smart Grid. IEEE Transactions on Network Science and Engineering*, vol. 9, no. 2, pp. 807–819, 2021.

[8] N. K. Manaswi and S. John, *Deep Learning with Applications Using Python*. Springer, 2018.

[9] M. Mishra, *Power Quality Disturbance Detection and Classification Using Signal Processing and Soft Computing Techniques: A Comprehensive Review. International Transactions on Electrical Energy Systems*, vol. 29, no. 8, p. e12008, 2019.

[10] C. Ge, R. A. de Oliveira, I. Y.-H. Gu, and M. H. Bollen, *Deep Feature Clustering for Seeking Patterns in Daily Harmonic Variations. IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–10, 2020.

[11] M. Sahani and P. K. Dash, *FPGA-Based Semisupervised Multi-Fusion RDCNN of Process Robust VMD Data with Online Kernel RVFLN for Power Quality Events Recognition*. *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 2, pp. 515–527, 2020.

[12] D. Gu, Y. Gao, Y. Li, Y. Zhu, and C. Wu, *A Novel Label-Guided Attention Method for Multilabel Classification of Multiple Power Quality Disturbances*. *IEEE Transactions on Industrial Informatics*, vol. 18, no. 7, pp. 4698–4706, 2021.

[13] D. H. Chiam and K. H. Lim, *Power Quality Disturbance Classification Using Transformer Network*. In *International Conference on Cyber Warfare, Security and Space Research*, Springer, pp. 272–282, 2021.

[14] D. H. Chiam, K. H. Lim, and K. H. Law, *Detection of Power Quality Disturbances Using Wavelet-Based Convolutional Transformer Network*. In *2022 International Conference on Green Energy, Computing and Sustainable Technology (GECOST)*, pp. 150–154, 2022.

[15] M. F. Al-Mashdali, A. Islam, A. Hassan, M. Shafiullah, M. Al-Dhaifallah, and K. Al Fuwail, *Signal Processing-Free Intelligent Model for Power Quality Disturbances Identification*, *IEEE Access*, 15 January 2025.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[17] V. Nair and G. E. Hinton, *Rectified Linear Units Improve Restricted Boltzmann Machines*. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pp. 807–814, 2010.

[18] A. Graves, S. Fernandez, and J. Schmidhuber, *Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition*, Neural Networks, 2005.

[19] Z. Hou, B. Wang, J. Liu, Y. He, and Y. Yao, *Physics-Inspired Time-Frequency Feature Extraction and Lightweight Neural Network for Power Quality Disturbance Classification*, Frontiers in Physics, 17 July 2025.

[20] P. Warden and D. Situnayake, *Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, 2020.

[21] V. G. Neve, P. V. Pullawar, and C. H. Kidile, *Power Quality Issues & Detection Methods for Voltage Sag and Swell*, I J R B A T, Special Issue 16, vol. II, pp. 71–79, July 2020.

[22] R. Machlev, A. Chachkes, J. Belikov, Y. Beck, and Y. Levron, *Open Source Dataset Generator for Power Quality Disturbances with Deep-Learning Reference Classifiers*.

[23] A. M. Saber, A. Youssef, D. Svetinovic, H. Zeineldin, D. Kundur, and E. El-Saadany, *Enhancing Power Quality Event Classification with AI Transformer Models*, arXiv, 2024.

[24] F. A. Albalooshi and M. R. Qader, *Deep Learning Algorithm for Automatic Classification of Power Quality Disturbances*, MDPI, 2025.

[25] F. W. Scholz and M. A. Stephens, *K-Sample Anderson-Darling Tests*, Journal of the American Statistical Association, vol. 82, no. 399, pp. 918–924, September 1987.

[26] A. Ichwani, R. I. Kesuma, A. Setiawan, I. E. Wicaksono, and R. Hanifah, *Preventing Data Leakage in Classification via Integrated Machine Learning Pipelines: Preprocessing, Feature Transformation, and Hyperparameter Tuning*, Jurnal Teknik Informatika (JUTIF), February 2026.

[27] C. Zhang, D. Yang, T. Ishizaki, R. Zhu, B. Li, and W. Liao, *A Power Quality Disturbance Classification Method Based on Fine-Tuned Moment Foundation Model*, IET Generation, Transmission & Distribution, 18 April 2026.

[28] R. C. Staudemeyer and E. R. Morris, *Understanding LSTM: A Tutorial into Long Short-Term Memory Recurrent Neural Networks*, arXiv, 2019.