NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

LEANDRO CORREIA CORDEIRO

BSc in Electrical and Computer Engineering

REAL-TIME PHYSICAL LAYER AUTHENTICATION ON ZIGBEE NETWORKS

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
September, 2024

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

REAL-TIME PHYSICAL LAYER AUTHENTICATION ON ZIGBEE NETWORKS

LEANDRO CORREIA CORDEIRO

BSc in Electrical and Computer Engineering

Adviser: Luís Filipe Lourenço Bernardo
Associate Professor, NOVA University Lisbon

Examination Committee

Chair: Bruno João Nogueira Guerreiro
Assistant Professor, NOVA University Lisbon

Rapporteur: Rodolfo Alexandre Duarte Oliveira
Associate Professor, NOVA University Lisbon

Adviser: Luís Filipe Lourenço Bernardo
Associate Professor, NOVA University Lisbon

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
September, 2024

Real-time Physical Layer Authentication on ZigBee networks

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.7) [1].

Acknowledgements

I would like to thank the following people who have accompanied me over the last few years and without whom none of this would have been possible.

To my university, NOVA School of Science and Technology, for giving me the opportunity to study on this incredible campus, full of exceptional professionals.

I would like to start by thanking my adviser, Luís Bernardo. There are no words to describe his willingness to help me, his commitment to the project and his support throughout the development of this thesis. Thank you for all your advice and for always encouraging me to achieve a better result.

I would like to thank the friends I made at university, who have been by my side through the most difficult times and the most exciting celebrations. The conversations, the laughs, the study sessions and the trips all contributed to making this experience unforgettable.

I would also like to thank my closest friends outside of university: Sofia Vivar, Tiago Gomes and Gabriela Cordeiro, for their interest in helping me and for their availability.

I would also like to thank my girlfriend, Marta Aleixo, for her constant presence and unconditional support. Thank you for believing in me. I am very grateful to have you in my life.

I cannot fail to mention the importance of my family, who have always believed in me and provided me with all the necessary conditions so that I could dedicate myself to my studies. A simple thank you doesn't come close to expressing all my gratitude.

Finally, I would like to thank all those who have contributed in some way to my journey over the years. Each and every one of you has left an important mark on my journey, and for that I am eternally grateful.

To all of you, thanks.

This work is funded by *Instituto de Telecomunicações* and *Fundação para a Ciência e Tecnologia* under the projects CELL-LESS6G (2022.08786.PTDC) and UIDB/50008/2020.

”

“Where focus goes, energy flows.”

— Anthony Robbins

Abstract

The rapid expansion of the Internet of Things (IoT) raises significant security concerns, especially for device authentication. This thesis focuses on Physical Layer (PHY) authentication through Radio Frequency Fingerprinting (RFF), using machine learning algorithms implemented with Software Defined Radio (SDR).

A custom and extended GNU Radio module was developed to capture ZigBee frames and extract the preamble. The system employed a One-Dimension Convolutional (Conv1D) Autoencoders (AE) Neural Network (NN) with a dimension of 12 to generate features, which were subsequently used to train a Random Forest (RF) classifier.

A prototype application was developed to implement the RFF algorithms in real-time. The application's graphical user interface supports the training of the RF classifier and enables real-time classification of IEEE 802.15.4 signals. The application provides an intuitive platform for users to train models, integrate new sensors, and perform real-time classifications efficiently.

Results demonstrated that the system performed best in static conditions, achieving an F1-score of 97% when the training and testing conditions were aligned at a distance of 15 cm. However, as the distance increased, performance declined, with the F1-score dropping to 90% at 1 meter and further decreasing to 89% when movement was introduced. Tests with different antennas and Low-Noise Amplifier (LNA)s from the same manufacturer showed no significant impact on performance. However, changing the testing environment, which inherently involved changing communication channels, led to a noticeable decrease in accuracy.

In conclusion, this thesis shows that Machine Learning (ML)-based PHY layer authentication is feasible and effective with aligned training and testing conditions.

Keywords: Machine Learning, Physical Layer Authentication, Software-Defined Radio, Internet of Things, RF Fingerprinting, Neural Network, Real-time Classification, GNU Radio

Resumo

O crescimento rápido da Internet das Coisas levanta preocupações de segurança significativas, especialmente para a autenticação de dispositivos. Esta dissertação foca-se na autenticação de nível físico através de impressão digital de radiofrequência, utilizando algoritmos de aprendizagem automática implementados com um rádio definido por software (SDR).

Foi desenvolvido um módulo GNU Radio personalizado e alargado para capturar tramas ZigBee e extrair o preâmbulo. O sistema utilizou um autoencoder de Rede Neuronal Convolucional de uma dimensão com um estado latente de 12 para gerar caraterísticas, que foram posteriormente utilizadas para treinar um classificador *Random Forest*.

Foi desenvolvido um protótipo de aplicação para implementar os algoritmos de Impressão digital por radiofrequência em tempo real. A interface gráfica da aplicação suporta o treino do classificador *Random Forest* e permite a classificação em tempo real de sinais IEEE 802.15.4. A aplicação fornece uma plataforma intuitiva para os utilizadores treinarem modelos, integrarem novos sensores e efetuarem classificações em tempo real de forma eficiente.

Os resultados demonstraram que o sistema teve um desempenho melhor em condições estáticas, atingindo um F1-score de 97% quando as condições de treino e teste estavam alinhadas e a transmissão é realizada a 15 cm. No entanto, à medida que a distância aumenta, o desempenho diminuí, com o F1-score a cair para 90% a 1 metro e a diminuir ainda mais para 89% quando se introduz movimento. Os testes com antenas e amplificadores de baixo ruido diferentes, mas do mesmo fabricante, não mostraram um impacto significativo no desempenho. No entanto, a alteração do ambiente de classificação, que implicava inerentemente a mudança do canal de comunicação, conduziu a uma diminuição significativa da precisão.

Em conclusão, esta tese mostra que a autenticação do nível físico baseada na aprendizagem automática é viável e eficaz quando as condições de treino e teste estão devidamente alinhadas.

Palavras-chave: Aprendizagem automática, Autenticação no nivel fisico, Rádio Definido

por Software, Internet das Coisas, Impressão Digital por Radio Frequência, Redes Neuronais, Classificação em tempo real, GNU Radio

Contents

List of Figures ix

Acronyms xi

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Objectives . . . 1
- 1.3 Contributions . . . 2
- 1.4 Document structure . . . 2

2 State of Art 4
- 2.1 Physical Layer Authentication . . . 4
  - 2.1.1 Traditional cryptographic techniques . . . 4
  - 2.1.2 PLA schemes in the literature . . . 5
  - 2.1.3 Active Methods . . . 6
    - 2.1.3.1 Non-covert schemes . . . 6
    - 2.1.3.2 Covert schemes . . . 6
  - 2.1.4 Passive Methods . . . 7
    - 2.1.4.1 Channel based . . . 7
    - 2.1.4.2 RF fingerprint based . . . 10
    - 2.1.4.3 RF Fingerprinting Approaches in Literature . . . 15
- 2.2 Integration of ML and GNU Radio . . . 20
  - 2.2.1 Deep Learning for RF Fingerprinting . . . 20
  - 2.2.2 GNU Radio . . . 24
    - 2.2.2.1 Zigbee Signals . . . 25
    - 2.2.2.2 Signal Capture . . . 25
    - 2.2.2.3 Signal Processing . . . 27
    - 2.2.2.4 Integration . . . 28

3 Development 30

3.1 System Overview . . . 30
3.2 GNU Radio Module . . . 31
3.3 GNU Radio and ML Module Integration . . . 36
3.4 Machine Learning Module . . . 38
3.4.1 Training Mode . . . 39
3.4.2 Classification Mode . . . 42
3.5 User Interface . . . 43
3.5.1 Technologies and Implementation Challenges . . . 44
3.5.2 Training Mode . . . 46
3.5.3 Real-Time Classification Mode . . . 48

4 Results **52**
4.1 Analysis of Key Performance Metrics . . . 52
4.2 Training Significance in System Performance . . . 55
4.2.1 Testing Scenarios . . . 55
4.2.2 Training-Test Alignment . . . 57
4.2.3 Hardware Influence on System's Performance . . . 60
4.2.3.1 Antenna Performance Comparison . . . 60
4.2.3.2 Amplifier impact (LNA) . . . 62
4.2.4 Environmental Impact . . . 63
4.2.4.1 Performance Comparison between Both Environments . 64
4.3 Processing Time . . . 65
4.4 Performance Conclusions . . . 68

5 Conclusions **70**
5.1 Future Work . . . 71

Bibliography **73**

Appendices

A Results **79**
A.1 Training-Test Alignment . . . 79
A.1.1 Confusion Matrices for Scenario 1 . . . 79
A.1.2 Confusion Matrices for Scenario 2 . . . 80
A.1.3 Confusion Matrices for Scenario 3 . . . 81
A.1.4 Confusion Matrices for Scenario 4 . . . 82
A.2 Environment Impact . . . 83
A.2.1 Confusion Matrices for Scenario 1 . . . 83
A.2.2 Confusion Matrices for Scenario 2 . . . 84
A.2.3 Confusion Matrices for Scenario 3 . . . 86

List of Figures

2.1 Feature generation in the RSS-based scheme [5] . . . 9
2.2 System model of the RFF recognition scheme[22]. . . . 10
2.3 Application domains for Radio Frequency (RF) fingerprinting [24]. . . . 11
2.4 Differences between received signal and ideal signal [22]. . . . 12
2.5 3 devices fingerprint after processing [26]. . . . 15
2.6 3 devices fingerprint after Linear Discriminant Analysis (LDA) processing [26]. 15
2.7 Neuron [45]. . . . 21
2.8 Convolutional Neural Network (CNN) Architecture for Image Classification [45]. . . . 21
2.9 Recurrent Neural Network (RNN) architecture [45]. . . . 22
2.10 Long Short-Term Memory (LSTM) architecture [47]. . . . 22
2.11 Autoencoders (AE) architecture [49]. . . . 23
2.12 Generative Adversarial Network (GAN) architecture [45]. . . . 24
2.13 PPDU structure [52]. . . . 25
2.14 Transceiver Architecture [54]. . . . 26
2.15 Minimum Shift Keying (MSK)-based receiver [55]. . . . 26
2.16 Suggested Improvements to the MSK-based Receiver [55]. . . . 27
2.17 Modified transceiver [34]. . . . 28

3.1 System Overview . . . 31
3.2 Iris Devices . . . 31
3.3 Nuand bladeRF . . . 32
3.4 Gnu Radio Receiver Module . . . 33
3.5 Gnu Radio Script . . . 33
3.6 State machine diagram . . . 36
3.7 ZMQ Push Sink . . . 37
3.8 Structure of the name of each dataset . . . 39
3.9 DataBase Generation Script . . . 40
3.10 Convolutional 1D taken from [59] . . . 41
3.11 In train Dataframe . . . 41

3.12 Overview of the threading architecture within the application 45
3.13 Graphical User Interface (GUI) menu 46
3.14 Scenario details 47
3.15 Training Mode Interface 47
3.16 Classification Interface 49
3.17 System's prediction Interface 50

4.1 Binary Confusion Matrix 53
4.2 MultiClass Confusion Matrix 54
4.3 Static Scenario 56
4.4 Dynamic Scenario 56

A.1 Confusion Matrix for the 15 cm static training environment. 79
A.2 Confusion Matrix for the 1 meter static testing environment. 80
A.3 Confusion Matrix for the 15 cm static training environment. 80
A.4 Confusion Matrix for a 1m dynamic testing environment 81
A.5 Confusion Matrix for a 1m static training Environment 81
A.6 Confusion Matrix for a 15cm dynamic training Environment 82
A.7 Confusion Matrix for a 1m static training Environment 82
A.8 Confusion Matrix for a 15cm dynamic testing environment 83
A.9 Confusion Matrix for Test 1. 83
A.10 Confusion Matrix for Test 2. 84
A.11 Confusion Matrix for Test 1. 84
A.12 Confusion Matrix for Test 2. 85
A.13 Confusion Matrix for Test 3. 85
A.14 Confusion Matrix for Test 1. 86
A.15 Confusion Matrix for Test 2. 86
A.16 Confusion Matrix for Test 3. 87

Acronyms

AE Autoencoders *(pp. iv, ix, 23, 38)*
ANA Artificial Noise Adding *(p. 13)*
ANN Artificial Neural Networks *(pp. 22, 23)*
AWGN Additive White Gaussian Noise *(pp. 12, 13)*

BPSK Binary Phase Shift Keying *(p. 25)*

CFO Carrier Frequency Offset *(pp. 16, 18)*
CFR Channel Frequency Response *(pp. 8–10)*
CI Confidence Interval *(p. 65)*
CIR Channel Impulse Response *(pp. 8–10)*
CNN Convolutional Neural Network *(pp. ix, 19, 21)*
Conv1D One-Dimension Convolutional *(p. iv)*
CSI Channel State Information *(pp. 8–10)*

DAE Denoising Autoencoder *(p. 23)*
DNN Deep Neural Network *(p. 19)*
DoS Denial-of-service *(p. 4)*

FNN Feedforward Neural Network *(pp. 20, 21)*

GAN Generative Adversarial Network *(pp. ix, 24)*
GRC GNU Radio Companion *(pp. 24, 26, 28, 29)*
GRLVQI Generalized Relevance Learning Vector Quantization-Improved *(p. 17)*
GRU Gated Recurrent Unit *(p. 38)*
GUI Graphical User Interface *(pp. x, 2, 24, 25, 28, 30, 31, 43, 46)*

I/Q In-phase/Quadrature *(pp. 16, 18, 19)*
IoT Internet of Things *(pp. iv, 1, 4, 5)*

k-NN k-Nearest Neighbors *(p. 18)*

LDA Linear Discriminant Analysis *(pp. ix, 15)*
LNA Low-Noise Amplifier *(p. iv)*
LPF Low Pass Filter *(p. 27)*
LSRS Long-Term Stacking of Repetitive Symbols *(p. 12)*
LSTM Long Short-Term Memory *(pp. ix, 19, 22, 23)*

MDA/ML multiple Discriminant Analysis Maximum Likelihood *(p. 17)*
ML Machine Learning *(pp. iv, 18, 28–31, 36)*
MLP Multilayer Perceptrons *(pp. 20, 66)*
MSCNN Multisampling Convolutional Neural Network *(p. 19)*
MSK Minimum Shift Keying *(pp. ix, 26, 27)*

NIC Network Interface Card *(p. 16)*
NN Neural Network *(pp. iv, 38)*

OFDM Orthogonal Frequency-Division Multiplexing *(p. 16)*
OHE One Hot Encoder *(p. 40)*
OQPSK Offset Quadrature Phase Shift Keying *(pp. 25, 26)*

PHR PHY Header *(p. 25)*
PHY Physical Layer *(pp. iv, 1, 2, 7, 25, 26)*
PLA Physical Layer Authentication *(pp. 1, 4, 5, 7, 8, 10)*
PPDU PHY Protocol Data Unit *(p. 25)*
PSDU PHY Service Data Unit *(p. 25)*

RF Random Forest *(pp. iv, 38–42, 66)*
RF Radio Frequency *(pp. ix, 1, 2, 5, 7, 10–12, 14, 15, 17–20, 24, 32)*
RF-DNA Radio Frequency Distinct Native Attribute *(pp. 17, 32)*
RFF Radio Frequency Fingerprinting *(p. iv)*
RNN Recurrent Neural Network *(pp. ix, 19, 22, 23)*
ROI Region of Interest *(p. 17)*
RSS Received Signal Strength *(pp. 8–10)*
RSSI Received Signal Strength Indicator *(p. 9)*

SD Spectral Domain *(p. 17)*
SDR Software Defined Radio *(pp. iv, 1, 2, 11, 24, 31, 32, 70)*
SFD Start Frame Delimiter *(p. 25)*

SHR Synchronization Header *(p. 25)*
SNR Signal Noise Ratio *(pp. 13, 16, 19)*
STFT Short-Time Fourier Transform *(p. 18)*
SVM Support Vector Machine *(p. 16)*

TD Time Domain *(p. 17)*

USRP Universal Software Radio Peripheral *(pp. 12, 27)*
UWB Ultra WideBand *(p. 17)*

1 Introduction

1.1 Background

The rapid advancement of the IoT is revolutionizing aspects of daily life, from smart homes and healthcare to industry 4.0 and urban infrastructure [2]. Cisco's projection that there will be between 25 billion and 125 billion connected IoT devices globally by 2030 illustrates IoT's significant impact and widespread integration in sectors like energy, transportation, and manufacturing [3].

This widespread adoption of IoT devices, however, raises significant security concerns. Traditional security mechanisms often rely on upper-layer cryptographic methods, which, while technically feasible, present vulnerabilities to hardware tampering and reverse engineering. These risks highlight the necessity of robust security measures to safeguard against unauthorized access and misuse.

In response to these challenges, Physical Layer Authentication (PLA) has gained attention as a viable security enhancement. PLA leverages the unique characteristics of a device's analog circuitry for authentication purposes, a concept central to RF fingerprinting. RF fingerprinting utilises identifiable information from transmission imperfections inherent to the devices. This approach has been extensively investigated as a promising solution for IoT security, offering an energy-efficient and difficult-to-replicate method without adding extra complexity or requirements to the devices.

1.2 Objectives

The primary aim of this thesis is to design and test real-time machine learning algorithms for PHY authentication using SDR.

To achieve this main goal, the objectives are broken down into the following essential tasks:

- Study existing machine learning algorithms and explore new ones capable of real-time.

- Develop a new GNU Radio module for capturing ZigBee frames.
- Construct and optimize a testbed for real-time classification.
- Create a graphical interface that makes it easy to train models, add new sensors and test in real time.
- Write a detailed and well-structured dissertation documenting the research and findings.

1.3 Contributions

This thesis presents several contributions in the field of real-time authentication at the PHY using SDR and Machine Learning. Among the main contributions are:

– Modifications and improvements have been made to the GNU Radio module previously developed to capture and store the preamble of a frame.

– An integration solution between the GNU Radio module and the machine learning model was presented, including an unsuccessful attempt, before achieving the final functional integration.

– A detailed analysis of the impact of hardware changes on the performance of such a system was carried out and presented.

– Results of the system's performance in various scenarios were presented, assessing its reliability and accuracy in static and dynamic conditions, and at different distances;

– A GUI was developed to make it easier to train sensors and run tests in real time, making the system more accessible and intuitive to operate.

1.4 Document structure

This chapter provides an overview of the document structure. The subsequent chapters are organized as follows:

- **Chapter 2:** delves into the state of the art, offering a comprehensive examination of Physical Layer Authentication methods, extensive insights into RF Fingerprinting Approaches found in literature, and an in-depth look at Machine Learning self-extracted features. It also covers the application of deep learning models in RF fingerprinting, the process of signal acquisition using GNU Radio, and its integration with machine learning environments.

- **Chapter 3:** describes the system architecture, including a detailed presentation of the GNU Radio module responsible for signal processing, the machine learning module that implements models capable of running in real time, and finally, it addresses the developed application that integrates all of this into a simple and practical environment for training and testing classification scenarios.
- **Chapter 4:** presents the results obtained by the system, highlighting the importance of training and its influence on system performance, the impact of hardware changes on system elements, the effect of the testing environment, and finally, an analysis of the system's reliability.
- **Chapter 5:** provides a conclusion of the work completed and addresses potential future work and improvements.

2

State of Art

2.1 Physical Layer Authentication

Due to the exponential growth of the 'Internet of Things', it is imperative to ensure the security of all connected devices. Securing the IoT devices poses unique challenges since the inherent broadcast nature of wireless communications raises several security and privacy issues, including IP or MAC spoofing and Denial-of-service (DoS) attacks. Most wireless communications guarantee their authentication and confidentiality through the upper layers using mechanisms based on cryptography, which means that computational complexity and secret keys play a significant role in the security of cryptography-based techniques. Although implementing these mechanisms in IoT devices is technically feasible, it introduces potential vulnerabilities, such as the risks of hardware tampering and reverse engineering. These vulnerabilities pose significant threats to the security of authentication keys.

PLA has been extensively researched in recent years due to its utilization of the inherent hardware features of devices to ensure device authentication. Essentially, PLA aims to confirm that a user or device is indeed who or what it claims to be. Wireless physical layer authentication verifies whether a wireless transmitter is real by examining physical traits in communication. This type of identification at the physical layer uses tiny differences in the transmitted analog signal to recognize a device. These differences happen due to flaws in the analog parts of a radio transmitter.

Consequently, PLA can be viewed as an additional layer of security that complements higher-layer authentication schemes for authenticating IoT devices without imposing additional power costs on the devices themselves.

2.1.1 Traditional cryptographic techniques

Cryptography involves concealing or encoding information to ensure only the intended recipient of a message can comprehend its content. Typically, authentication revolves around a handshake process between a device seeking access and the network element verifying authentication messages. Imagine a scenario where Alice and Bob aim to

communicate securely over a wireless network. To authenticate their exchange, they both agree on a shared secret key, S. Alice encodes her message using S and transmits it across the open wireless medium. Upon reception, Bob uses S to decode and access the message. In this process, the confidentiality of S acts as a shield, ensuring only Alice and Bob possess the means to interpret the transmitted data. While traditional cryptographic methods have the potential in safeguarding wireless networks against identity-based attacks, they face limitations in various scenarios, such as insufficient protection in management and control frames, computational complexity, and challenges in key management [4]. This underscores the pressing need for authentication and security methods that demand reduced processing power and energy consumption.

In contrast, PLA non-cryptography methods, being hardware-specific and typically inadvertent characteristics at the analog component level, pose challenges in replication. This uniqueness results in a robust authentication approach that is difficult to replicate, thus enhancing its resilience against unauthorized access attempts compared to the traditional reliance on secret keys.

2.1.2 PLA schemes in the literature

In the search for effective physical layer authentication mechanisms for IoT devices and wireless networks, different approaches that differ in their operations and fundamental principles have emerged. Thus, we can separate PLA schemes into two main categories: passive and active[5].

Passive authentication methods rely on the received signal's physical characteristics to authenticate between the receiver and the transmitter. These characteristics pertain to the device's components, including controllers, circuits, clocks, and other elements. Analyzing these components is commonly referred to as RF Fingerprinting, as its purpose is to generate signatures derived from the signal's unique characteristics[6]. Later on, there will be a section where we will discuss in greater detail passive methods, specifically RF Fingerprinting.

On the other hand, active methods are characterised by the transmitter embedding a tag that has to be decoded by the receiver, thus altering the original message. One of the main differences between passive and active methods is whether or not the original message is modified.

Typically, robustness, security, and covertness are the three main qualities that must exist when designing a PLA scheme. According to [7], robustness refers to the need for the PLA to ensure that authentication performance is not disturbed by channel fading and noise interference. Guaranteeing the scheme's security means ensuring that any attack on the system does not influence authentication. Finally, guaranteeing covertness means that the system doesn't overload the transmission of information, avoiding an increase in overall communication costs.

As far as passive schemes are concerned, since they don't alter the original message's

content, they only need to guarantee security and robustness. Alternatively, considering that active methods typically embed a tag within the source message, ensuring covertness is essential.

2.1.3 Active Methods

In active methods, embedding a tag in the original message can compromise the decoding of the message on the receiver's side. Therefore, considering covertness is essential. Typically, active schemes comprise two phases: firstly, the phase where a key is established, and secondly, the phase of message transmission. In the first phase, the transmitter generates a tag based on a secret key exchanged between them. In the second phase, the receiver authenticates the transmitter by verifying whether this tag was embedded in the message.

Moreover, [5] categorizes active methods into subcategories, distinguishing between non-covert and covert approaches, thus providing a more detailed classification within active authentication techniques.

2.1.3.1 Non-covert schemes

When it comes to non-covert schemes, their limited applicability arises from the absence of consideration for covertness during their design, consequently restricting their usage to receivers who are aware of this limitation [8].

A non-covert scheme is proposed in [9], wherein Supangkat *et al.* embedded a tag into a source message using spread spectrum modulation in a telephone system, focusing on the modification of the frame structure.

The study detailed in [8] presents a non-covert system proposed by the authors. The proposed system involves changing an initial part of the bits within the context of modifying the transmission parameters.

In [10], the author proposes a non-covert scheme based on encrypting the original message. This scheme employs an encoding approach for the source message to defend against impersonation and substitution attacks in a noiseless channel. In practice, Alice sends a codeword comprising the secret key and the source message. Using the secret key, Bob authenticates by verifying whether the received codeword originates from Alice.

2.1.3.2 Covert schemes

Covert schemes, in stark contrast to their non-covert counterparts, prioritize the consideration of covertness during their design phase. The requirement for covert attributes emerges when authentication is integrated into an established system. An unaware receiver can authenticate a transmitting device using a specific upper-layer authentication technique, even while she is unaware of the specifics of the active scheme. On the flip side, an informed receiver creates a two-factor authentication scheme for increased security

in addition to authenticating the sending device. Furthermore, increased covertness frequently translates into increased security, making it harder for attackers to identify the tag in their observed signals. This means that covert authentication in the PHY layer can be used with other security techniques in the upper layer to provide a more secure system. There are numerous applications of this type of scheme in current literature. Recently, the prevailing active-style authentication scheme involves the concurrent transmission of an additional authentication tag, superimposed onto the messages [11][12].

In [11], the author presents a comprehensive framework for analyzing and designing authentication at the physical layer. This paper introduces a technique for transmitting authentication data alongside the main data stream. Authentication is added to the signal without more bandwidth, as do spread-spectrum approaches, by superimposing a properly planned secret modulation on the waveforms. In the detailed study in [12], a superimposition scheme for PHY authentication is introduced, relying on tag embedding and tag verification.

Another covert implementation strategy for active schemes with covertness involves replacing specific bits of the source message with corresponding bits of the tag, as proposed in the scheme outlined in [13].

2.1.4 Passive Methods

This section will explore passive authentication methods, categorizing them into two main groups: radio frequency fingerprint-based and channel-based schemes [14]. Our focus will be on delineating these two categories before delving into detailed discussions.

In the context of Passive PLA methods, RF Fingerprinting emerges as a central technique. This method capitalizes on the intrinsic characteristics of RF signals emitted by wireless devices. Each device possesses a unique signature, known as the RF Fingerprint, arising from subtle variations in hardware components, circuits, and the communication environment.

Channel-based schemes analyze communication characteristics like signal strength, timing, and frequency response. In contrast, RF Fingerprinting functions as signal intelligence, extracting the unique characteristics of the transmitter's hardware from the transmitted waveform. This enables passive receivers to identify transmitter hardware without altering the signal.

Following this categorization, our review will encompass the typical schemes, applications, and methodologies associated with RF Fingerprinting, highlighting its significance and applications within wireless communication systems.

2.1.4.1 Channel based

Channel-based authentication methods rely on the unique characteristics inherent in wireless channels, encompassing spatial variability, temporal fluctuations, and reciprocal

behaviour. These attributes distinctly vary between transmitters and receivers across diverse spatial locations, forming the basis for physical layer authentication.

The core principle of channel-based schemes involves authenticating the source of received signals by leveraging their location-specific traits. The security foundation of these methods rests on the premise that as transmitters move apart spatially, the correlation between their channel-based attributes diminishes significantly. These location-specific features within the wireless channel serve as intrinsic markers for authentication. Their inherent uncontrollability bolsters their significance in authentication processes, offering inherent uniqueness that cannot be intentionally manipulated.

Within PLA methods, distinguishing between authorized and unauthorized nodes is achieved through analyzing channel traits, including aspects like the Received Signal Strength (RSS), the Channel State Information (CSI), the Channel Impulse Response (CIR), and Channel Frequency Response (CFR).

Received Signal Strength

RSS is a metric used in wireless networks to measure the intensity of the signal received by a device. RSS can be acquired by converting the power readings received at the physical layer of the IEEE 802.11 platform [15]. In this paper, they consider an RSS-based authentication system to identify the transmitters, relying on a mathematical expression derived from [16] that demonstrates the RSS value between two nodes can be obtained using the following expression:

$$P(d) = P(d_0) - 10\alpha \log_{10}\left(\frac{d}{d_0}\right) + S, \tag{2.1}$$

where d is the distance between the nodes, $P(d_0)$ represents the transmission power, α is the path loss exponent and S is a zero-mean Gaussian noise.

There are several ways in which the study of RSS can be used to ensure authentication [5]. For example, in [17], the authors propose an authentication scheme based on comparing the estimated RSS with the real one. This method is designed to protect against both impersonation and Sybil attacks. This study presents an example of RSS-based feature generation as shown in Figure 2.1. Briefly, at Bob's location, where there are N Access Points (APs), Bob extracts N RSS values represented as $R(i), i = 1, ..., N$. Subsequently, he constructs statistical tests $T_i = \sum_{i=1}^{N} |R_X(i) - R_A(i)|$, where $R_X(i)$ and $R_A(i)$ denote the estimated and legitimate ith RSS values, respectively. These statistical tests are designed to discern differences between the estimated and legitimate RSS values. A high value of the test statistic, in the case of two received signals sharing the same MAC address, leads Bob to conclude an impersonation attack. Conversely, a low value of the test statistic, with two received signals from different MAC addresses, leads Bob to conclude a Sybil attack.

Another approach in utilizing RSS for authentication is through RSS ratio-based methods, as described in [18]. In this paper, the authors propose a solution using

Received Signal Strength Indicator (RSSI)-based techniques to counter the Sybil attack, demonstrating that despite radio transmission's general variability and non-isotropic nature, leveraging the ratio of RSSI obtained from multiple receivers effectively mitigates these challenges.

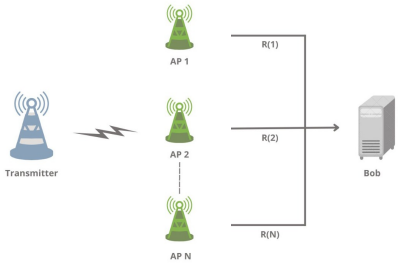

Figure 2.1: Feature generation in the RSS-based scheme [5]

Channel State Information

The CSI comprises detailed information on communication channel conditions, offering insights such as amplitude, phase, and other parameters. Contrasting with the RSS, which provides a simpler signal intensity measure, CSI provides a more comprehensive and intricate view of the channel. However, obtaining an accurate channel estimation using CSI involves more complex processing and demands greater computational resources. Additionally, there are other important metrics within CSI, like CIR and CFR.

The CIR refers to the temporal behavior of a communication channel. It describes how the signal propagates through the channel, enabling an understanding of signal dispersion, delays, and potential distortions over time. Essentially, the CIR provides insights into how the signal spreads over time within the channel. In [19], the author introduced an enhanced CIR-based authentication scheme that leverages both the amplitude and multipath delay of the CIR to bolster robustness, especially in mobile scenarios.

The behavior of a communication channel in the frequency domain is referred to as the CFR. It details the channel's response to various frequencies, showing how it behaves regarding signal phase and magnitude at different frequencies. In essence, the CFR helps to explain frequency-selective fading and distortions by providing insights into how the channel impacts various frequency components of the transmitted signal. Reference [20] studies the use of a CFR-based scheme in a time-invariant channel. More precisely, the author proposed a CFR-based scheme where Bob estimates the CFR vector from Alice and the CFR vector from an unknown transmitter. Authentication of the transmitter occurs by comparing these estimated CFR vectors.

To sum up, channel-based schemes rely on location-specific features to authenticate signal origins, underpinning their security on the diminished correlation between these attributes as transmitters distance themselves from each other. It is essential to highlight

that RSS and CSI have been research subjects for physical layer authentication due to their temporal correlation within the propagation environment. Nonetheless, the effectiveness of RSS-based authentication is restrained by channel stability and communication noise.

Efforts to improve reliability have explored additional channel features, such as CFR. However, using CFR for authentication encounters challenges, including heightened complexity in implementing broadband systems and the need for more spatial information regarding signal propagation. As an alternative, investigations into time-domain channel characteristics have also aimed to enhance the authentication process [19].

In comparison with PLA schemes relying on instantaneous channel information features like CSI, CIR, or CFR, PLA schemes based on statistical channel information features such as RSS offer lower complexity due to their ease of estimation. Yet, this ease of estimation poses a security risk since these features are also easily estimable by an eavesdropper. Moreover, PLA schemes based on statistical channel information features demonstrate better robustness than those relying on instantaneous channel information features, as they are less affected by changes in communication environments.

2.1.4.2 RF fingerprint based

As previously mentioned, RF Fingerprinting has garnered significant attention over the years due to its utilization of inherent hardware characteristics for identification and verification purposes. This approach capitalizes on the distinctive attributes of devices, employing individual hardware signatures for authentication and validation. As mentioned in [21], wireless signals derive RF fingerprints resembling human fingerprint biometrics. In this section, we will delve deeper into what a typical RF Fingerprinting scheme looks like, its applications, challenges, and approaches.

Typical RF fingerprint scheme

In terms of the typical RF fingerprint scheme, it usually involves three primary components, such as data acquisition, feature extraction, and recognition [22]. In this paper, the authors illustrate the structure of an RF fingerprinting scheme, as depicted in Figure 2.2

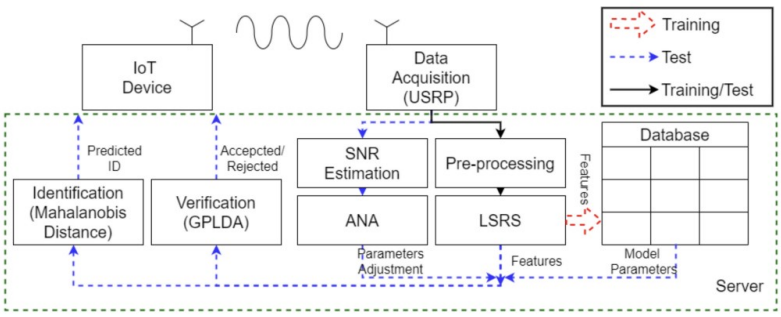

Figure 2.2: System model of the RFF recognition scheme[22].

The initial phase involves **data acquisition**. Lately, there has been a shift in RF fingerprinting research, with an increasing reliance on affordable SDR equipment instead of the previously predominant high-end oscilloscopes or spectrum analyzers. After data acquisition, the next step typically involves a feature **extraction module**. This process can be carried out in various ways, depending on the approach: modulation-based, transient-based, statistical-based, and so on. Further sections will delve into the detailed functioning of each approach and review the existing literature related to each approach.

Following feature extraction, the presence of a **recognition** module becomes crucial as it handles the identification of a device.

Applications

In terms of applications, RF fingerprint schemes have gained increasing popularity due to their versatility and utility across various fields as depicted in Figure 2.3.

IoT devices proliferate and raise concerns about security across various sectors, prompting the emergence of RF fingerprinting as a pivotal application. Recent literature emphasizes the contribution of RF fingerprinting to bolstering system resilience. People increasingly acknowledge its role in enhancing security and fortifying systems against potential threats. Another domain witnessing the growing prominence of RF fingerprinting is the realm of localization, navigation, and tracking. An example is the study by [23], where the authors utilized RF fingerprinting to track illicit mobile phones smuggled into a prison. An exciting application demonstrated in [24] utilizes RF fingerprinting for intrusion detection. Before a squadron sets out on a mission, each member could hold RF fingerprint information about their fellow group members. This approach empowers each device to detect the presence of an intruder not listed in the signature database. This paper discusses the impact and potential applications of RF fingerprinting in 6G technologies. Intelligent Telehealth, Autonomous UAV and V2X, Smart Grid, and Extended Reality are highlighted as domains within the scope of 6G where RF fingerprinting can be applied.

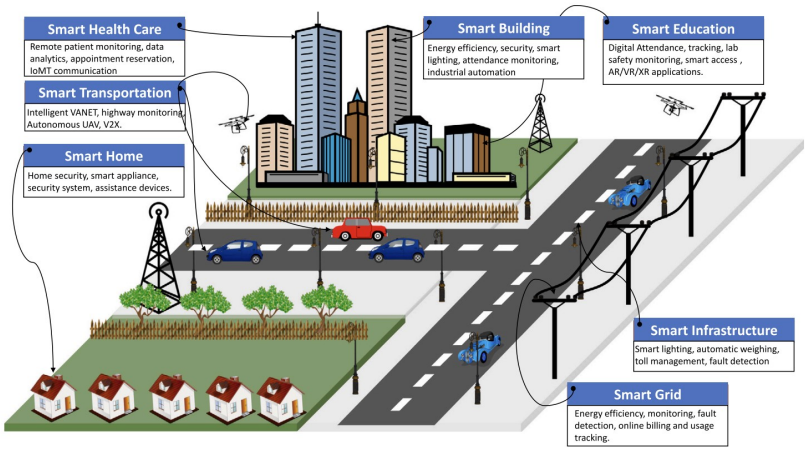

Figure 2.3: Application domains for RF fingerprinting [24].

Feature Extraction

In this segment, we will explore an instance of the feature extraction process, beginning with a discussion on RF generation systems and the signal model. In [22], the authors propose a scheme where the Zigbee Devices and the Universal Software Radio Peripheral (USRP) are positioned close to each other. However, the differences between the original signal and the received signal are visible in the Figure 2.4. Certainly, these discrepancies arise from hardware imperfections and channel interferences. Due to the short distance between the two, the channel is typically modeled as an Additive White Gaussian Noise (AWGN) channel. Consequently, device identification can be achieved by deriving hardware characteristic coefficients using specific nonlinear regression models [25].

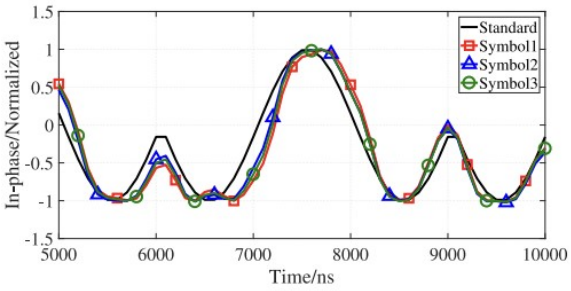

Figure 2.4: Differences between received signal and ideal signal [22].

However, the imperfections observed can be seen as a combination of the characteristics of circuit elements and device noise. Consequently, we can model the received signal as:

$$y(t) = y'(t) + f(t) + n(t), \tag{2.2}$$

where $y'(t)$ is the ideal signal, and $y(t)$ is the received signal, $n(t)$ is the AWGN channel noise with a mean of 0 and a variance of δ^2, and $f(t)$ is the device fingerprint [22].

In [22], the authors propose a long-term stacking of repetitive symbols algorithm aimed at mitigating the impact of both channel noise and device noise, facilitating the comparability of the device fingerprint.

Long-Term Stacking of Repetitive Symbols Algorithm

In many wireless communication systems, fixed repetitive symbols are transmitted for synchronization. The Long-Term Stacking of Repetitive Symbols (LSRS) algorithm aims to mitigate the interference caused by temporal, environmental, and device-related noise, while the stacking process aids in reducing variance, resulting in a more stable fingerprint.

After the symbols are stacked we get the following result:

$$LSRS(y(t)) = \sum_{i=0}^{N-1} b_i y(t + P_i T), \quad 0 < t \leq T, \\ \sum_{i=0}^{N-1} b_i = 1, \tag{2.3}$$

where N represents the number of stackings, P_i denotes the placement of the ith target symbol, and b_i stands for the stacking weight. In [22], the authors consider $b_i = (1/N)$ which resulted in

$$\begin{aligned} LSRS(y(t)) = \frac{1}{N} \sum_{i=0}^{N-1} y(t + P_i T) &= \\ = x(t) + \frac{1}{N} \sum_{i=0}^{N-1} f(t + P_i T) + \frac{1}{N} \sum_{i=0}^{N-1} n(t + P_i T) &= \\ = x(t) + f'(t) + n'(t). \end{aligned} \tag{2.4}$$

The final feature vector F in discrete form is

$$\begin{bmatrix} \text{real}\left(LSRS(y(T_s))\right) & \text{img}\left(LSRS(y(T_s))\right) \\ \text{real}\left(LSRS(y(2T_s))\right) & \text{img}\left(LSRS(y(2T_s))\right) \\ \vdots & \vdots \\ \text{real}\left(LSRS(y(T))\right) & \text{img}\left(LSRS(y(T))\right). \end{bmatrix} \tag{2.5}$$

This paper also concluded that the number of stacked symbols influences the variance. More specifically, as N increases, the distribution becomes more concentrated and has a lower variance.

Artificial Noise Adding Algorithm

Adding artificial noise in communication systems aims to bolster performance in time-variant channels and AWGN environments. It helps minimize differences within time-varying channels by amplifying variance and aligning the Signal Noise Ratio (SNR) between training and test sets in AWGN channels. Nonetheless, excessive artificial noise can be detrimental to system performance.

In both cases [26] and [22], the authors propose the incorporation of Artificial Noise Adding (ANA) based on the understanding that when the training and test sets share similar channel conditions, the recognition scheme performs notably better by utilizing similar channel information.

In [26], the authors introduced noise into the system with an SNR close to the original signal. With their collected data operating at around 26 dB, they deliberately added 30 dB of noise. The process of ANA can be described as:

$$y(t) = y'(t) + f(t) + a(t) + n(t), \tag{2.6}$$

where $a(t)$ is artificial noise.

Feature Processing

Once we have the fingerprint generation model and obtain a vector of features, prior to data visualization or classification, the features require processing for effective organization and identification. In this section, we will delve into the application of Mahalanobis Distance as found in current literature.

Mahalanobis Distance

Based on Bayes' rule [27],

$$P(k|F) = \frac{P(F|k)P(k)}{P(F)}, \tag{2.7}$$

the typical discriminant function derived from Bayes' rule is given by

$$g_k(F) = \ln P(F|k) + \ln P(k). \tag{2.8}$$

In [26], as the devices in the study are similar and send the same data, they are assumed to have identical covariance. With this, the discriminant function is rewritten as:

$$g_k(F) = -\frac{1}{2}(F - \mu_k)^T \Sigma^{-1}(F - \mu_k) + \ln p(k). \tag{2.9}$$

Let's assume an equal probability for each class,

$$p(k) = \frac{1}{N_d}, \tag{2.10}$$

where N_d is the number of devices. Therefore, the goal is to minimize the Mahalanobis distance between μ_k stored in the local database and the received RF fingerprint of the data. Thus, it follows from [22] that the Mahalanobis distance is given by

$$\text{Mah}_{F_{\text{test},i},\mu_k} = (F_{\text{test},i} - \mu_k)^T \Sigma^{-1}(F_{\text{test},i} - \mu_k). \tag{2.11}$$

RF Fingerprint Classification

In terms of data classification, numerous algorithms can be applied based on the specific needs of the problem. Later in the RF fingerprint approaches section, we will have the opportunity to review what the current literature proposes. Thus, here, we will draw upon a framework presented in the previous section based on [26].

Essentially, the authors delineate their scheme into two main phases: the **training phase** and the **authentication phase**. During the training phase, the mean μ_k and covariance for each device were estimated and placed in the database. Subsequently, in the authentication phase, the Mahalanobis distance is calculated to find the closest mean μ_k. After processing, the following fingerprint was obtained:

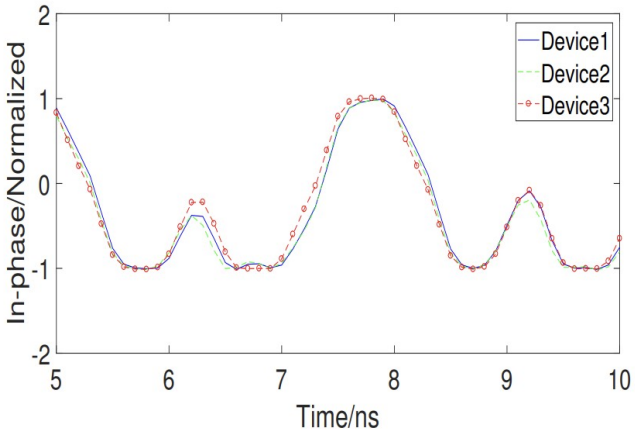

Figure 2.5: 3 devices fingerprint after processing [26].

This study also highlights the significance of employing LDA to enhance fingerprint visualization. LDA can be applied in this case to reduce the dimensionality of fingerprint features, facilitating the process of identifying differences between devices, as shown in Figure 2.6.

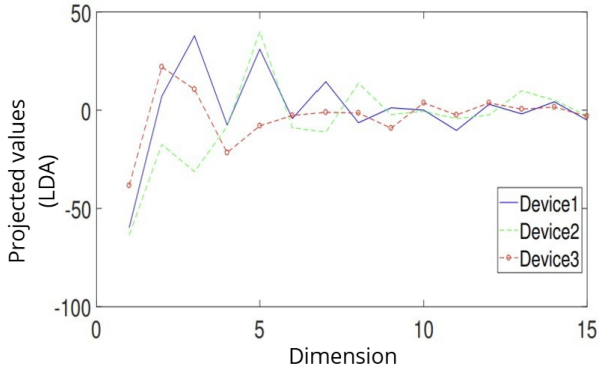

Figure 2.6: 3 devices fingerprint after LDA processing [26].

2.1.4.3 RF Fingerprinting Approaches in Literature

After discussing the typical constitution of an RF fingerprint scheme, its applications, feature extraction, processing, and some associated algorithms, it is now opportune to delve into the existing approaches to RF fingerprinting in the literature.

We will first begin by categorizing these approaches into different classifications, including modulation-based, statistical-based, transient-based, and other approaches. Each subsection will discuss the typical features to extract within each approach alongside the associated classification methods and algorithms.

Modulation Based

Approaches utilizing modulation for device identification focus on extracting distinctive attributes from the modulated segment of the signal, typically the data component.

These distinctive features, such as **phase noise**, Carrier Frequency Offset (CFO), **clock offset**, **clock skew**, **phase noise**, **imperfect power amplifier**, and **In-phase/Quadrature (I/Q) imbalance**, have gained prominence in device identification techniques, emerging relatively recently in this realm [28]. It is essential to highlight that while authentication using a single feature is feasible, employing multiple features can significantly enhance accuracy, though at a higher resource cost [5].

Moreover, among modulation-based identification methods, there is a specific approach known as the **PARADIS**. In their work on PARADIS, referenced as [29], the authors introduce this methodology as the Passive Radiometric Device Identification System. PARADIS capitalizes on subtle discrepancies in transmitter hardware acquired during manufacturing and persisting among identical Network Interface Card (NIC)s. These unique imperfections in each transmitter manifest as distinctive attributes within the transmitted signals. By meticulously analyzing these individual wireless frames' unique signal artifacts and employing machine-learning classification techniques, the study concluded that PARADIS excelled in distinguishing among more than 130 identical 802.11 NIC with an accuracy surpassing 99%.

One of the essential features of modulation-based authentication is the **I/Q imbalance**. I/Q imbalance arises from imperfections in the hardware of the I/Q quadrature modulator [24]. In [30], a method was developed to leverage the inherent I/Q imbalance within quadrature modulation signals for extracting RF fingerprint features. This method investigated the origins and signal impact of radio frequency fingerprinting using an I/Q modulator model. The outcome of this approach was a fingerprint feature vector demonstrating an accuracy exceeding 90% for SNR $\geq$ 15 dB and surpassing 99% for SNR $\geq$ 20 dB. Furthermore, in the referenced article, the fingerprint feature vector was extracted using the proposed method, and an Support Vector Machine (SVM) classifier was trained using half of the dataset.

The authors in [31] introduced a physical layer authentication method designed for Orthogonal Frequency-Division Multiplexing (OFDM) systems. This scheme relies on the distinct carrier CFO between each transmitter-receiver pair. These individualized CFO values are unique signatures linked to each transmitter, facilitating their authentication.

Statistical Based

In statistical approaches, effectively identifying devices depends on the paramount importance of feature extraction. Leveraging statistical techniques, we meticulously select and transform device features, preserving similarities among akin objects and emphasizing differences between dissimilar ones [32]. This nuanced approach substantially enhances the discriminative power of these features, laying a crucial foundation for implementing robust statistical-based methods.

In the realm of statistical approaches, we categorize them into two primary groups: parametric and non-parametric. These categories guide the selection of features for device identification. Non-parametric features like **mean**, **mode**, and **median** are chosen for their robustness across diverse datasets, capturing central tendencies without assuming specific data distributions. Conversely, parametric features such as **variance**, **skewness**, and **kurtosis** rely on distribution assumptions, allowing for detailed characterization of data variability.

In [33], the author explores the statistical distribution of a specific Region of Interest (ROI). The study shows that most collected signals exhibit multi-modal or non-parametric distributions. The author suggests using non-parametric methods for feature generation, including mean, median, mode, and trend represented by linear model coefficient estimates. These methods are better suited for the non-parametric distribution of the collected ZigBee preamble signal, which enhances device classification performance. The article also compares classification performance between parametric and non-parametric features using the Random Forest classifier.

Another notable subfield within statistical-based device identification is Radio Frequency Distinct Native Attribute (RF-DNA). RF-DNA features can be generated from Time Domain (TD) or Spectral Domain (SD) responses [34]. Statistical RF-DNA fingerprints are created based on statistical properties of the received signal. These properties are derived from three key aspects of the signal response: its **instantaneous amplitude**, **instantaneous phase**, and **instantaneous frequency**. **Variance (σ^2), Skewness (γ), and Kurtosis (κ)** are then computed for each of these response elements, forming the essential features of the fingerprint [35].

The study presented in [36] proposes RF-DNA-based RF fingerprinting to identify Ultra WideBand (UWB) noise radar devices. They extract features such as variance, skewness, and kurtosis from the time-domain response and normalized PSD and Gabor transform from the spectral-domain response. Classification employs multiple Discriminant Analysis Maximum Likelihood (MDA/ML) and Generalized Relevance Learning Vector Quantization-Improved (GRLVQI) methods. Notably, using a combination of time and frequency domain features, the method achieves high classification accuracies for three-class datasets with 97.65% and 93.79% for MDA/ML and GRLVQI classifiers, respectively.

Transient Based

Transient-based RF Fingerprinting techniques focus on identifying distinctive features during the activation or deactivation of transmitters. The short duration of these transients adds to the implementation cost of acquisition devices. Typically, the process involves transient detection, feature extraction, and classification, emphasizing unique characteristics found in radio turn-on transients, which include **amplitude**, **frequency**, **phase**, and **wavelet coefficients**.

In [37], the author employs transient amplitude to classify wireless devices, explicitly utilizing transient length and the variance of normalized amplitude as critical features. The authors in [38] propose a method for generating unique fingerprints by analyzing wavelets and applying an algorithm to extract the wavelet coefficients that characterize transient features. Additionally, a neural network is employed in this paper to assess whether the generated features are sufficient for radio transmitter identification.

In [39], the authors introduce a novel approach for RF fingerprinting of Bluetooth devices using features extracted from the energy envelope of transient signals. As for transient extraction, it was performed by analyzing amplitude variance, similar to the method employed in [37]. The energy envelope is derived by applying a spectrogram, defined as the squared magnitude of the Short-Time Fourier Transform (STFT). Ultimately, the RF fingerprint is created by extracting various features from the energy envelope curve. These features encompass the area under the normalized curve, the transient duration, the maximum slope observed in the transient curve, the kurtosis, skewness, and the variance of the transient envelope. A dataset of 300 signals was collected from each of the seven Bluetooth transceivers, from which RF fingerprint features were extracted. A k-Nearest Neighbors (k-NN) classifier with three nearest neighbours was employed to classify the feature vector. For classifier training, 50 signals from each of the seven devices were utilized, leaving 250 signals for testing. Remarkably, the proposed method achieved a classification accuracy of 99.9%.

Machine Learning Self-Extracted Features

Beyond the modulation-based, statistical-based, and transient-based RF fingerprinting categories previously discussed, the literature encompasses a variety of other approaches. In this section, we specifically focus on ML self-extracted features, an approach notable for its relevance and potential application in our thesis. This methodology employs machine learning algorithms to identify distinctive RF signal features automatically.

The significance of feature extraction in RF fingerprinting cannot be overstated, as it directly influences the efficacy of device identification. Conventionally, this process involved the utilization of hand-crafted features such as I/Q offset, CFO [31], and time-frequency statistics analysis [40]. These manual methods have demonstrated effectiveness in identifying a restricted number of devices. However, they face limitations in scalability

and generalization when applied to larger sets of devices. Creating these features also tends to be quite demanding, often requiring a lot of technical knowledge and hands-on experience. This is where the role of self-extracted features becomes pivotal. By leveraging machine learning algorithms, these features are automatically generated, bypassing the need for manual engineering.

As described in [41], the article introduces a Multisampling Convolutional Neural Network (MSCNN) framework for classifying ZigBee devices to automate and improve the RF feature extraction process for more accurate classification. The MSCNN innovatively combines feature extraction and classification into a unified process. It achieves this through its first layer, which contains multiple downsampling branches, each performing distinct transformations. This design allows the MSCNN to capture features at various time scales, enabling it to discern subtle, short-term changes and broader, long-term trends within the RF signals. Additionally, the MSCNN incorporates data slicing techniques to mitigate overfitting, a crucial enhancement for handling potential timing errors during preprocessing, particularly in low SNR scenarios. Ultimately, the MSCNN's ability to automate feature extraction across different time scales significantly enhances the classification of ZigBee devices.

In [42], the research introduces an innovative Deep Neural Network (DNN) framework, precisely a Siamese Network, for effective automatic feature extraction in the physical layer of IoT networks. This DNN comprises 1D convolution layers that extract local temporal features, batch normalization for enhanced training stability and speed, and max pooling layers for reducing data dimensionality and selecting crucial features. A Long Short-Term Memory (LSTM) layer is also incorporated to learn and understand the sequential distribution of these extracted features. The Siamese Network features two identical sub-networks, each outputting a distance metric to differentiate between inputs from various devices. If the inputs are from the same device, the network ideally outputs zero; otherwise, it indicates a maximum distance. This structure demonstrates a robust method for device identification through self-extracted features.

The research detailed in [43] investigates the application of deep learning models-DNN, CNN, and Recurrent Neural Network (RNN)-for automatic feature extraction in RF fingerprinting. These models effectively distinguish identical RF devices at various SNR levels using historical I/Q data from ZigBee devices. The study conducted tests with different window sizes and learning rates, finding that larger window sizes reduce accuracy in all models. Notably, LSTM, a variant of RNN, showed lower performance than DNN and CNN. This methodology capitalizes on the extensive availability of wireless big data, proving its effectiveness for precise ZigBee RF device identification and authentication.

2.2 Integration of ML and GNU Radio

2.2.1 Deep Learning for RF Fingerprinting

Deep learning has emerged as a promising approach in RF fingerprinting, offering innovative methods to extract characteristics of various RF devices by learning directly from their RF data. This surge in deep learning applications is primarily attributed to the recent revival of machine learning, propelled by the rapid growth in computational capabilities and the widespread availability of digital data. Deep learning algorithms, characterized by their multi-layered architectures and non-linear processing neurons, are adept at learning higher-level representations of input data [44]. They have been increasingly employed in research, setting a new standard for state-of-the-art techniques in this domain.

Experimental evidence has consistently demonstrated the high accuracy of deep learning methods in wireless device identification, presenting a potential to enhance IoT security significantly [43]. In this section, we will explore various deep-learning models presented in the literature, focusing on their applications and effectiveness in RF fingerprinting. While classical machine learning methods offer robust solutions for classification tasks, our focus will be primarily on deep learning models, given their growing prominence and adaptability in handling complex RF data.

Feedforward Neural Network (FNN)

Among the foundational models in deep learning, Feedforward Neural Network (FNN) stand out for their simplicity and effectiveness. FNN, also commonly referred to as Multilayer Perceptrons (MLP), are characterized by their sequential layer architecture where each layer consists of neurons that are fully connected to the neurons in the subsequent layer. These networks propagate data from the input to the output layer without loops or cycles in the architecture, hence the term **'feedforward'**.

A perceptron is a fundamental unit of a FNN and consists of neurons, each having a simple yet crucial structure, as illustrated in Figure 2.7. The operation of a perceptron involves three key steps. First, each input x is multiplied by its corresponding weight. Then, these products are summed to form a weighted sum. Finally, an activation function is applied to the sum, which determines the perceptron's output.

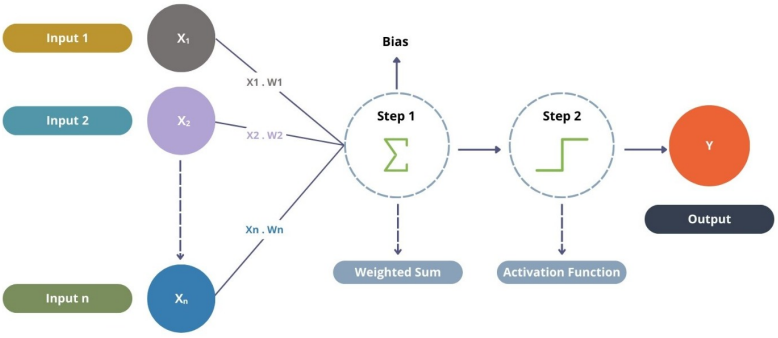

Figure 2.7: Neuron [45].

Convolutional Neural Network (CNN)

CNN are specialized Feedforward Neural Network (FNN) for image recognition and natural language processing [46].

The architecture comprises **convolutional** layers that apply kernels to inputs, capturing important features and reducing dimensionality. **Pooling** layers follow the convolutional layers and serve to further reduce the data's dimensionality by consolidating the extracted features, typically through methods like max pooling. **Fully connected** layers conclude the network, refining features for outputs like classification. CNN efficiently process spatial data, outperforming fully connected networks in complex pattern recognition.

An example of a CNN structure is depicted in Figure 2.8, illustrating the layer-by-layer process from feature extraction to classification output.

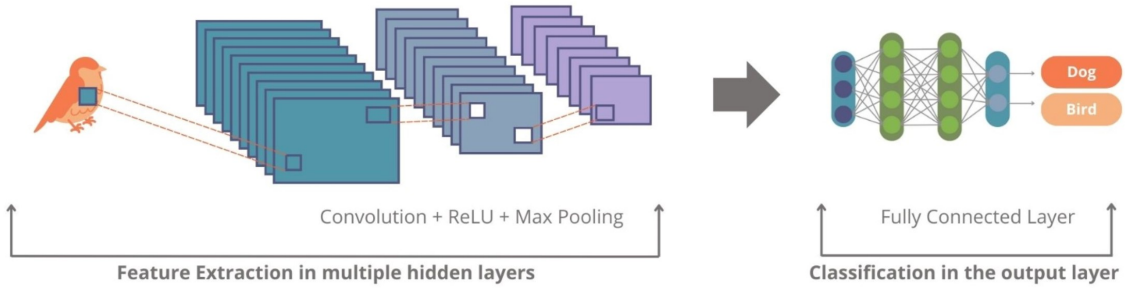

Figure 2.8: CNN Architecture for Image Classification [45].

Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) are a specialized class of Artificial Neural Networks (ANN) meticulously crafted to process data sequences. These networks operate on the principle of preserving the output of a given layer and then cyclically feeding it back to the input, allowing them to forecast the subsequent layer's output. By harnessing internal memory states and recurrent connections, RNNs excel in capturing temporal dependencies from sequential data. Consequently, they find their niche in solving a wide range of sequential problems, making them particularly suitable for tasks such as image captioning, video processing, speech recognition, and various natural language processing applications. In Figure 2.9, we can observe the typical architecture of an RNN.

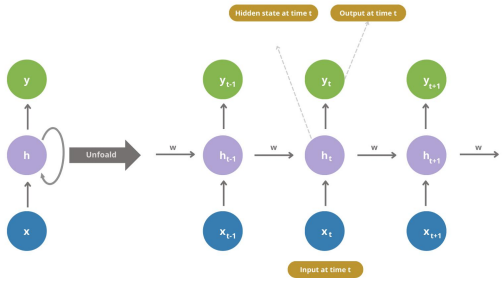

Figure 2.9: Recurrent Neural Network (RNN) architecture [45].

In an RNN, the architecture involves the input layer **x** processing data, with the middle layer **h** accommodating multiple hidden layers. What sets RNN apart is their ability to incorporate historical information: the output at time **t-1** influences **t** and **t** influences **t+1**. This recursive feedback loop allows RNN to process inputs of varying lengths while keeping the model size constant, making them proficient in handling sequential data.

Still, within the realm of RNN, it is worth highlighting the existence of a model known as Long Short-Term Memory (LSTM). LSTM are specialized for comprehending and processing sequential data. What sets LSTM apart is their unique ability to capture and remember long-term dependencies. This is achieved through a trio of essential gates: the **Forget gate**, **Input gate**, and **Output gate**.

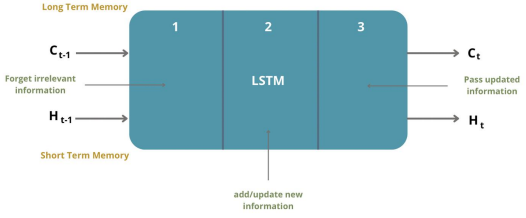

Figure 2.10: Long Short-Term Memory (LSTM) architecture [47].

Initially, the forget gate decides whether prior data is relevant or can be discarded. Subsequently, the Input gate selectively updates the cell state with new information from the current input. Finally, the Output gate facilitates the transfer of updated information to the next time step, completing one LSTM cycle.

These gates effectively manage information flow and prevent the vanishing gradient issue common in RNN. LSTM maintain a dual-memory system consisting of short-term memory (hidden state) and long-term memory (cell state), enabling them to excel in tasks involving sequential data and long-term dependencies.

Autoencoders (AE)

AE are an unsupervised machine learning technique and belong to the family of ANN. These neural networks are designed to take unlabelled data, compress it into a compact representation, and then use this code to generate an output that reconstructs the original data. This ability to compress and reconstruct the input data can be used to automatically extract useful features from the data.

An autoencoder typically consists of three layers: the **encoder**, **code** and the **decoder** as illustrated in Figure 2.11. The encoder component creates a code by compressing input data, and the decoder utilizes this code to reconstruct the input data, resulting in the output. Since the generated output retains essential input features while reducing their dimensionality, this method is often employed for both dimensionality reduction and feature learning [48].

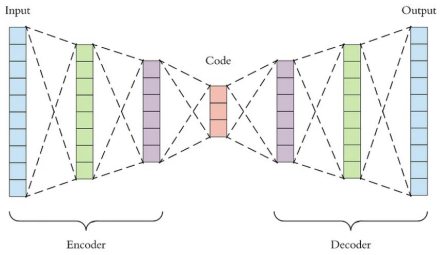

Figure 2.11: Autoencoders (AE) architecture [49].

The dimension of the **code** can be adjusted according to preferences. The number of nodes in the **code** layer, known as *code size* is a hyperparameter set before training the AE. While not obligatory, it is commonly observed that the decoder's architecture mirrors the encoder's. The only requirement is that the dimensionality of the input and output matches.

In both [48] and [50], the authors used AE to generate features automatically. In [50], the authors utilized a Denoising Autoencoder (DAE) to automatically extract features from auditory spectral data. The DAE was trained to reconstruct input signals from corrupted versions, and the reconstruction error was used to detect novel acoustic events, achieving significant performance improvements.

Generative Adversarial Network (GAN)

Generative Adversarial Network (GAN)s represent a groundbreaking approach in deep learning, akin to convolutional neural networks, but with unique capabilities in data generation. Generative Adversarial Network (GAN) comprise two interconnected sub-models: the generator model, which creates new examples, and the discriminator model, which classifies examples as real or fake. They are trained together in a zero-sum and adversarial game, until the discriminator is fooled about half the time, indicating that the generator is creating plausible examples [51]. The typical structure of a GAN can be seen in Figure 2.12.

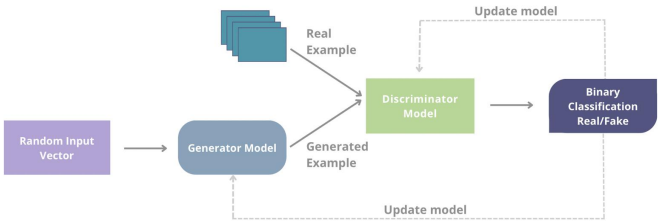

Figure 2.12: Generative Adversarial Network (GAN) architecture [45].

GANs operate through a feedback loop where the generator learns to produce more realistic data by trying to outsmart the discriminator. Initially, the generator produces data that is easily distinguishable from real data. As training progresses, the generator improves its ability to create convincing fake data. Simultaneously, the discriminator becomes better at identifying fakes. This dynamic training process leads to the generation of high-quality synthetic data. GANs are particularly effective in producing complex data like images, where they can capture and replicate intricate patterns and textures.

Expanding beyond image generation, GANs also find innovative applications in other fields. For example, in the context of RF fingerprinting, GANs can generate synthetic signals that mimic specific device characteristics. This enables more accurate device identification and improved detection of spoofing attempts, enhancing overall network security and reliability in wireless communication systems.

2.2.2 GNU Radio

GNU Radio is a popular open-source platform widely used to develop SDR systems. One of its key features is the built-in tool called GNU Radio Companion (GRC), which allows users to quickly create and modify signal processing flow graphs through a user-friendly GUI.

This GUI will enable users to quickly drag and drop signal blocks onto a flow graph and connect them. Additionally, GNU Radio has various pre-defined signal processing blocks that serve as building blocks for standard functions such as modulation, demodulation, filtering, and more. Furthermore, for users with specific requirements, GNU Radio supports the creation of custom signal processing blocks using Python or C++. This exceptional flexibility makes GNU Radio suitable for various wireless communication projects.

2.2.2.1 Zigbee Signals

Zigbee is a notable solution among the diverse wireless communication technologies available, particularly well-suited for applications requiring low power consumption, short-range communication, and high reliability.

ZigBee devices comply with the IEEE 802.15.4 standard and are built to support different modulation schemes and frequency bands. This makes them incredibly versatile and adaptable to various applications. Typical modulation schemes Zigbee utilizes include Binary Phase Shift Keying (BPSK) and Offset Quadrature Phase Shift Keying (OQPSK). These modulation techniques enable efficient data transmission with low power consumption, a critical feature for battery-operated devices.

Per the 802.15.4 protocol [52], the OQPSK PHY Protocol Data Unit (PPDU) is structured as depicted in Figure 2.13, comprising the following components: Synchronization Header (SHR), PHY Header (PHR), and PHY payload.

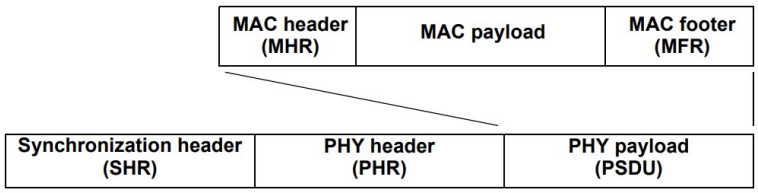

Figure 2.13: PPDU structure [52].

The SHR is the initial portion of the PPDU, designed to enable synchronization between the transmitter and receiver. It typically encompasses a preamble field and a Start Frame Delimiter (SFD) field. After the SHR, the PPDU frame includes the PHR, which consists of the frame length information. The PHR is followed by the PHY payload, which carries the PHY Service Data Unit (PSDU).

2.2.2.2 Signal Capture

The following section comprehensively examines advanced methodologies for capturing IEEE 802.15.4 signals with GNU Radio. This exploration will cover state-of-the-art techniques and delve into the intricacies of using GNU Radio to accurately acquire and decode these signals, which are critical for wireless communication within the Zigbee protocol.

IEEE 802.15.4 Transceiver

For signal capture, a transceiver block implementing the IEEE 802.15.4 OQPSK PHY, found within the gr-ieee802.15.4 module, can be employed. This component has proven effective in capturing and processing IEEE 802.15.4 packets [53].

In [54], the authors provide a detailed overview of the transceiver structure as presented in the GRC, a graphical user interface for setting up and configuring signal processing flow graphs. Through this exposition, the layered structure of the communication system becomes identifiable. As illustrated in Figure 2.14, this visual representation further clarifies the arrangement and functionality of the various components within the system.

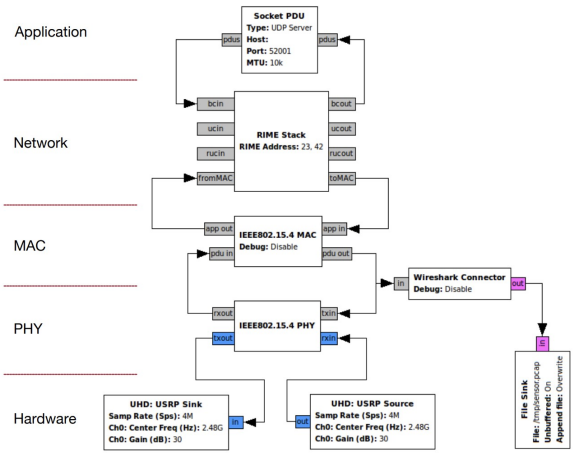

Figure 2.14: Transceiver Architecture [54].

Based on [55], a fully operational receiver encompasses components for synchronizing carrier frequency, timing of chips and symbols, and modules for detecting data. Specifically, this article presents a MSK-based receiver, which is composed of four distinct modules, depicted in 2.15. The first module is the **Quadrature Module**, which is responsible for estimating the instantaneous frequency of the signal. Next, the **Single Pole IIR Filter**, crucial for calculating the DC bias from the estimated frequency, follows. The **Clock Recovery MM Module** implements the Muller and Mueller timing recovery algorithm to synchronize the receiver's clock with the incoming signal's timing, ensuring precise data interpretation. Finally, the **Packet Sink**, a multifunctional module, manages synchronization, frame decoding, and data detection, accurately extracting and processing data from received signals.

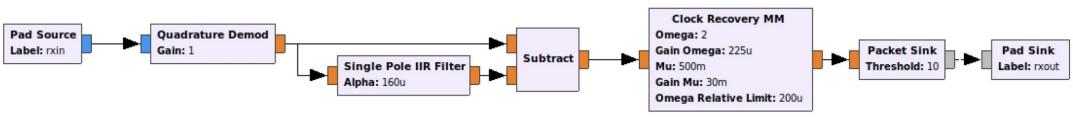

Figure 2.15: MSK-based receiver [55].

Furthermore, this paper proposes enhancements to the MSK-based receiver, as shown in Figure 2.16. These improvements include adding a Low Pass Filter (LPF) module designed to significantly reduce noise before phase differentiation of incoming samples, with a cutoff frequency set at 1.5 MHz, and a Decimating FIR Filter module for matched filtering on raw frequency estimates.

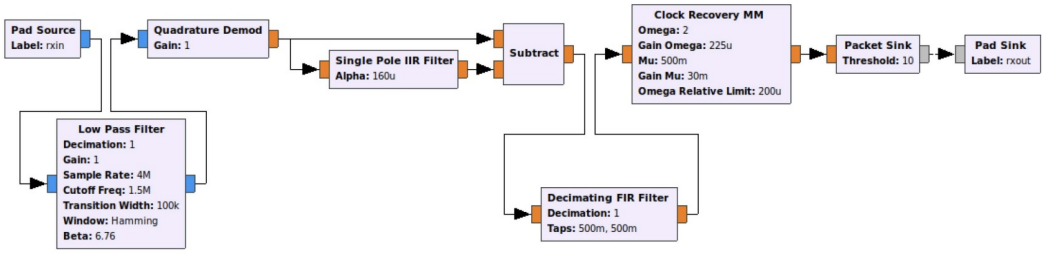

Figure 2.16: Suggested Improvements to the MSK-based Receiver [55].

2.2.2.3 Signal Processing

After examining the intricacies of signal capture in the previous section, we now focus on exploring essential modules and methodologies in signal processing. This segment will delve into the vital components and techniques for efficient and accurate signal processing.

Moving on to a more detailed analysis of the IEEE 802.15.4 transceiver's operation, it becomes clear that essential modifications are required. Currently, the transceiver outputs only an Async Message, which is inadequate for practical analysis of the incoming packet. Direct observation of the signal from the SDR is crucial for this purpose. Since the analysis demands access to the non-demodulated signal, the transceiver must undergo modifications to allow this. Ensuring that only IEEE 802.15.4 packets are provided to the feature extraction blocks is critical. Feature extraction should, therefore, only commence once it is confirmed that the samples received are from an IEEE 802.15.4 packet, enabling the effective utilization of the complex samples for advanced analysis.

To address these requirements, João Faria [56], drawing on insights from a previous study [34], propose targeted modifications to the transceiver's architecture, as depicted in Figure 2.17. These changes, primarily implemented in the packet sink block's source code, introduce a new preamble sink block. This innovative block is designed to accept both the original float stream and a delayed complex stream of the signal before demodulation, enabling further processing within the flowgraph. The newly modified scheme distinguishes itself in the output content of the two streams. The preamble output is designed to truncate the packet at the preamble, while the USRP pass-through provides the entire packet. In scenarios where the preamble sink (akin to the packet sink) detects an IEEE 802.15.4 packet, it releases the contents of the maintained buffer. Conversely, if no packet is detected, the output remains zero in both the real and imaginary parts of the signal. This modification ensures robust and accurate processing of IEEE 802.15.4 packets

by allowing feature extraction only when confirmed that the received samples are from an IEEE 802.15.4 packet. This approach prevents feature extraction on devices that might not be ZigBee devices, thus ensuring precision and relevance in the processing.

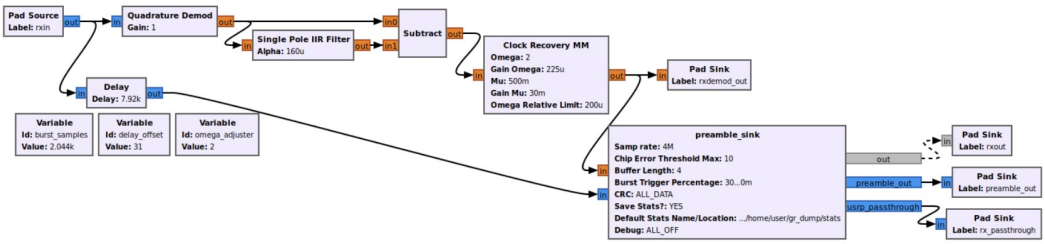

Figure 2.17: Modified transceiver [34].

Effective signal processing can be achieved by utilizing the transceiver discussed in the previous section and existing blocks within GNU Radio. Furthermore, the author of [56] introduces new blocks explicitly developed to enhance signal processing capabilities. These blocks include a **Preamble Cutter**, a **Frequency Analysis Sink**, and a **CFO Estimating** block.

2.2.2.4 Integration

This section explores multiple approaches showcasing how GNU Radio can be integrated with a Machine Learning environment. Each approach offers distinct methods for combining these technologies, enhancing signal analysis capabilities.

The approach we will explore involves using the GRC software, which stands out for its GUI, making signal processing flowgraph development more intuitive and accessible. With GRC, users can easily create a visual flowgraph, aiding in understanding how the signal is being processed. Furthermore, a notable feature of GRC is its ability to automatically convert a flowgraph into a Python file, which can later be compiled and executed from the terminal. This functionality is handy as it allows for efficient deployment of the flowgraph in operational environments. GRC is also compatible with Python development tools, offering additional flexibility in developing signal processing applications.

Another approach to integrating GNU Radio with ML environments involves exporting processed signal data to a file subsequently used for analysis. This method uses GRC to capture and process radio or digital signals. A *File Sink* block records the data into a file. The data is then read and processed in an ML environment, such as Python with TensorFlow or scikit-learn, using libraries like NumPy or Pandas. This technique facilitates the application of ML models for tasks such as classification, regression, or pattern detection on the loaded data. It is noteworthy that a similar technique focusing

on data exportation and subsequent reading for ML analysis is also employed in [56], although the specifics of the integration approach may vary.

A distinct approach to integrating GNU Radio with ML, demonstrated in [57], involves developing custom GNU Radio blocks that interface with the TensorFlow API. This method enables direct application of TensorFlow's ML models within the GNU Radio flowgraph, allowing for the seamless combination of signal processing and ML analysis. It leverages TensorFlow's advanced capabilities to enhance signal data processing within the GNU Radio environment.

As the final approach in this section, we move towards a more specific and complex method for **real-time classification**. This approach adopts a client-server communication model, utilizing ZeroMQ with TCP for data exchange [58]. In this configuration, GRC serves as the server, using a specialized block, such as ZMQ PUSH Sink, for transmitting data blocks. This block establishes a PUSH socket to distribute messages to PULL clients evenly. The client, tasked with receiving these data blocks from the GRC server, processes the incoming data to generate classification patterns based on previously acquired calibration data. Subsequently, the client calculates features from the received data and inputs them into a classifier tailored for real-time applications.

3

Development

In this chapter, we present a comprehensive overview of the system developed for real-time physical layer authentication in Zigbee networks. The system is structured around key components that integrate machine learning techniques with GNU Radio, a software-defined radio platform, to achieve robust and efficient authentication. This chapter is organized as follows: we begin by explaining the overall system architecture and its division into distinct modules. We then delve into the details of the GNU Radio module, including its origins, the modifications made, and its objectives. Next, we discuss the integration with machine learning, highlighting both the failed attempts and the successful approaches. Following this, we explore the ML module, focusing on its communication, the training script, and the process of database generation. Finally, we conclude with an overview of the GUI that ties the system together.

3.1 System Overview

The developed system is fundamentally divided into two main components. The first is the GNU Radio module, which captures and processes the Zigbee signals, preparing the raw data for subsequent analysis. The second component is the ML module, where advanced algorithms are applied to the processed signals to make authentication decisions in real-time. This division allows for a clear separation of responsibilities within the system, ensuring that each module can be optimized for its specific function.

A visual representation of the system's architecture is provided in Figure 3.1, which illustrates the flow of data between these two key modules. This diagram offers a clear view of how the system components interact, from the initial signal capture to the final authentication decision.

Figure 3.1: System Overview

The following sections will provide a detailed examination of each module, starting with the GNU Radio module, followed by the integration with the ML module, and concluding with an overview of the GUI.

3.2 GNU Radio Module

The developed system begins with the capture and initial processing of ZigBee signals, which is achieved through the coordinated deployment of several IRIS ZigBee devices. A total of twelve devices were used, all configured to transmit the same binary sequence every 63 ms, with identical MAC addresses, rendering them indistinguishable at the network level. Each device periodically transmitted a data frame with 30 bytes of payload, using 16-bit short addresses, from address 1 to the broadcast address 0xFFFF. These devices were programmed using TinyOS 2.1.2 and the AMSenderC component. Figure 3.2 shows the IRIS ZigBee sensors used in the system.

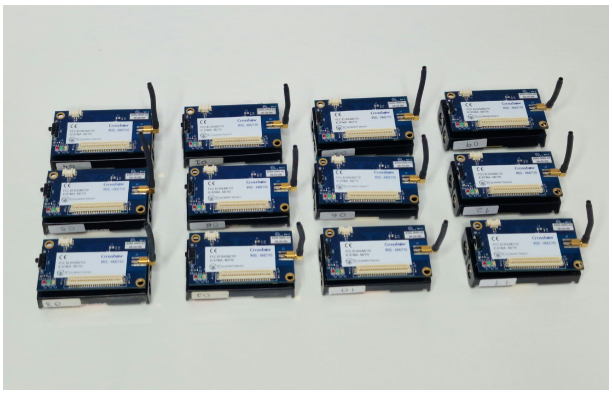

Figure 3.2: Iris Devices

These signals were captured using a Nuand bladeRF 2.0 micro xA4 SDR module, which offers a wideband transceiver range from 300 MHz to 3.8 GHz. The bladeRF, operating at a sampling rate of 4 MHz, was connected to a computer via a USB 3.2 Gen 1 port to ensure adequate bandwidth for processing the high-rate data streams. Figure 3.3 illustrates the Nuand bladeRF module utilized for signal capture.

Figure 3.3: Nuand bladeRF

To fully understand the role of the GNU Radio module in this system, it is necessary to explore how these captured signals are further processed and analyzed. We will now examine the implementation of the GNU Radio module, focusing on the specific modifications made to meet the system's requirements and the objectives it was designed to achieve.

The GNU Radio code used in this thesis is based on an original implementation sourced from [34]. This foundational code was further adapted using the modifications introduced by Inês Pereira in her thesis work [59].

One of the key adaptations made by Inês Pereira, and employed in this thesis, is the direct application of fingerprinting algorithms to the raw samples received by the NUAND SDR module [59]. This approach contrasts with other methodologies, such as Cruz's implementation [34] of the RF-DNA fingerprinting system for IEEE 802.15.4, where algorithms were applied to the output of the clock recovery module, resulting in a filtered signal with a reduced sampling rate of 2 MHz. While this filtering effectively reduces noise, it also leads to the loss of some transient characteristics of the signal, which are crucial for precise RF fingerprinting.

By retaining more of the signal's transient characteristics through this direct processing approach, the modifications made to the original GNU Radio code enhance the system's ability to perform accurate and reliable authentication in real-time. Figure 3.4 shows the GNU Radio receiver module that served as the foundation for the work in this thesis, including the structure and key elements used as a starting point.

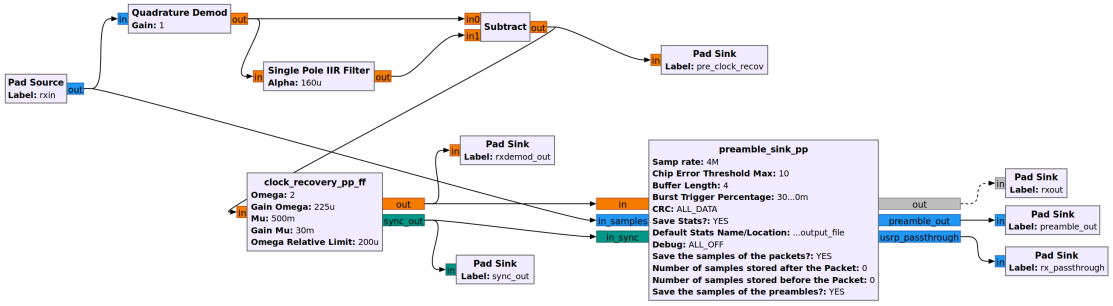

Figure 3.4: Gnu Radio Receiver Module

One of the first steps in developing and testing the system involved using a pre-recorded file of samples to validate the basic functionality of the flowgraph. This file, named *samples.cap*, was recorded in an environment that included various sources of signals, resulting in approximately 7 GB of data. The samples contained signals from multiple Zigbee sensors, along with other types of signals such as Wi-Fi, providing a realistic test scenario for the system. Utilizing this file allowed for a stable and repeatable testing process, which was crucial for identifying and resolving potential issues in the system before moving on to real-time data processing.

One of the most significant aspects of this development involved making specific modifications to the *preamble_sink_pp* block, which led to the creation of a new module named *PHY_PREAMBLE_LEANDRO*. This new module is essentially an altered version of the *IEEE802.15.4 OQPSK PHY PREAMBLE GRT2*, adapted to better suit the needs of the system. Figure 3.5 illustrates the *PHY_PREAMBLE_LEANDRO* module, showing how it integrates into the overall flowgraph.

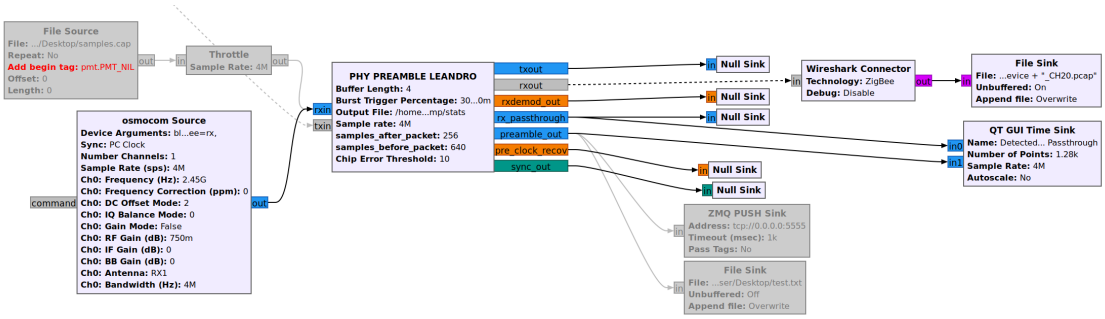

Figure 3.5: Gnu Radio Script

In this thesis, the Machine Learning algorithms are applied directly to the preamble of the Zigbee signals. To achieve this, one of the primary objectives was to modify the *preamble_sink_pp* block within the GNU Radio module so that it outputs only groups of 640 samples, corresponding to the exact length of the Zigbee preamble.

This approach was chosen to ensure that the data entering the Machine Learning module was already segmented and aligned with the preamble, making it easier to organize and process the information consistently. By structuring the system in this way, the *preamble_sink_pp* block filters out unnecessary data and focuses solely on the preamble. This modular approach not only helps in maintaining a clear separation of tasks but also ensures that each module receives and processes data in the format it was designed for. As a result, when the data reaches the Machine Learning module, it is pre-processed and ready for immediate analysis, aligning with the specific goals of this thesis.

First, the system needs to detect when the Zigbee signal begins its preamble. To do this, the `preamble_sink_pp` block looks for a specific synchronization sequence within the signal. Typically, this sequence consists of four bytes of $0x00$ followed by one byte of $0xA7$, as defined by the Zigbee standard IEEE 802.15.4. This helps the receiver identify the start of the Zigbee frame.

The `preamble_sink_pp` block utilizes a shift register to continuously accumulate incoming bits as the signal is received. This shift register acts as a sliding window that compares the incoming bitstream with the expected sync vector. As new bits arrive, they are shifted into the register while older bits are discarded. To determine whether the sync vector is present, the block uses a process called chip mapping. The shift register's contents are compared against predefined sequences stored in a $CHIP\_MAPPING$ array, which represents the possible chip sequences in Zigbee signals. Each possible byte, including the sync vector's components, is mapped to a specific sequence of 32 chips. The comparison is made using an XOR operation, which identifies how many bits in the shift register differ from the expected sequence. If the number of differing bits, or errors, is below a certain threshold, the block concludes that the sync vector has been detected. This detection triggers the system to begin capturing the relevant data, starting with the preamble and continuing through the subsequent data frame. At this point, the preamble should be fully captured in the buffer.

At this stage the system is supposed to have detected that we have a preamble and so we should have in the `d_preamble_samples_buffer` all the information we need to inject into the output for the machine learning module, but this is not the case. Through some tests, it becomes clear that sometimes the buffer fails to capture all the signal samples with the initial zeros, making it necessary to restructure the `d_preamble_samples_buffer`.

To address this, we created a function called `adjust_preamble_size`, whose purpose is to restructure the preamble buffer by adding the signal samples associated to the missing zeros, ensuring that exactly 641 samples are consistently injected into the output as detailed in Algorithm 1.

In addition to the `d_preamble_samples_buffer`, we also maintain a `d_packet_samples_buffer`, which stores a configurable number of samples both before and after the packet. This design allows the system to recover any lost initial samples by using the data stored "before" the packet in the `d_packet_samples_buffer` to fill in the missing parts, ensuring the integrity of the preamble in the output stream.

Algorithm 1 Adjust Preamble Size

```
current_size ← size(d_preamble_samples_buffer) − 1
if current_size < PREAMBLE_LEN then
    missing ← PREAMBLE_LEN − current_size
    offset ← size(d_packet_samples_buffer) − size(d_preamble_samples_buffer) − missing + 1
    if offset ≥ 0 then
        Copy missing samples from d_packet_samples_buffer[offset] to d_preamble_samples_buffer
    else
        Error: insufficient samples
    end if
else if current_size > PREAMBLE_LEN then
    Reduce d_preamble_samples_buffer to PREAMBLE_LEN + 1
end if
```

The algorithm must fill in any gaps in the `d_preamble_samples_buffer` with data from the complete `d_packet_samples_buffer`. To do this, it calculates the number of missing samples and copies that amount from the packet buffer, making sure the preamble buffer ends up with exactly 640 samples.

Another important feature of this `preamble_sink_pp` code is the addition of a separator at the start of each preamble section to serve as a reference point, symbolized by `gr_complex(10000.f, 0)`. Consequently, when the system outputs the preamble, it actually consists of 641 samples, rather than the expected 640. While this separator may not be critical in contexts with isolated messages, such as those discussed in this thesis, it becomes crucial in continuous data streaming environments where it effectively distinguishes between consecutive preambles.

The operation of the `preamble_sink_pp_impl` code is based on a state machine that manages the detection and processing of packets. The purpose of the state machine is to guide the flow of data from the detection of a preamble to the complete formation of the packet. This process takes place in several phases, represented by the states that the system goes through:

- **STATE_SYNC_SEARCH**: It is the initial state where the system is searching for the synchronization vector in the incoming data. During each iteration, the system checks the input bits for the identification of the preamble pattern. The shift register `d_shift_reg` flows the incoming data bits and then checks whether they match or not with the expected pattern.

- **STATE_HAVE_SYNC**: Once the preamble has been detected, it goes to this state. Samples are stored in two buffers, one for the preamble, `d_preamble_samples_buffer`, and one for the packets, `d_packet_samples_buffer`. This state also invokes the `adjust_preamble_size` function, which ensures that the preamble buffer is adjusted to contain 641 samples, something necessary for correct processing precisely.

- **STATE_HAVE_HEADER**: After synchronization and identification of the preamble, the system transits to this state. Now, the packet header is processed, and the system starts building the final packet, byte by byte. After decoding the packet header and

payload, the system returns to the synchronization search state, ready to process the next packet.

The flow between these states is illustrated in Figure 3.6.

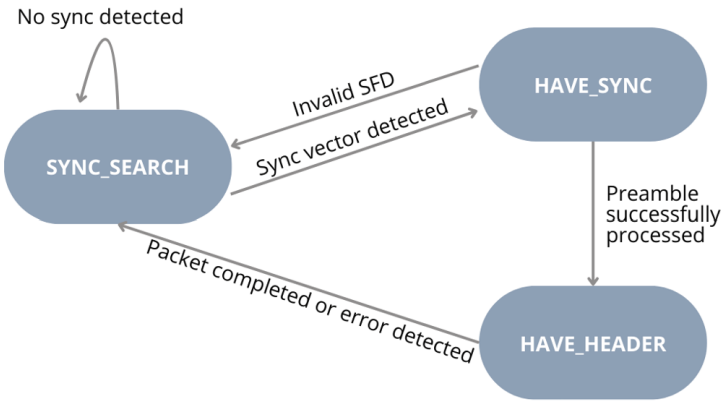

Figure 3.6: State machine diagram

3.3 GNU Radio and ML Module Integration

Now that we have ensured the preamble buffer has exactly 641 samples, it's time to focus on integrating it with the ML module. To do this, the system was tested using a technology called ZeroMQ [58].

As discussed in the state-of-the-art section of this thesis, the idea was for GNU Radio to function as a server using the ZMQ push sink block, while the ML module would operate as a Python client capable of receiving the data stream. Initially, the system was set up as shown in Figure 3.7, where the zmq push sink module was directly connected to the `preamble_out output`, where the 641 samples were injected whenever a preamble was detected.

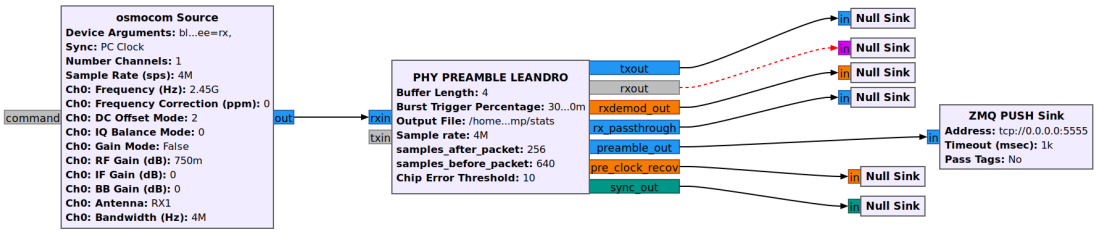

Figure 3.7: ZMQ Push Sink

In this setup, since GNU Radio was running on a VMware virtual machine, we chose the address tcp://0.0.0.0 and the port 5555 for simplicity, making it easier to accept connections. The Machine Learning module was configured to receive the data stream using standard ZeroMQ methods, as outlined in the official ZeroMQ documentation [60]. The code we used to set up the connection was similar to the following algorithm 2:

Algorithm 2 Connecting to GNU Radio via ZeroMQ

```
protocol ← "tcp://"
vm_ip ← "192.168.2.119"
port ← "5555"
address ← protocol + vm_ip + ":" + port
context ← zmq.Context()
pull_socket ← context.socket(zmq.PULL)
pull_socket.connect(address)
timeout_in_milliseconds ← 500
pull_socket.setsockopt(zmq.RCVTIMEO, timeout_in_milliseconds)
print("Trying to connect to GNU Radio at {address}...")
while True do
    data ← pull_socket.recv()
    complex_data ← np.frombuffer(data, dtype=np.complex64)
end while
```

However, after exhaustive testing of the system, we concluded that the zmq push sink block didn't have the necessary throughput to communicate with the Machine Learning module, which caused the system to go down several times.

As a result, we had to switch to a lighter solution that sends isolated messages via TCP sockets. On the GNU Radio side, once the preamble buffer has been processed through the `adjust_preamble_size` function and confirmed to have 641 samples, a function called `save_preamble_samples` is invoked. This function first saves the preamble samples to a binary file and then checks whether the TCP socket is active. If the socket is active, it also sends the 641 samples through it; if not, the samples are simply stored in the file.

To make this possible, a new class name `socket_preamble_impl.cc` was created and added to the `gr-ieee802_15_4_Nova` project. This class is responsible for creating, opening and closing the socket and ensures that the data is sent correctly when the connection is active.

To access data from the machine learning module, we used a straightforward approach for managing TCP sockets in Python. We set up a connection where the machine learning module acted as the server, ready to receive data from GNU Radio. In Python, this typically involves using the socket library to create a connection, link it to a specific address and port, and then wait for incoming connections. Once the connection was established, the data came through in chunks of 641 samples.

3.4 Machine Learning Module

To develop this module, we used Anaconda to manage the Python environment and dependencies, ensuring a smooth and compatible setup. Coding was primarily done in Visual Studio Code, with the machine learning models implemented in Python using the TensorFlow and Keras libraries.

The goal of this thesis wasn't to build neural networks and classifiers from scratch but to identify which ones perform best and how different training methods impact the system's performance. For testing, we primarily used two types of AE NN: one was a Conv1D network, and the other was a Gated Recurrent Unit (GRU) network. These were chosen according to their performance in performing automatic feature generation for Radio Frequency Fingerprinting systems. In [59], the author demonstrated that an optimized Conv1D network achieved an F1-score of 93% with a latent state of size 8, when combined with an MLP classifier. Additionally, when using the same Conv1D network with a latent state of size 12 and an RF classifier, the system also achieved an F1-score of 93%. These results serve as important benchmarks for comparison with our system.

In this thesis, we use the same Conv1D autoencoder defined in [59], with a latent state of size 12 and a RF classifier. The choice of the RF was motivated by its lower computational overhead and fast training time, which are particularly important for real-time systems where efficiency is crucial.

When the machine learning module receives the preambles in chunks of 641 samples, it first ensures that the full message has been received. Since TCP is a stream-oriented protocol, a single send on the server side might not correspond to a single recv on the client side. The data can arrive in multiple chunks, so the module carefully assembles the complete message before proceeding. Once the message is confirmed, the module checks the format, ensuring the first position contains the value 10000, which serves as the separator. Once we have done that, all we have to do is ignore the first position and we are guaranteed to have the 640 samples of the preamble.

At this stage, the system can take two paths, necessitating the implementation of two distinct modes: training mode and classification mode. The developed system should

operate in both modes for it to execute at maximum performance and flexibility. The former will record the captured data and use it for model modification to enable the classifier to generalize properly for new samples, while the latter will use the trained model to classify incoming data in real-time. These two modes are fundamental, as they allow the system to adapt to various conditions and ensure its robustness. These modes can be easily selected and alternated through an app specifically developed for this system. This app allows users to control the system's operations efficiently, and it will be described in further detail in a future section.

3.4.1 Training Mode

When it comes to the training mode the first step is to save the samples in binary format so that we can create our own database. To achieve optimal system performance, it is crucial to ensure that the training environment matches the real-time test conditions. This is why training the autoencoder and the classifier in a scenario identical to the one being replicated is so important.

DataBase Generation

Before discussing the database generation itself, it is important to understand that various training scenarios were considered for the system. These scenarios include different distances between the sensor and the antenna, as well as different movement types during training. Consequently, multiple databases were created, each corresponding to a specific training scenario. Each database is used to train an RF classifier, and the naming structure for the binary files is shown in in Figure 3.8:

- **Distances:** 15 cm, 40 cm, 100 cm
- **Test Types:** Static, Dynamic (with movement)

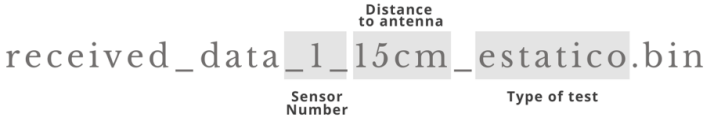

Figure 3.8: Structure of the name of each dataset

Regarding the creation of the database, we initially started by recording several binary files generated from various combinations of sensors, distances and types of training. Data was collected by keeping the sensor on at each distance for 60 seconds. With the various files already created and properly named, as described above, the database generation script searches the output folder for all the files that correspond to the same scenario.

For example, as described below in Figure 3.9, if there are four sensor files positioned at 15 cm in static mode, the script will process these files to generate two output files.

The first is the final dataset file, which contains the data from these four sensors. The second is a settings file that includes information such as the bin files used for generating the dataset, the preamble size, the directory of the database file, and the directory of the settings file.

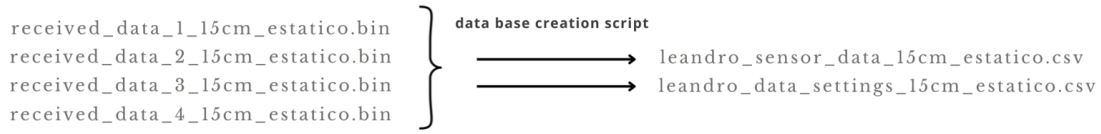

Figure 3.9: DataBase Generation Script

As a result, we obtain a data set with the following Table: **sensor**, **sensor_id**, **position**, **preambles_real**, **preambles_img**, as shown in Table 3.1.

Sensor	Sensor_id	Position	preambles_Real	preamble_img
iris_1	1	15cm	0.02050 0.06201 ...	0.02123 0.00456 ...
iris_2	2	15cm	0.03852 0.08931 ...	0.04538 0.06708 ...
iris_3	3	15cm	0.02637 0.01292 ...	0.04788 0.04146 ...
iris_4	4	15cm	0.08963 0.00372 ...	0.71189 0.08890 ...

Table 3.1: Example of the structure of the generated dataset.

Feature extraction and classifier training

Now that we have the two csv files containing the dataset of the respective context, the main objective of this script is to train the RF classifier based on the features generated by the AE. It all starts with loading the previously generated datasets to which an One Hot Encoder (OHE) will be applied for categorical variables. This technique transforms each categorical class into a separate binary column, effectively splitting the classes into categories for better processing by the model. After adding this column to our dataset, the script loads a previously trained AE NN model, in this thesis the model most used for testing and improving the system was a Conv1D AE.

The Conv1D used in the development of this system, as described in [59], consists of the encoder network, and is trained as part of an autoencoder. The autoencoder includes an encoder and a decoder. The encoder consists of three Conv1D layers with ReLU activation functions, associated with pooling layers, followed by a dense layer before the linear output layer. A latent state of 12 was used, which means that the input data is compressed into a representation of 12 essential characteristics (features). These 12 features represent the most relevant information from the original data. The decoder is used during training and includes a dense layer and four Conv1DTranspose layers, which reconstruct the real and imaginary parts of the 640 complex numbers from the compressed representation.

The system is trained to minimize the error between input and output. The structure of a Conv1D Autoencoder is illustrated in Figure 3.10, taken from Inês Pereira's article [59].

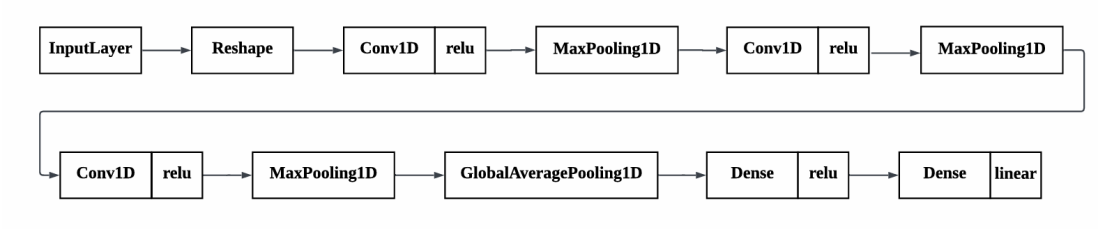

Figure 3.10: Convolutional 1D taken from [59]

The encoder of the Conv1D Autoencoder consists of three Conv1D layers with ReLU activation functions, followed by a pooling layer and a dense layer. The decoder, used during training, includes a dense layer followed by four Conv1DTranspose layers, which reconstruct the input data from the compressed representation

Once the dataset has passed through the encoder, it will return 12 self-extracted features that will be added to a training dataframe. Before using this dataset to train the classifier, the columns **sensor**, **sensor id** and the columns from **OHE.props**, which correspond to the categorization of each sensor, are added. This results in a total of 18 columns, 12 of which are autoencoder features, 2 informative columns and 4 columns from OHE.props, one for each of the 4 sensors in this example, as illustrated in Figure fig:24. It is worth noting that this is just an example of the system flow.

p0	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	sensor	sensor_id	is_iris_1	is_iris_2	is_iris_3	is_iris_4
-0,11328	1,28699	-1,19216	-0,48959	1,36827	-0,28283	1,23867	-0,29672	0,23953	-0,93225	-0,50128	-0,18439	iris_1	1	1	0	0	0
1,55648	0,69823	-1,21189	-0,55152	-1,44809	0,35157	0,46271	-0,96593	-0,91059	-0,52233	-0,52288	1,68259	iris_1	1	1	0	0	0
1,57786	0,60878	-1,31187	-0,42104	-1,81145	0,42767	0,37728	-0,9086	-0,8809	-0,52633	-0,42216	1,7509	iris_1	1	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0,57212	-0,47826	0,01426	2,2696	1,49879	0,42332	-0,27358	0,95196	0,95196	-0,41695	-1,17722	-0,42791	iris_4	4	0	0	0	1
2,04604	-0,73833	-0,22065	0,60414	-0,16836	0,82524	-0,80246	-0,51413	0,0821	0,07064	-0,21688	0,59363	iris_4	4	0	0	0	1
1,09322	-0,81966	0,07355	0,29368	1,65482	0,46259	-0,25689	-0,07642	0,88476	-0,18042	-1,37754	-0,27538	iris_4	4	0	0	0	1

Figure 3.11: In train Dataframe

The classifier used to develop this system was the RF Multiclass. This RF was built using the **Sensor_RFmultiClassifier** class, with specific parameters such as **n_estimators** ranging from 50 to 1000, **max_leaf_nodes** from 16 to 64, and **max_depth** from 1 to 50, as defined in the code. These parameters were optimized using a random search to ensure the best possible performance.

In [59], this RF classifier was used in combination with a Conv1D autoencoder with a latent state of size 12, achieving an F1-score of 93%. This result underscores the efficiency of this combination for Radio Frequency Fingerprinting systems and serves as an important point of reference for our system's performance.

Once the training dataframe was ready, the process continued with training the RF classifier. To do this, the previously prepared dataset, containing the extracted features

and sensor information, is used to train the model to identify patterns and make accurate classifications.

After training, the RF classifier is saved in a pickle file, allowing the trained model to be reused later, without the need for re-training.

In addition, the training time of this classifier, as described in [59], is significantly shorter than that of other types of classifiers, which is extremely advantageous in our case, since we are developing a real-time system. This issue of training time and its importance will be discussed in greater depth in future sections in the next chapter.

Thus, at the end of the training mode, the classifier generated is directly associated with the parameters of the training environment, such as distance and type of movement, so that the system has several classifiers depending on the environment in which they were trained. The classifier name is generated automatically based on the variables supplied by the user, following the structure `leandro_rf_classifier_{distancia}_{tipodetreino}.pickle`, which allows the system to easily identify the classifier because it is associated with its training environment.

3.4.2 Classification Mode

Within the machine learning module, there are two main operational modes, as explained above. While the training mode focuses on training a classifier using a previously trained feature extractor, the classification mode allows the system to perform real-time predictions based on pre-trained classifiers. In this mode, the user selects the database that was used to train the classifier, ensuring that the system can correctly identify the sensors during operation. This mode is designed to provide fast and efficient responses in real-time scenarios.

Once we have the 640 samples prepared, the system invokes the `process_samples` function. This function isolates and facilitates classification, hence organizing the flow into well-defined steps. The steps of this function are detailed in Algorithm 3 below.

Algorithm 3 Classification Flow in process_samples Function

1: $samples\_normalized \leftarrow$ `convert_normalize_complexVec2float2D2Vec`$(samples)$
2: $encoder\_output, \_ \leftarrow$ `get_encoder_vector`$(encoder\_nn, samples\_normalized)$
3: $prediction \leftarrow$ `rf_classifier.predict`$(encoder\_output)$
4: **return** $prediction[0]$

First, it uses the function `convert_normalize_complexVec2float2D2Vec` to normalize the complex samples, converting them into a format suitable for processing and ensuring proper scaling.

The normalized samples are then sent to the neural encoder via the `get_encoder_vector` function. This encoder transforms the samples into a latent vector, reducing their dimensionality. In our case, the resulting latent vector has 12 positions, which encapsulate the most relevant features for classification.

As far as the classifier used in order to make the prediction, it has to be understood that the choice depends on the information given by the user through the application. The core objective of this system is to ensure the best possible classification in real time. In other words, given a user's decision to test sensors in some particular scenarios, such as proximity to the antenna and no movement, the user selects the training database that was used to train the classifier, ensuring that the classifier is aligned with similar conditions.

For example, if the chosen scenario is 100 cm away and static, the system will search the classifier folder for the model trained for these specific conditions, such as the classifier `leandro_rf_classifier_100cm_statico.pickle`.

Once this is done, and with the prediction given by the classifier, it will be redirected to the front end of the application using a callback mechanism implemented in python. The graphics for each mode and how the user can interact with each mode will be explained in the next section.

3.5 User Interface

Before diving into the specifics of the user interface, this subsection will provide an overview of the challenges faced during the system's development, particularly around managing synchronization between the app and the underlying processes. One of the main challenges was keeping the app responsive while managing both real-time classification and training modes. This section will offer a user-centric perspective, highlighting the solutions implemented to address these challenges before moving on to the design choices that made the interface effective and easy to use.

The GUI plays a fundamental role in the development of any technological application, especially when it comes to tools that need to be accessible to a wide range of users.The main aim of the GUI is to make interaction with the system intuitive, efficient and accessible, improving and simplifying the user experience.

In the context of this application, the GUI has been designed to be extensible and integrable, allowing users to adapt and expand the tool according to their specific needs.The extensibility of the GUI means that new modules and functionalities can be easily added to the interface without the need for complex reconfigurations.This is particularly important in dynamic environments, where new sensors or devices can be integrated into the system on an ongoing basis. For example, if a new type of sensor is introduced, it can be quickly incorporated into the application via the GUI, with no complications for the end user.

In addition, the GUI's integration capability ensures that it can be easily connected to other applications or systems, creating a more robust and interconnected ecosystem.This feature makes the application more versatile, allowing it to be used in different scenarios and for different purposes, without the need to develop specific interfaces for each new use case.

By adopting a modular approach to building the GUI, we have managed to create a platform that not only meets current needs, but is also prepared for future expansion and

adaptation.

3.5.1 Technologies and Implementation Challenges

Essentially, the app is divided into two main modes of operation. The first is the training mode, where the user can interact in a simple and intuitive way, indicating the number of the sensor they want to add, the distance and the type of training. The application guides the user step by step until the sensor is correctly inserted into the system. The second mode is real-time classification, where the user selects the training database used to train the classifier. From there, the system then provides a real-time response by displaying the prediction based on the selected classifier. Both modes will be explained in more detail in the following sections.

To develop the application, the Python programming language was used, which offers a wide range of libraries and tools that facilitated the creation of a strong and functional graphical interface. The interface was built using Tkinter, one of the most popular libraries for developing GUIs in Python, known for its simplicity and efficiency in creating interactive interfaces.

During the development of the application, one of the biggest challenges encountered was managing the synchronization between the different processes taking place simultaneously. Let's think about real-time classification mode, for example. The system needs to be able to react to the user's interactions with the GUI, manage the data it receives, process it and generate a prediction while at the same time displaying it on the interface. It must do all of this while maintaining the interface responsive to user inputs, ensuring a smooth experience. To solve this problem, threads were used to ensure that the application could perform several tasks simultaneously without compromising the responsiveness of the interface. This is particularly important in real-time sorting mode, where continuous data processing must take place without causing delays in the user interface.

The first important thread is the **`servidor_thread`**, which runs the `start_server` function. This function is responsible for establishing communication between the system and the GNU Radio environment, receiving data samples in real time from the connected sensors. In training mode, the thread stores these samples to build the database used to train the classifier. In classification mode, the server thread continues to receive data from the sensors, but in this case it is sent directly to the previously trained classifier to generate real-time predictions.

The **`classificar_thread`** thread's job is to process the incoming data and generate predictions in real time. It interacts directly with `servidor_thread` via callbacks, since `servidor_thread` sends the data as soon as it receives it from the sensors. This mechanism allows `classificar_thread` to start the classification process immediately, as soon as the new data is available. Meanwhile, the `servidor_thread` continues to take samples from the sensors, ensuring that the two threads operate in parallel and efficiently. This process ensures that the system maintains its responsiveness, allowing the interface to function

fluidly while classification is carried out in real time.

Initially, the training timer was implemented directly in the **`main_thread`** created by tkinter, which after a few tests was found to cause a problem with the interface freezing and becoming unresponsive. So one of the ways to deal with this problem was to create a new thread responsible for dealing with this timer. This thread called **`cronometro_thread`** maintains control independently and sends updates to the interface without the application freezing, significantly improving the user experience.

The **`treinamento_thread`** is responsible for the process of training the classifier based on the samples received. Like other threads in the system, it interacts with `servidor_thread` to ensure that the necessary data is received correctly and stored for training. During training mode, `server_thread` receives the data and stores it in a binary file. The **`treinamento_thread`** is responsible for managing the entire training process afterwards, from generating the database, passing the data through the encoder module of the autoencoder, training the classifier and storing it in pickle format.

Figure 3.12 illustrates the interaction between the various threads created in the system, showing how they work together to ensure the application remains responsive while performing tasks such as real-time classification, sensor data handling, and user interface updates.

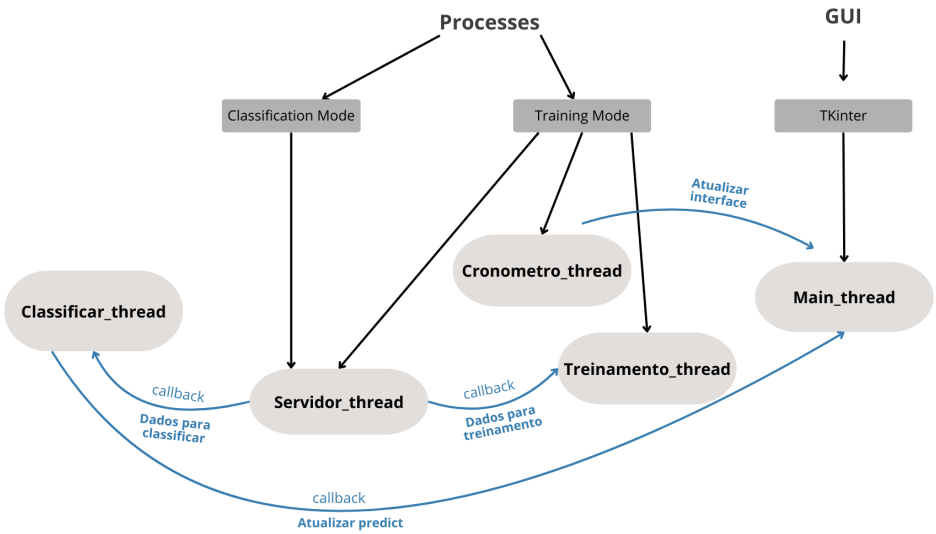

Figure 3.12: Overview of the threading architecture within the application

Tkinter runs its main thread to handle the system's graphical interface and manage events like button clicks and data input. This thread ensures the interface responds quickly to user actions. While not manually created like the other threads, it's crucial in keeping the GUI active and responsive. Even when tasks like training or classification are running in the background, this main thread keeps the interface working smoothly, processing user interactions without freezing or slowing down. This setup helps the application

remain responsive, even during resource-heavy processes.

Callbacks were also implemented to manage user interaction with the interface, allowing actions performed, such as choosing a sensor or setting test parameters, to be processed asynchronously and efficiently.

In Figure 3.13, we can see the application's main menu, where these technological elements are integrated to provide a fluid and intuitive user experience.

Figure 3.13: GUI menu

3.5.2 Training Mode

The training mode allows the user to easily configure and integrate a new sensor into the system. When selecting this mode, the user will need to provide some information, such as the sensor number, the desired distance for training and the type of movement, whether static or dynamic. Figure 3.14 clearly illustrates this task.

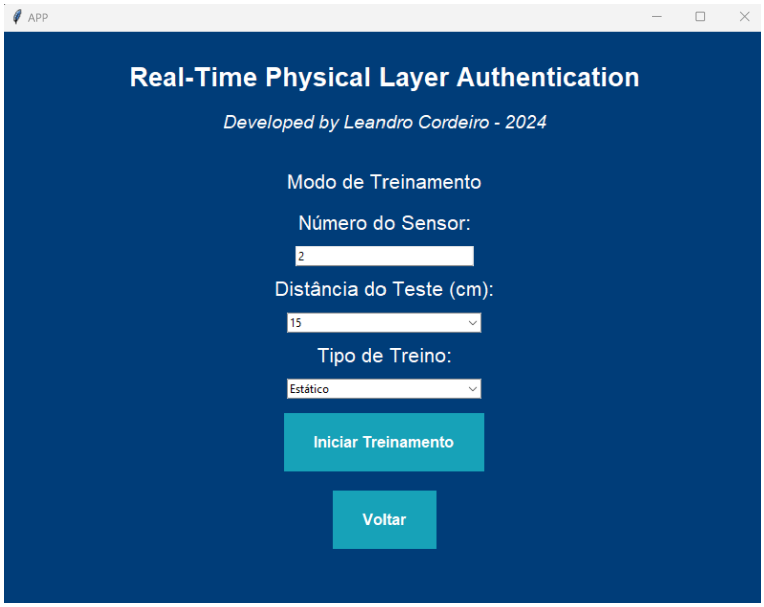

Figure 3.14: Scenario details

Once this has been done, the system will ask the user to wait a few seconds before informing them that they can turn on their sensor. The sensor should remain on for 60 seconds, during which time the system will be saving the sensor's samples. During these 60 seconds, a new file **`received_data_nrsensor_{distancia}_{tipotreino}.bin`** is created and populated with the samples the system receives. As soon as the timer has finished, as shown in Figure 3.15, the user is informed that they can turn off the sensor and should wait briefly. The database generation script will then be run again, using the new bin file to create a new training dataset **`leandro_sensor_data_{distancia}_{tipotreino}.csv`** and a new settings file **`leandro_data_settings_{distancia}_{tipotreino}.csv`**.

Figure 3.15: Training Mode Interface

This training mode focuses exclusively on training the classifier using a feature encoder that has already been pre-trained. In other words, the system does not completely re-train the neural network of the autoencoder but uses the existing encoder to extract the relevant features from the newly collected data. The classifier can be trained to generalize the predictions with the new data set by applying the mentioned features. It will ensure that the system keeps enhancing its capacity for classification and does not need to redesign or retrain the feature extraction model, hence increasing the speed of adaptation for new scenarios.

At this point, the training script described in the previous section will be executed again, this time with the new training dataframe and settings file, resulting in a new classification model that will be saved in pickle format.

In training mode, information provided by the user, such as distance and type of training (static or dynamic), is used to name the trained classifier. This ensures that the test parameters are reflected in the classifier's name, making it easy to identify when it needs to be used in classification. The system saves the classifier in pickle format with a name that follows this structure: **`leandro_rf_classifier_{distancia}_{tipodetreino}.pickle`**. For example, if the user chooses a distance of 100 cm and a static test, the classifier will be saved as `leandro_rf_classifier_100cm_estatico.pickle`.

In summary, the process is automated and guided, allowing users without specialized knowledge to add new sensors simply and quickly. From data collection to the creation of the new model, all steps are carried out consistently and efficiently. As a result, the system remains flexible and scalable, ready to integrate new devices without compromising classification accuracy. In just a few steps, the user has an up-to-date system, adapted to their new needs, without any additional effort.

3.5.3 Real-Time Classification Mode

With regard to real-time classification mode, this, as the name suggests, is the mode where the user can test whether the system can correctly identify the desired sensor. Initially, when selecting this mode, as with the training mode, the user must choose the classifier they want to use, which means selecting the database that was used to train it, based on the distance and type of training performed. The interface for entering this data is illustrated in Figure 3.16.

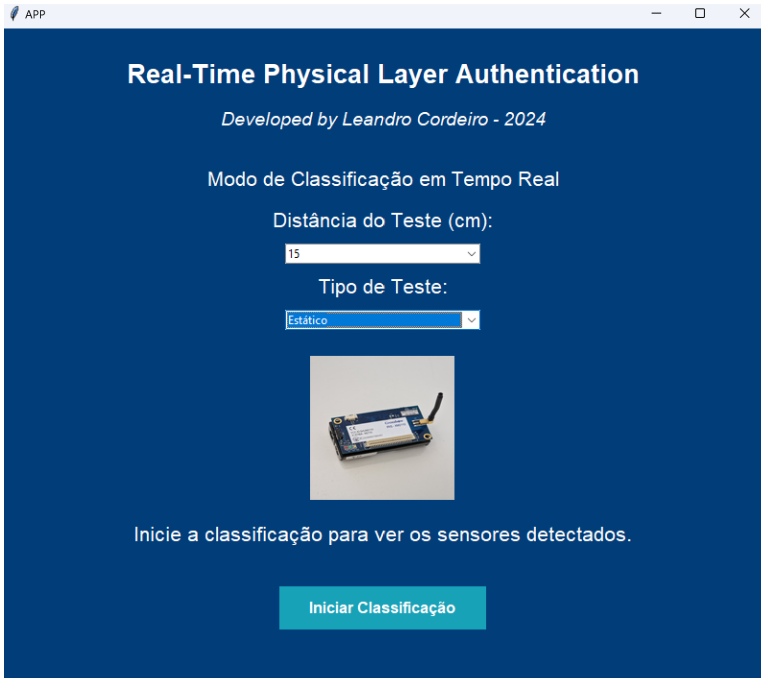

Figure 3.16: Classification Interface

This step is necessary because, in the classification mode, the system selects the pre-trained classifier. As will be shown in the following chapter, the system's performance is improved when the classifier is trained in the same scenario where the test is conducted. Different sensors may behave differently depending on their environment or setup, so aligning the classification with similar training conditions is crucial for achieving the best results.

With the data provided, the system will load the networks and the appropriate classifier, informing the user that they can start the classification. In real time, a label with the most recent active sensor will appear on the interface, as shown in Figure 3.17. For example, if sensor 2 is activated, the label will display "sensor 2"; if it is switched off, it will continue to display "sensor 2" until another sensor is switched on, at which point the system will update the display to the new sensor.

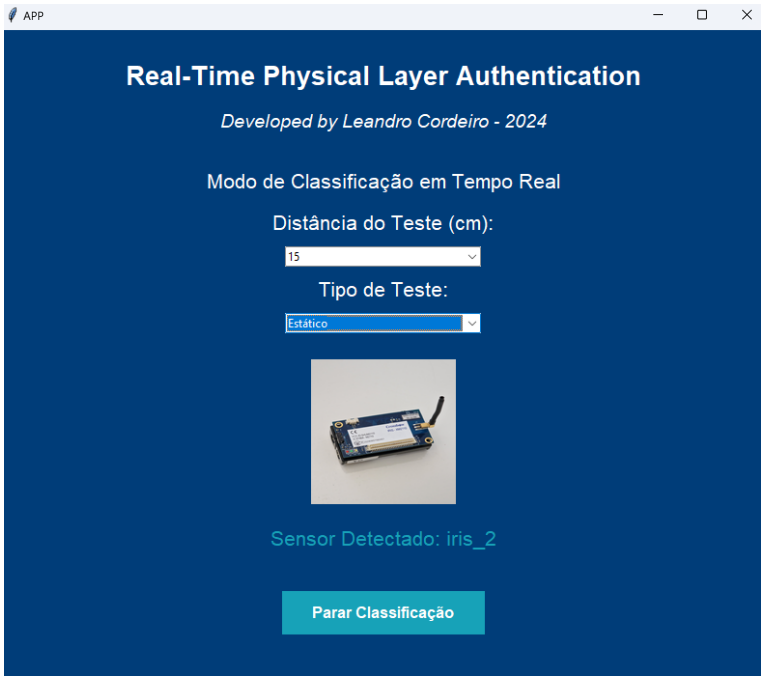

Figure 3.17: System's prediction Interface

If the user wishes to stop sorting, they can do so via a button on the interface, which will take them back to the application's main menu.

To address the need for performance evaluation, the system has been designed to allow the user to test the system's performance in real time. During classification mode, there is an input box where the user can manually enter the number of the sensor they are connecting. Whenever the system makes a prediction, it uses the "get" function, a native Tkinter method, to obtain the value present in that box and store it in a list called `true_labels`. At the same time, the value predicted by the system is stored in another list called predictions. In this way, it is possible to record both the system's prediction and the actual value entered by the user, creating a database for evaluating the system's accuracy.

Figure 4 shows the pseudocode that illustrates this process.

Algorithm 4 Sensor Callback

```
1: function CALLBACK(sensor)
2:     true_label ← entrada_sensor_numero_classificacao.get()      ▷ Gets the sensor number
3:     if true_label ≠ None then
4:         true_labels.append(int(true_label))
5:         sensor_number ← int(sensor.split('_')[1])
6:         predictions.append(sensor_number)
7:     end if
8: end function
```

When the classification session ends, the system checks for values in the `true_labels` and predictions lists. If there are, a specific function is called to calculate performance metrics such as F1-score, accuracy, precision and recall. In addition, this function also generates a confusion matrix, allowing detailed visualization of the system's hits and misses.

After calculating the metrics and generating the confusion matrix, the `true_labels` and predictions lists are automatically cleared to ensure that there are no residues from previous sessions, ensuring the accuracy of the next classification iterations.

4

Results

This chapter covers and analyzes the results obtained from the system developed, as discussed in the previous chapter. First, we discuss the most important metrics for evaluating system performance, giving a clear overview of the key indicators. Then, the different test scenarios are described, along with their configurations to measure effectiveness. After that, we look at how aligning the training and testing impacts the system's ability to classify data. Finally, the last section summarizes the main results, highlighting the key findings.

4.1 Analysis of Key Performance Metrics

With regards of testing we have to define whether a machine learning model is executing the job properly in case of correct and incorrect results. Most of the time, the effectiveness of the system is assessed by the difference of the predicted values by the model and the expected values. One of the ways used to do this evaluation is using accuracy. Of all the assessment parameters, accuracy stands out to be a valuable measure of a system's effectiveness. However, while the criterion of accuracy shows the percentage of correct predictions, it has its drawbacks, for example when working with multi-class classification issues or taking into consideration the imbalance of the number of samples in classes. For instance, a model can be having an accuracy of 90 percent while in real sense the model always fails in a certain class.

In evaluating classification models in machine learning, it is essential to understand the model's successes and failures. To achieve this, the **confusion matrix** is a valuable tool as it provides a clear overview of the predictive performance

In most cases, evaluating classification models entail understanding of the correct prediction and the wrong one in the machine learning. The confusion matrix is a perfect tool for this work, allowing the user to quickly compare the results of the prediction algorithms with the real situation. Figure 4.1 below displays a confusion matrix with two classes; the positive and the negative results.

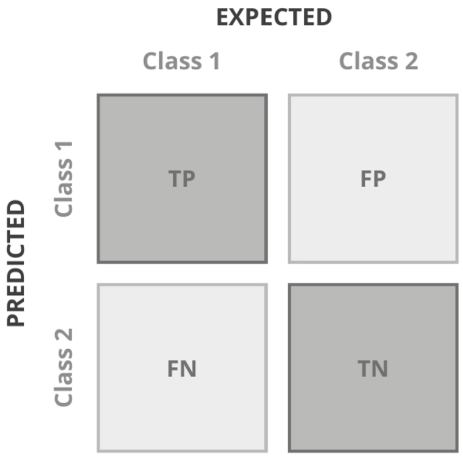

Figure 4.1: Binary Confusion Matrix

In this matrix, True Positives **TP** means the actual positive case is correctly predicted as positive, False Positives **FP** are errors in which actual negative case is mistakenly predicted as positive, True Negatives **TN** are actual negative cases which are correctly predicted as negative and False Negatives **FN** are actual positive cases which are incorrectly predicted as negatives.

When there are more classes in the classification system, the confusion matrix will have several classes as shown below in figure. In the matrix, the rows and columns represent a particular class which helps you dissect the working of the model in a respective class.

From the confusion matrix, several essential metrics are derived to evaluate the model's performance:

Accuracy measures the proportion of correct predictions in relation to the total number of predictions. It can be calculated by

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{4.1}$$

Precision indicates the proportion of positive predictions that are correct. It can be computed by

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{4.2}$$

Recall measures the model's ability to correctly identify all positive examples. It is calculated by

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{4.3}$$

F1-Score combines precision and recall, being particularly useful in situations where there is balance between these two metrics, and it is computed by

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{4.4}$$

These metrics provide an overall picture of the model's performance quite accurately. However, in our particular case, we are dealing with a more complex situation: a multi-class classification problem. This is because we are working with several sensors, each belonging to a different class. That means we would have to apply a multi-class confusion matrix instead of a binary one where only two classes are considered: positive and negative. This will enable us to correctly grade the classifier's performance when there are more than two possible outputs, as with our system, which can classify multiple sensors. For example, if the classifier has to choose between four sensors, it needs to have the capability of correctly categorizing between four possible outputs.

So, the confusion matrix of the multi-class problem is the same principle, just with higher dimensions. Not only two categories that represent positive and negative, respectively, but each row or column represents one class. The cases where the prediction was correct are along the main diagonal of the matrix: class-1 predictions were predicted as 1, class-2 predictions were predicted as 2, and so forth. The Off-diagonal cells of the matrix include the errors the model has made and show which classes were confused for others. Below, Figure 4.2 shows an example of a multi-class confusion matrix, where four sensors are classified. Each cell in the matrix indicates the number of predictions made by the classifier for each pair of true and predicted classes.

Figure 4.2: MultiClass Confusion Matrix

The evaluation of the multiclass confusion matrix also allows the derivation of the same metrics described above: Precision, Accuracy, Recall and F1-Score, but calculated for each individual class.

4.2 Training Significance in System Performance

This section discusses the need to align the training conditions of a machine learning system with those of testing, especially for real-time classification. A model's success has quite a lot to do with its being trained under the same conditions as it will be tested so that, in actual practice, the response is efficient and effective. Most models trained outside the conditions corresponding to reality also do not generalize well, decreasing performance.

Here, we consider three key varieties: first, we consider a situation in which there is, versus is not, a correspondence between the training and test conditions. Second, we look at the changes resulting from the exchange of hardware elements, such as antennas or amplifiers, even from the same manufacturer. Third, we look at the differences that result simply from changing the operating environment- that is, system performance across environments in which the system was trained versus tested.

The set of tests and comparisons shown here will allow us to assess how important it is to maintain coherence between training conditions and operating conditions, underlining the relevance of this phase on the accuracy, response time, and robustness of the system.

4.2.1 Testing Scenarios

When it comes to the test scenarios used to test the system, they were designed to work in a controlled environment using both static and dynamic scenarios. Each scenario was designed to replicate the system's behavior in different conditions, varying the distance between the sensors and the antenna and whether or not there was movement. The tests were carried out inside a room, where the sensors were placed on a stable table, providing an environment with control over external variables.

The training environment was chosen to minimize external interference and ensure that the variability in the signals resulted mainly from changes in the scenario, such as the distance and movement of the sensors. The table provided a fixed reference to ensure consistency in the tests, both in static and dynamic scenarios.

Static Training Scenario

In the static scenario, both during training and testing, the sensors remained in fixed positions throughout the process. The aim of this type of training is to evaluate the system's performance when the distance between the sensors and the antenna is kept constant. To simulate different conditions, several static tests were created, where the variable that changed was the distance of the sensors from the antenna. Tests were carried out at 15, 40 and 100 cm. Throughout this type of scenario, the signals are captured without any interference from movement or changes in the position of the sensors. Figure 4.3 shows the layout of the sensors and antenna, helping to visualize how they were arranged during the tests.

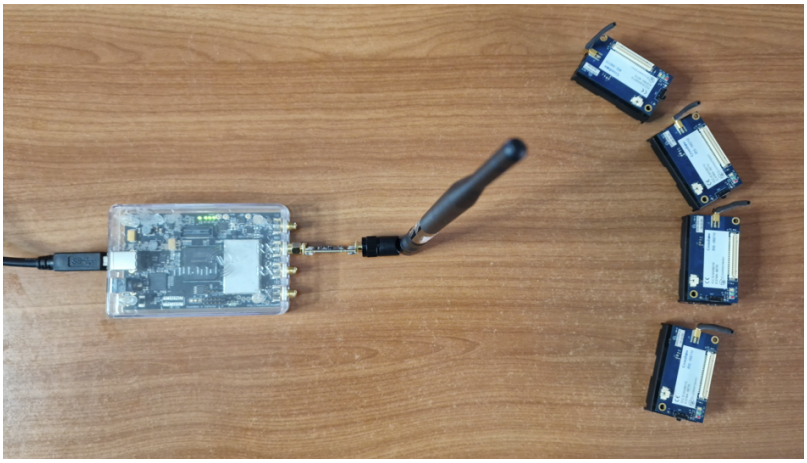

Figure 4.3: Static Scenario

Dynamic Training Scenario

In the dynamic scenario, the aim is to evaluate the system's reaction to the introduction of movement in the sensors. Unlike the static scenario, here the position of the sensors varies throughout the test. Different scenarios were created where the sensors describe a certain pattern, as illustrated in Figure 4.4.

The numbers shown in Figure 4.4 represent the sequence of movements the sensor follows during the dynamic test, moving through positions 1, 2, 3, and 4 according to the specified pattern. This pattern has a maximum distance from the antenna of 15, 40 or 100 cm, just like in the static scenario. This scenario was designed to simulate a more realistic environment where the sensors are moving, but the system should still be able to classify them correctly. In addition, these tests make it possible to analyze the impact that movement has on the performance of a machine learning-based classification environment.

The movement of the sensors was carried out at a constant speed of around 1 m/s, ensuring that the variation in position was controlled and consistent throughout the tests.

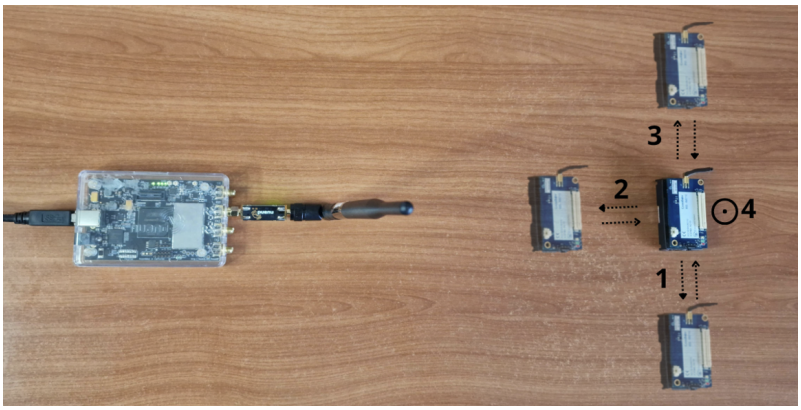

Figure 4.4: Dynamic Scenario

4.2.2 Training-Test Alignment

This subsection will discuss the performance of the system when operating under conditions different from those it was trained for. A few tests and some comparisons will be made, showing their results and allowing an assessment of how the various scenarios affect the system's performance. The purpose will be to see the performance difference between conditions when the training and testing conditions remain alike and in cases where the conditions are quite different.

Specific details are provided in the following sections with regard to the metrics that have been used for measuring the system's performance, which include accuracy, F1-score, precision, and recall. The comparison will underpin the need to match training conditions with actual operating conditions, where consistency between both greatly enhances the precision and effectiveness of this system, making it robust enough and adaptable to all those factors that would be present during real-world operations.

Below, we present the results for the different training and testing scenarios. For each scenario, we show the performance metrics in the environment in which the classifier was trained, as well as the metrics and confusion matrix for the environment in which it was tested. This analysis will make it possible to assess the impact of the discrepancy between training and testing on the system's performance. The confusion matrices for both scenarios can be found in Appendix A.1.

Scenario 1: Impact of Distance in a static scenario
In this scenario, the classifier was trained in a static environment with the sensors 15 cm away from the antenna and was later tested in a static environment where the sensors were 1 meter away. The goal is to evaluate how the increase in distance impacts the classification accuracy when the classifier is trained under different conditions. For these comparisons, 4 sensors were used, with each sensor remaining active for approximately 15 seconds. Below are the metrics for both the training and testing environments.

Train: 15cm distance

Accuracy: 97%
F1-Score: 0.97
Precision: 97%
Recall: 97%

Test: 100cm distance

Accuracy: 90%
F1-Score: 0.90
Precision: 90%
Recall: 90%

Analysis of this scenario shows that when the classifier was trained with sensors at a distance of 15 cm, the system achieved excellent performance, with an F1-score of 0.97.

However, when testing the same classifier with sensors at a distance of 1 meter, the F1-score, while still strong at 0.90, dropped slightly. This indicates that although the system classifies well, the increased distance reduces performance slightly, underscoring the importance of aligning training conditions with testing conditions to optimize performance in real-world situations.

Scenario 2: Impact of Distance and Movement
In this second scenario, the classifier was trained in a static environment with the sensors 15 cm away from the antenna. It was then tested in a dynamic environment, where the sensors were moving at a distance of 1 meter. The objective here is to evaluate how both the increase in distance and the introduction of movement impact the classification accuracy, considering that the system was originally trained under static conditions. Below are the metrics for both the training and testing environments.

Training: 15cm static

Accuracy: 97%
F1-Score: 0.97
Precision: 97%
Recall: 97%

Test: 100cm dynamic

Accuracy: 89%
F1-Score: 0.89
Precision: 89%
Recall: 89%

As we can see from the metrics, although the system continues to classify each sensor relatively well, performance tends to decrease as we move away from the conditions for which it was trained. The F1-score, which was 0.97, drops to 0.89 when movement and a greater distance between the sensors and the antenna are introduced. This reflects how the system's performance is affected by changes in operating conditions, especially in scenarios with different variables from the original training.

Scenario 3: Impact of Distance in a static scenario

In this case, the model was initially trained with the sensors positioned 1 meter away from the antenna, under static conditions. However, during the test, the distance was drastically reduced, with the sensors moved to just 15 cm from the antenna. This change was intended to assess how reducing the distance between the sensors and the antenna impacts classification accuracy, given that the model was originally trained to work with greater distances. Below are the metrics and confusion matrices for the training and test environments.

Training: 100cm distance

Accuracy: 94%
F1-Score: 0.94
Precision: 94%
Recall: 94%

Test: 15cm distance

Accuracy: 85%
F1-Score: 0.85
Precision: 85%
Recall: 85%

This scenario clearly shows a significant drop in performance when the model, trained for distances greater than 1 meter, is used to classify data from a much smaller distance of 15 cm. Although the classifier still has a reasonable accuracy and F1-score of 0.85, this performance is lower than in the training environment, where both metrics were 0.94. This indicates that the model's classification ability is strongly affected by the drastic reduction in distance.

Scenario 4: Impact of Distance and Movement
using a 1m static classifier on a 15cm Dynamic Scenario In this last scenario, the classifier was trained in a static environment with the sensors positioned 1 meter away from the antenna. It was then tested in a dynamic environment, where the sensors were moving at a much closer distance of 15 cm from the antenna. Below are the metrics and confusion matrices for both the training and dynamic testing environments.

Train: 100cm static

Accuracy: 94%
F1-Score: 0.94
Precision: 94%
Recall: 94%

Test: 15cm dynamic

Accuracy: 91%
F1-Score: 0.91
Precision: 91%
Recall: 91%

Looking at these results, we can see that adding the motion component to the classification causes the system's performance to drop. However, in this scenario, due to the proximity of the sensors to the antenna, the decrease in performance is not so pronounced. In the test, the classifier obtained an F1-score of 0.91, slightly lower than the 0.94 in the

training environment. We can therefore conclude that both the increase in distance from the antenna and the change in the type of training the system was subjected to directly affect the system's performance.

4.2.3 Hardware Influence on System's Performance

In this subsection, we will examine the behavior of identical hardware components, specifically multiple Tri-Band antennas and the BladeRF LNA BT-200 amplifier, in our system, which operates directly on physical layer signals. While these components are from the same manufacturer and model, slight variations in physical and electrical parameters may influence the received signal's behaviour and, consequently, the system's performance.

The Tri-Band antenna covers a frequency range from 700 MHz to 2600 MHz. The antenna gain is 5 dBi, and the maximum VSWR is 1.5. Although this antenna is designed to maintain consistency in frequency response, polarization, and gain, minor variations may still arise due to the manufacturing process. These variations can directly affect the quality of the received signal, particularly in sensitive systems that rely on physical-level features. With low noise, the BladeRF LNA BT-200 amplifier was designed to provide an enhanced signal level without significantly introducing noise. However, variations in the noise factor and linearity between the identical model amplifiers may cause differences in amplification and create inherent differences in the signal at the physical level.

The aim of this subsection is to assess whether or not these variations between identical components have a significant impact on the accuracy and robustness of the classifications made by our machine learning system.

4.2.3.1 Antenna Performance Comparison

The exchange of identical antennas, from the same manufacturer and model, can have an impact on the performance of the machine learning system. To investigate this impact, tests will be carried out in three different scenarios, comparing the system's performance with a new antenna identical to the one used during training. In the first scenario, the sensors will be placed 15 cm from the antenna, in a static environment, to check that proximity maintains consistency in classification. In the second scenario, the sensors will be placed 1 meter from the antenna, also in a static environment, to analyze whether increasing the distance generates variations in signal quality with the new antenna. In the third and final scenario, the system will be tested with moving sensors 1 meter from the antenna, simulating a dynamic environment. The aim is to identify whether small variations in identical hardware affect the robustness and accuracy of the classification, comparing the results with those obtained during training.

Scenario 1: 15cm Static In this scenario, the sensors are 15 cm from the antenna in a static environment. Below are the performance metrics comparing Antenna 1 (trained)

and Antenna 2:

Antenna 1 Metrics:

Accuracy: 99%
F1-Score: 0.99
Precision: 99%
Recall: 99%

Antenna 2 Metrics:

Accuracy: 97%
F1-Score: 0.97
Precision: 97%
Recall: 97%

Scenario 2: 1m Static In this scenario, the sensors are 1 meter from the antenna, in a static environment. Below are the performance metrics for the two antennas:

Antenna 1 Metrics:

Accuracy: 98%
F1-Score: 0.98
Precision: 98%
Recall: 98%

Antenna 2 Metrics:

Accuracy: 98%
F1-Score: 0.98
Precision: 98%
Recall: 98%

Scenario 3: 1m Dynamic In this scenario, the sensors are 1 meter from the antenna, but in motion, simulating a dynamic environment. The comparative metrics are:

Antenna 1 Metrics:

Accuracy: 99%
F1-Score: 0.99
Precision: 99%
Recall: 99%

Antenna 2 Metrics:

Accuracy: 99%
F1-Score: 0.99
Precision: 99%
Recall: 99%

After analyzing the results for these 3 scenarios, it is possible to conclude that the fact that we trained the system with a given antenna and then tested it with another antenna of the same model and manufacturer did not introduce any significant differences in the system's performance. The only scenario where it is possible to see a change in performance is the first one, where the f1 score dropped from 0.99 to 0.97. This represents a slight decrease, likely because when the sensor is very close to the antenna, even small differences in the angle or position of the antenna relative to the table can be magnified and affect performance. Nevertheless, even in this case, both antennas perform very well.

4.2.3.2 Amplifier impact (LNA)

In this subsection, we analyze the impact of the BladeRF LNA BT-200 amplifier on the system's performance. Three different tests were carried out, varying the distance between the sensors and the antenna (15 cm and 1 meter) and introducing movement or not. The aim is to assess how changing the amplifier, even if it's the same model and manufacturer, influences the classification in these different scenarios.

Additionally, it is important to consider the antenna gain, which was 0.75 dB in this setup. While this gain improves the signal reception, it may also cause signal saturation in some cases, particularly when the signal strength is already high.

Scenario 1: 15cm Static In this scenario, the sensors were placed at a distance of 15 cm from the antenna, without any kind of movement.

LNA 1 Metrics:

Accuracy: 99%
F1-Score: 0.99
Precision: 99%
Recall: 99%

LNA 2 Metrics:

Accuracy: 96%
F1-Score: 0.96
Precision: 96%
Recall: 96%

Scenario 2: 1m Static In this scenario, the sensors were positioned 1 meter from the antenna without any movement.

LNA 1 Metrics:

Accuracy: 98%
F1-Score: 0.98
Precision: 98%
Recall: 98%

LNA 2 Metrics:

Accuracy: 95%
F1-Score: 0.95
Precision: 95%
Recall: 95%

Scenario 3: 1m Dynamic Finally, in the 1-meter dynamic scenario, the sensors were moved constantly during the test.

LNA 1 Metrics:

Accuracy: 98%
F1-Score: 0.98
Precision: 98%
Recall: 98%

LNA 2 Metrics:

Accuracy: 98%
F1-Score: 0.98
Precision: 98%
Recall: 98%

In conclusion, when analyzing the results of changing the LNA, it can be seen that, as in the section on changing the antenna, these changes did not introduce significant differences in the system's classification performance. In fact, in the three scenarios evaluated, only the first two showed a slight reduction of 0.03 in the f1-scores, which shows that, despite having some impact, the LNAs, being from the same manufacturer, have practically no effect on the system's performance.

4.2.4 Environmental Impact

The environment in which the sensors are tested can have a significant impact on the performance of the classification system. Changing the environment can be seen as altering the communication channel, since physical characteristics such as the layout of the space, the presence of obstacles and the density of objects directly influence signal propagation. These factors result in variations in attenuation, reflection and interference, which significantly affect the accuracy of the classification system.

In this subsection, the results obtained in two different environments will be compared: a closed room, where conditions are more controlled, and an outdoor area, where the system is subject to environmental interference. These tests aim to identify how external factors, such as the presence of obstacles, signal reflections and interference, affect the system's performance.

This subsection will present some results where the system was trained in an indoor, controlled environment and then tested in an outdoor environment.

Environment 1: Closed room

In the first environment, the tests were conducted inside a room of regular dimensions, with the sensors and antenna positioned on a stable table. This environment allowed for greater control over external variables, minimizing interference and reflections. The walls of the room were carefully considered to avoid significant impacts on the measurements.

Environment 2: Outdoor Area

In the second environment, the tests were carried out outdoors in the middle of a garden, where the system was subject to unpredictable conditions like light wind and no nearby walls that could affect signal reflection. The open space allowed for a more natural setting without barriers, making it ideal for assessing how wind and other environmental factors influence the system's classification accuracy compared to the more controlled indoor environment. This setup aimed to capture the potential impacts of real-world conditions on performance.

4.2.4.1 Performance Comparison between Both Environments

The scenarios below compare the system's performance in the two environments for three different configurations: 15 cm static, 1 meter static, and 1 meter dynamic. The performance metrics and confusion matrices are presented for each scenario. The tests were always carried out in the same way, where 4 sensors were placed on a table in the middle of the garden and all transmitted for the same period of time.

Scenario 1: 15cm Static

Metric	Indoor	Outdoor
Accuracy	97%	46% ± 4%
F1-Score	0.97	0.46 ± 0.04
Precision	97%	46% ± 4%
Recall	97%	46% ± 4%

Table 4.1: Comparison of Indoor and Outdoor Metrics - Scenario 1

Scenario 2: 1m Static

Metric	Indoor	Outdoor
Accuracy	95%	80.67% ± 17.30%
F1-Score	0.95	0.81 ± 0.17
Precision	95%	80.67% ± 17.30%
Recall	95%	80.67% ± 17.30%

Table 4.2: Comparison of Indoor and Outdoor Metrics - Scenario 2

Scenario 3: 1m Dynamic

Metric	Indoor	Outdoor
Accuracy	97%	95.33% ± 2.35%
F1-Score	0.97	0.95 ± 0.02
Precision	97%	95.33% ± 2.35%
Recall	97%	95.33% ± 2.35%

Table 4.3: Comparison of Indoor and Outdoor Metrics - Scenario 3

First, note that the values in the outdoor environment are given as the mean ± the 95% Confidence Interval (CI). The confidence interval provides an estimate of the range within which the real value could be expected to lie, were the test repeated under identical conditions a number of times. This reflects the variation observed in outdoor environments where it is more difficult to control for wind and interference compared to indoor environments.

In indoor environments, the system performed quite well for all scenarios with small fluctuation, as can be depicted by the high F1-Scores ranging from 0.95 to 0.97 and the narrow confidence intervals. This can be explained mainly by the nature of indoor environments, where outside factors such as noise or interference are much lower. Moreover, since it was indoors that the system was trained, it performs much better in similar situations.

A variation in the performance is present, presumably due to ambient effects such as wind, which can move objects like leaves, flowers, and dust, increasing interference, along with other general outdoor conditions that are harder to normalise than indoor conditions. This highlights the importance of using the system in the same environment where it was trained. As the results evidence, the system performs with more consistency indoors, where environmental variables are controlled. The uncertainty in performance increases in the uncontrolled outdoors, as evidenced by the larger confidence intervals in the first two scenarios. Yet again, this reinforces that training the system in the same environment where deployment is expected is critical to ensuring that performance is reliable and predictable.

4.3 Processing Time

The time taken in processing is an important factor in the efficiency of a machine learning system especially where it is in real time. Real-time classification also requires the system to be fast in its functioning as it is in the case of this system. A high processing time also poses a problem in this case because it results in delay and the system is therefore less appealing for real-time classification. Hence, the time taken for a system to accept information and produce a specific output becomes one of the most critical characteristics of a real-time system.

In this section, we discuss the hardware configurations used, the factors that affect processing time and the system's performance in various test scenarios. By analyzing

these elements, we aim to highlight the importance of balancing model complexity and processing speed to maintain an efficient, real-time machine learning system.

Hardware Utilized

The machine learning model was trained and evaluated on an Asus VivoBook S 14 M3402QA laptop with the following hardware specifications:

- **Processor:** AMD Ryzen 7 5800H, 8 cores, 16 threads, up to 4.4 GHz
- **RAM:** 16 GB DDR4
- **GPU:** Integrated Radeon Graphics
- **Operating System:** Windows 11 Home

Data Acquisition System

Data acquisition was conducted using a virtual machine running on VMware with Ubuntu 22.04 as the operating system. The capture system used an OsmoCom Source with the following configuration:

- **Sample Rate:** 4M sps
- **Frequency:** 2.45 GHz
- **Gain:** 750 m
- **Antenna:** RX1
- **Bandwidth:** 4 MHz

In the context of real-time classification, one of the factors with the greatest impact on the system's performance is the time needed to carry out the classification. As described in the previous chapter, the classifier used throughout this work was RF Multiclass. According to the study presented in [59], the author concludes that, in scenarios where time is a critical factor, RF can offer shorter processing times compared to other classifiers, such as MLP. This advantage of RF reinforces its suitability for our context, where the speed of the system's response is essential for the model's effectiveness.

Regarding the processing time of our system, a mechanism was developed that records the time required for each prediction. This process was implemented in Python using timers, where the stopwatch is activated as soon as the system detects the reception of a preamble (640 samples) and is interrupted when the system generates the final prediction. Each prediction time is stored in a data structure, allowing the average processing time to be calculated at the end of each test. This calculation is made by dividing the sum

of all the times recorded by the number of forecasts made. In this way, we get a clear picture of how long the system takes, on average, to process a complete classification. This average processing time value is fundamental for assessing the system's viability in real-time classification contexts.

Algorithm 5 Measure and Calculate Average Processing Time

```
Initialize empty list processing_times
while data is received do
    initial_time ← time.time()
    Receive data ← client_socket.recv(buffer_size)
    ...
    Process samples samples_for_prediction ← [samples]
    prediction ← process_samples(samples_for_prediction, encoder_nn, rf_classifier)
    final_time ← time.time()
    Add final_time − initial_time to processing_times
end while
if processing_times is not empty then
    average_processing_time ← mean(processing_times) × 1000
    Print "Average processing time : average_processing_time ms"
else
    Error: No data was processed
end if
```

Once the code was implemented and tested, the average processing times were calculated in different scenarios. Next, the results of these scenarios will be presented, varying the distance between the sensors and the antenna and the introduction or not of movement to assess the impact of these variables on the system's performance.

The tests were carried out using four sensors, each connected for 15 seconds, totaling 1 minute of testing. During the tests, we varied the distance between the sensors and the antenna (15 cm and 100 cm) and introduced movement or not, in order to measure the impact of these variations on system performance and processing time.

For each scenario, multiple tests were performed, and the table below presents the **average processing time** (in milliseconds) calculated from these tests. The confusion matrices used to assess system performance can be found in the appendices.

The following table shows the **average processing time** for the three scenarios described:

Scenario	Distance	Average Processing Time (ms)
Static	15 cm	211 ± 1.96
Static	1 m	292 ± 74.26
Dynamic	1 m	218 ± 19.74

Table 4.4: Average Processing Time in the Different Scenarios with 95% Confidence Intervals

The average processing times in different scenes show significant differences due to

sensor distances and movement. In a static case at 15cm, the system gives a very steady result in terms of processing time, which indicates that the system is able to work stably when it is at a short distance and not moving. This stability may result from simpler decision trees built during the process of training, given the easier processing of simple patterns at ideal conditions. On the other hand, with a larger distance or moving objects, the system may form more complex trees to deal with the extra variables; this would result in slower processing and possibly lower accuracy due to the added complexity.

However, if we consider the cases for which we increase the distance between sensors and antenna, results are no longer so linear, because in the static case we got higher processing times compared with a dynamic case at the same distance. The higher processing time in the longer distance scenarios is due to the possibility that distance introduces more variables, such as interference and signal attenuation, making classification more complex.

In summary, these results suggest that distance clearly has an impact on signal processing time. At a short distance, the system is more consistent, but when the distance increases significantly, there is an addition of possible variables and interferences in the signal, which makes its processing time more inconsistent. Even so, in general the system shows good results when it comes to classification, achieving average processing times of around 200 to 300 ms.

4.4 Performance Conclusions

System reliability is crucial to guaranteeing user confidence and consistent performance in any operating scenario. Throughout this chapter, several factors that directly affect reliability have been analyzed, including test conditions, the impact of the environment, the hardware used and processing time. The system's ability to maintain predictable and stable behavior under different circumstances is essential to ensure that users can trust the results, regardless of variations in operating conditions.

From the results analyzed, we can observe that the system performance is generically more stable in static rather than dynamic scenarios. For static tests, in which sensors are stationary, the system is able to process data with much finer details and faster response times, reflecting greater stability. In contrast, in dynamic scenarios, the movement of the sensors introduces additional variables, such as the continuous change in position and the possible variation in signal intensity, which can lead to an increase in processing times and a reduction in accuracy. These results indicate that, in situations where accuracy is a priority, using the system in static conditions is preferable.

In addition, we can conclude that the alignment between training and testing is fundamental to guaranteeing the system's accuracy. A system can perform excellently under the conditions for which it was trained, but by increasing the distance, introducing movement or changing the environment, its performance can drop significantly. For example, a classifier that was trained at 15 cm static with an F1-Score of 97% was tested

at 1 meter and its value decreased to 90%. Later, further tests were done at 1 meter with movement, and the performance dropped to 89%.While the system still classifies well, ensuring alignment between training and testing is crucial for maintaining high accuracy.

In our tests, it was not possible to conclude that changing the hardware had a significant impact on the system's performance, since antennas and LNAs from the same manufacturer and model were used in all situations. However, the test environment proved to be one of the most critical factors for performance.

These results really showed that classifiers, which achieved F1-Scores of 97% without any problem when trained and tested in controlled environments, would drop in scores by as much as around 50% when tested outside. This raises the importance of aligning training and testing even more, since the environment in which the system is used drastically affects performance. In outdoor environments, where there is no control over external variables, the results were much more difficult to analyze and significantly less reliable.

In general, the processing time was satisfactory. This actually depends on the distance between sensors and the antenna, where closer distances resulted in more critical response times compared to scenarios with higher separation.

In conclusion, the system shows that it achieves good levels of performance when its training and testing conditions are met. When these conditions are not met, its performance tends to drop dramatically and its results become more difficult to interpret.

5

Conclusions

The main aim of this thesis was to develop and test a real-time Machine Learning system for physical layer authentication using SDR. In addition, a graphical interface was developed that could fully integrate this system and be capable not only of training new models but also of carrying out tests in real-time. To achieve these objectives, we explored some existing algorithms and new approaches capable of operating in real-time during the sample capture process.

Initially, new functionalities were developed for a GNU Radio module, focused on capturing ZigBee signals and accurately segmenting preambles. One of the main additions was the modification of the `preamble_sink_pp.` block, resulting in the `PHY_PREAMBLE_LEANDRO` module. This module was created to ensure that exactly 640 samples, corresponding to the preamble of the ZigBee signals, were extracted and sent directly to the machine learning module. To ensure that the exact number of samples was always captured, a mechanism was introduced that automatically adjusts the size of the sample buffer, correcting any faults in the capture and ensuring the consistency required for correct processing.

After ensuring that the preamble samples were correct, the GNU Radio module was integrated with the machine learning module. First, solutions that proved not to be good enough due to the limited communication bandwidth were tried out. Later, one relying on TCP sockets was chosen and used; it proved acceptable to carry the preamble data, hence ensuring that the integration of the two modules was effective and stable.

The next phase involved using networks based on autoencoders, which can automatically extract features from the signal at the physical level. A Conv1D autoencoder was used, which compressed the data into 12 essential features.

These features were then combined with other variables and used to train a multiclass Random Forest classifier, adjusted to maximize the system's performance. The last part of this thesis was the implementation of a graphical application with two main modes; the training mode allows to integration of new sensors and trains the classifiers with the data obtained from these sensors. The second is the real-time classification mode, where the system can be tested and evaluated continuously, allowing for practical performance

analysis. This application offers an intuitive interface that facilitates both expanding the system with new sensors and monitoring its operation in real-time, making it an effective tool for using and testing the system.

The results obtained demonstrate that the system works most effectively in static scenarios; that is, sensors do not move. Under such conditions, the system reached high values of F1-Score, reaching 97% obtained for 15 cm. In dynamic scenarios, on the other hand, the introduction of movement brought about a decrease in accuracy, where the F1-Score fell as low as 89% with a distance of 1 meter. This proves that the movement of sensors and distance are directly related to system stability.

Alignment between training and testing was one other fundamental reason for accuracy. In fact, the system showed excellent performance, with an F1-Score of 97%, when tested in the same static conditions at 15 cm in which it was trained. However, it was noted that there were severe degradations in performance with variations in distance, movement, or environment. For example, when the classifier trained at 15 cm was tested at a distance of 1 meter, the F1-Score dropped to 90%. Additionally, testing this system in uncontrolled outdoor environments led to even more dramatic drops in performance, with some classifiers seeing their F1 scores fall to around 50%.

Although hardware changes had no significant impact, the test environment was one of the most determining factors for performance. In outdoor environments, the lack of control over external variables made the results less reliable and more difficult to analyze.

In conclusion, the system proved to be effective under controlled conditions, but its performance is strongly impacted by variations in the environment and the alignment between training and testing.

5.1 Future Work

There are several interesting directions in which the work carried out in this thesis could be expanded. One of the main ones would be to test a wider variety of neural networks and classifiers. Due to the time dedicated to the development of the GNU Radio module, its integration with the machine learning module and the development of the graphical application, it was not possible to explore other network and classifier architectures in depth. However, this area represents significant potential for improving the system's performance.

Other future work could be the investigation of different ways of integrating GNU Radio with the machine learning module. While the final solution using TCP sockets worked well, the exploration of alternatives using more robust communication protocols or parallel processing architectures might further improve performance and stability.

In addition, it would be interesting to test the impact of using antennas and LNAs from different models and manufacturers.In this thesis, antennas and LNAs from the same manufacturer and model were used, but testing with varied hardware could provide

valuable insights into how the system responds to different physical conditions and equipment.

In summary, these areas represent opportunities for research and improvement that could lead to interesting discoveries and enhance the system, allowing it to reach new levels of performance and adaptability.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (cit. on p. i).

[2] J. Lin et al. "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications". In: *IEEE Internet of Things Journal* 4.5 (2017), pp. 1125–1142. DOI: 10.1109/JIOT.2017.2683200 (cit. on p. 1).

[3] D. Wilson. *IOT is creating massive growth opportunities*. 2021-10. URL: https://blogs.cisco.com/internet-of-things/iot-is-creating-massive-growth-opportunities (cit. on p. 1).

[4] K. Zeng, K. Govindan, and P. Mohapatra. "Non-cryptographic authentication and identification in wireless networks". In: *IEEE Wireless Communications* 17 (5 2010-10), pp. 56–62. ISSN: 15361284. DOI: 10.1109/MWC.2010.5601959 (cit. on p. 5).

[5] N. Xie, Z. Li, and H. Tan. "A Survey of Physical-Layer Authentication in Wireless Communications". In: *IEEE Communications Surveys & Tutorials* 23.1 (2021), pp. 282–310. DOI: 10.1109/COMST.2020.3042188 (cit. on pp. 5, 6, 8, 9, 16).

[6] W. Hou et al. "Physical Layer Authentication for Mobile Systems with Time-Varying Carrier Frequency Offsets". In: *IEEE Transactions on Communications* 62.5 (2014), pp. 1658–1667. DOI: 10.1109/TCOMM.2014.032914.120921 (cit. on p. 5).

[7] L. Bai et al. "Physical layer authentication in wireless communication networks: A survey". In: *Journal of Communications and Information Networks* 5.3 (2020), pp. 237–264. DOI: 10.23919/JCIN.2020.9200889 (cit. on p. 5).

[8] V. Kumar, J.-M. J. Park, and K. Bian. "PHY-Layer Authentication Using Duobinary Signaling for Spectrum Enforcement". In: *IEEE Transactions on Information Forensics and Security* 11.5 (2016), pp. 1027–1038. DOI: 10.1109/TIFS.2016.2516904 (cit. on p. 6).

[9] S. Supangkat, T. Eric, and A. Pamuji. "A public key signature for authentication in telephone". In: *Asia-Pacific Conference on Circuits and Systems*. Vol. 2. 2002, 495–498 vol.2. DOI: 10.1109/APCCAS.2002.1115320 (cit. on p. 6).

[10] G. J. Simmons. "Authentication Theory/Coding Theory". In: *Advances in Cryptology*. Ed. by G. R. Blakley and D. Chaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 411–431. ISBN: 978-3-540-39568-3 (cit. on p. 6).

[11] P. L. Yu, J. S. Baras, and B. M. Sadler. "Physical-Layer Authentication". In: *IEEE Transactions on Information Forensics and Security* 3.1 (2008), pp. 38–51. DOI: 10.1109/TIFS.2007.916273 (cit. on p. 7).

[12] P. Zhang et al. "Lightweight Tag-Based PHY-Layer Authentication for IoT Devices in Smart Cities". In: *IEEE Internet of Things Journal* 7.5 (2020), pp. 3977–3990. DOI: 10.1109/JIOT.2019.2958079 (cit. on p. 7).

[13] N. Xie, C. Chen, and Z. Ming. "Security Model of Authentication at the Physical Layer and Performance Analysis over Fading Channels". In: *IEEE Transactions on Dependable and Secure Computing* 18.1 (2021), pp. 253–268. DOI: 10.1109/TDSC.2018.2883598 (cit. on p. 7).

[14] L. Alhoraibi et al. "Physical Layer Authentication in Wireless Networks-Based Machine Learning Approaches". In: *Sensors* 23.4 (2023). ISSN: 1424-8220. DOI: 10.3390/s23041814. URL: https://www.mdpi.com/1424-8220/23/4/1814 (cit. on p. 7).

[15] P. Hao, X. Wang, and A. Refaey. "An enhanced cross-layer authentication mechanism for wireless communications based on PER and RSSI". In: *2013 13th Canadian Workshop on Information Theory*. 2013, pp. 44–48. DOI: 10.1109/CWIT.2013.6621590 (cit. on p. 8).

[16] J. H. Lee and R. M. Buehrer. "Characterization and detection of location spoofing attacks". In: *Journal of Communications and Networks* 14.4 (2012), pp. 396–409 (cit. on p. 8).

[17] D. B. Faria and D. R. Cheriton. "Detecting identity-based attacks in wireless networks using signalprints". In: *Proceedings of the 4th ACM workshop on Wireless security*. Association for Computing Machinery. 2006, pp. 43–52. DOI: 10.1145/1161289.1161298 (cit. on p. 8).

[18] M. Demirbas and Y. Song. "An RSSI-based scheme for sybil attack detection in wireless sensor networks". In: *2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks(WoWMoM'06)*. 2006, 5 pp.–570. DOI: 10.1109/WOWMOM.2006.27 (cit. on p. 8).

[19] F. J. Liu, X. Wang, and S. L. Primak. "A two dimensional quantization algorithm for CIR-based physical layer authentication". In: *2013 IEEE International Conference on Communications (ICC)*. 2013, pp. 4724–4728. DOI: 10.1109/ICC.2013.6655319 (cit. on pp. 9, 10).

[20] L. Xiao et al. "Fingerprints in the Ether: Using the Physical Layer for Wireless Authentication". In: *2007 IEEE International Conference on Communications*. 2007, pp. 4646–4651. DOI: 10.1109/ICC.2007.767 (cit. on p. 9).

[21] D. A. Knox and T. Kunz. "RF Fingerprints for Secure Authentication in Single-Hop WSN". In: *2008 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*. 2008, pp. 567–573. DOI: 10.1109/WiMob.2008.18 (cit. on p. 10).

[22] X. Zhou et al. "A Robust Radio-Frequency Fingerprint Extraction Scheme for Practical Device Recognition". In: *IEEE Internet of Things Journal* 8.14 (2021), pp. 11276–11289. DOI: 10.1109/JIOT.2021.3051402 (cit. on pp. 10, 12–14).

[23] N. Smith et al. "Real-time location fingerprinting for mobile devices in an indoor prison setting". In: *Signal Processing, Sensor/Information Fusion, and Target Recognition XXX*. Vol. 11756. SPIE. 2021, pp. 267–275 (cit. on p. 11).

[24] A. Jagannath, J. Jagannath, and P. S. P. V. Kumar. "A Comprehensive Survey on Radio Frequency (RF) Fingerprinting: Traditional Approaches, Deep Learning, and Open Challenges". In: *arXiv preprint arXiv:2201.00680* (2021) (cit. on pp. 11, 16).

[25] A. C. Polak, S. Dolatshahi, and D. L. Goeckel. "Identifying Wireless Users via Transmitter Imperfections". In: *IEEE Journal on Selected Areas in Communications* 29.7 (2011), pp. 1469–1479. DOI: 10.1109/JSAC.2011.110812 (cit. on p. 12).

[26] X. Zhou et al. "Design of a Robust RF Fingerprint Generation and Classification Scheme for Practical Device Identification". In: *2019 IEEE Conference on Communications and Network Security (CNS)*. 2019, pp. 196–204. DOI: 10.1109/CNS.2019.8802783 (cit. on pp. 13–15).

[27] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 14).

[28] B. Danev, D. Zanetti, and S. Capkun. "On physical-layer identification of wireless devices". In: *ACM Computing Surveys* 45.1 (2012-11), pp. 1–29. ISSN: 1557-7341. DOI: 10.1145/2379776.2379782. URL: http://dx.doi.org/10.1145/2379776.2379782 (cit. on p. 16).

[29] V. Brik et al. "Wireless Device Identification with Radiometric Signatures". In: *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*. MobiCom '08. San Francisco, California, USA: Association for Computing Machinery, 2008, pp. 116–127. ISBN: 9781605580968. DOI: 10.1145/1409944.1409959. URL: https://doi.org/10.1145/1409944.1409959 (cit. on p. 16).

[30] F. zhuo, Y. Huang, and J. chen. "Radio Frequency Fingerprint Extraction of Radio Emitter Based on I/Q Imbalance". In: *Procedia Computer Science* 107 (2017). Advances in Information and Communication Technology: Proceedings of 7th International Congress of Information and Communication Technology (ICICT2017), pp. 472–477. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2017.03.092. URL: https://www.sciencedirect.com/science/article/pii/S1877050917303678 (cit. on p. 16).

[31] W. Hou, X. Wang, and J.-Y. Chouinard. "Physical layer authentication in OFDM systems based on hypothesis testing of CFO estimates". In: *2012 IEEE International Conference on Communications (ICC)*. 2012, pp. 3559–3563. DOI: 10.1109/ICC.2012.6364429 (cit. on pp. 16, 18).

[32] C. Bishop. "Pattern recognition and machine learning". In: *Springer google schola* 2 (2006), pp. 35–42 (cit. on p. 17).

[33] H. Patel. "Non-parametric feature generation for RF-fingerprinting on ZigBee devices". In: *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*. 2015, pp. 1–5. DOI: 10.1109/CISDA.2015.7208645 (cit. on p. 17).

[34] F. A. Cruz. "Near Real-Time RF-DNA Fingerprinting for ZigBee Devices Using Software Defined Radios". Master of Science in Computer Engineering. Department of Electrical and Computer Engineering: Air Force Institute of Technology, 2019-03. URL: https://scholar.afit.edu/etd/2253 (cit. on pp. 17, 27, 28, 32).

[35] R. W. Klein, M. A. Temple, and M. J. Mendenhall. "Application of wavelet-based RF fingerprinting to enhance wireless network security". In: *Journal of Communications and Networks* 11.6 (2009), pp. 544–555 (cit. on p. 17).

[36] M. Lukacs, P. Collins, and M. Temple. "Classification performance using 'RF-DNA' fingerprinting of ultra-wideband noise waveforms". In: *Electronics Letters* 51 (2015-05), pp. 787–789. DOI: 10.1049/el.2015.0051 (cit. on p. 17).

[37] K. Bonne Rasmussen and S. Capkun. "Implications of radio fingerprinting on the security of sensor networks". In: *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops - SecureComm 2007*. 2007, pp. 331–340. DOI: 10.1109/SECCOM.2007.4550352 (cit. on p. 18).

[38] J. Toonstra and W. Kinsner. "Transient analysis and genetic algorithms for classification". In: *IEEE WESCANEX 95. Communications, Power, and Computing. Conference Proceedings*. Vol. 2. 1995, 432–437 vol.2. DOI: 10.1109/WESCAN.1995.494069 (cit. on p. 18).

[39] S. Ur Rehman, K. Sowerby, and C. Coghill. "RF fingerprint extraction from the energy envelope of an instantaneous transient signal". In: *2012 Australian Communications Theory Workshop (AusCTW)*. 2012, pp. 90–95. DOI: 10.1109/AusCTW.2012.6164912 (cit. on p. 18).

[40] C. K. Dubendorfer, B. W. Ramsey, and M. A. Temple. "An RF-DNA verification process for ZigBee networks". In: *MILCOM 2012 - 2012 IEEE Military Communications Conference*. 2012, pp. 1–6. DOI: 10.1109/MILCOM.2012.6415804 (cit. on p. 18).

[41] J. Yu et al. "A Robust RF Fingerprinting Approach Using Multisampling Convolutional Neural Network". In: *IEEE Internet of Things Journal* 6.4 (2019), pp. 6786–6799. DOI: 10.1109/JIOT.2019.2911347 (cit. on p. 19).

[42] S. Hazra, T. Voigt, and W. Yan. "PLIO: Physical Layer Identification using One-shot Learning". In: *2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*. 2021, pp. 335–343. DOI: 10.1109/MASS52906.2021.00050 (cit. on p. 19).

[43] H. Jafarii et al. "IoT Devices Fingerprinting Using Deep Learning". In: *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*. 2018, pp. 1–9. DOI: 10.1109/MILCOM.2018.8599826 (cit. on pp. 19, 20).

[44] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning". In: *Nature* 521.7553 (2015-05), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: http://dx.doi.org/10.1038/nature14539 (cit. on p. 20).

[45] A. Biswal. *Top 10 deep learning algorithms you should know in 2023*. 2023-08. URL: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm#what_is_deep_learning (cit. on pp. 21, 22, 24).

[46] S. Indolia et al. "Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach". In: *Procedia Computer Science* 132 (2018). International Conference on Computational Intelligence and Data Science, pp. 679–688. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.05.069. URL: https://www.sciencedirect.com/science/article/pii/S1877050918308019 (cit. on p. 21).

[47] S. Saxena. *What is LSTM? introduction to long short-term memory*. 2024-01. URL: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/ (cit. on p. 22).

[48] S. S. Khan and B. Taati. "Detecting unseen falls from wearable devices using channel-wise ensemble of autoencoders". In: *Expert Systems with Applications* 87 (2017), pp. 280–290. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2017.06.011. URL: https://www.sciencedirect.com/science/article/pii/S0957417417304281 (cit. on p. 23).

[49] A. Dertat. *Applied deep learning - part 3: Autoencoders*. 2017-10. URL: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798 (cit. on p. 23).

[50] E. Marchi et al. "A novel approach for automatic acoustic novelty detection using a denoising autoencoder with bidirectional LSTM neural networks". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, pp. 1996–2000. DOI: 10.1109/ICASSP.2015.7178320 (cit. on p. 23).

[51] J. Brownlee. *A gentle introduction to generative adversarial networks (Gans)*. 2019-07. URL: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/ (cit. on p. 24).

[52] "IEEE Standard for Low-Rate Wireless Networks". In: *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)* (2020), pp. 1–800. DOI: 10.1109/IEEESTD.2020.9144691 (cit. on p. 25).

[53] Bastibl. *Bastibl/GR-IEEE802-15-4: IEEE 802.15.4 Zigbee Transceiver*. URL: https://github.com/bastibl/gr-ieee802-15-4 (cit. on p. 26).

[54] B. Bloessl et al. "A GNU Radio-based IEEE 802.15.4 Testbed". In: *Proceedings of 12. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze (FGSN 2013)*. 2013-09, pp. 37–40 (cit. on p. 26).

[55] E. Faulkner et al. "An Advanced GNU Radio Receiver of IEEE 802.15.4 OQPSK Physical Layer". In: *IEEE Internet of Things Journal* 8.11 (2021), pp. 9206–9218. DOI: 10.1109/JIOT.2021.3057472 (cit. on pp. 26, 27).

[56] J. F. L. Faria. *RUN: SDR for Physical Layer Authentication*. https://run.unl.pt/handle/10362/152400. (Accessed on 01/17/2024). 2022-11 (cit. on pp. 27–29).

[57] Osh. *Osh/GR-TF: GNU Radio Tensorflow Blocks Repository*. URL: https://github.com/osh/gr-tf (cit. on p. 29).

[58] URL: https://github.com/zeromq/pyzmq/blob/main/examples/draft/client-server.py (cit. on pp. 29, 36).

[59] I. Pereira et al. "Radio frequency fingerprinting using autoencoder generated features on IEEE 802.15.4 networks". In: *IEEE Vehicular Technology Conference (VTC-Spring)*. 2024-06, pp. - (cit. on pp. 32, 38, 40–42, 66).

[60] P. Hintjens. *ZeroMQ Guide*. Available at: http://zguide.zeromq.org/. 2010 (cit. on p. 37).

A

Results

A.1 Training-Test Alignment

A.1.1 Confusion Matrices for Scenario 1

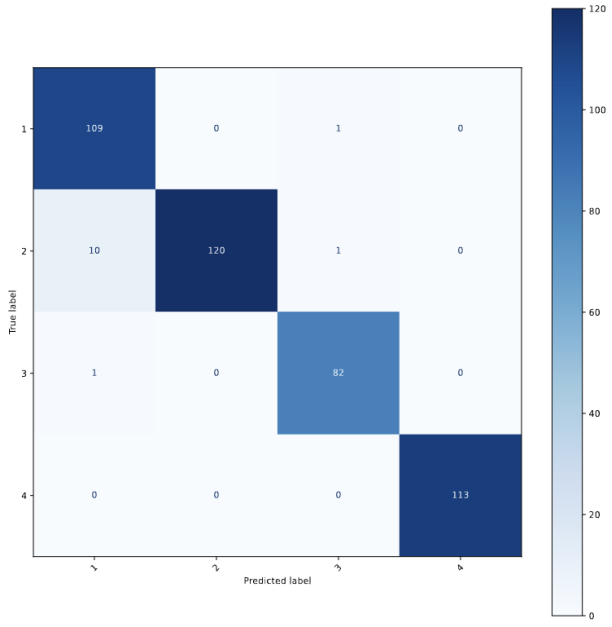

Figure A.1: Confusion Matrix for the 15 cm static training environment.

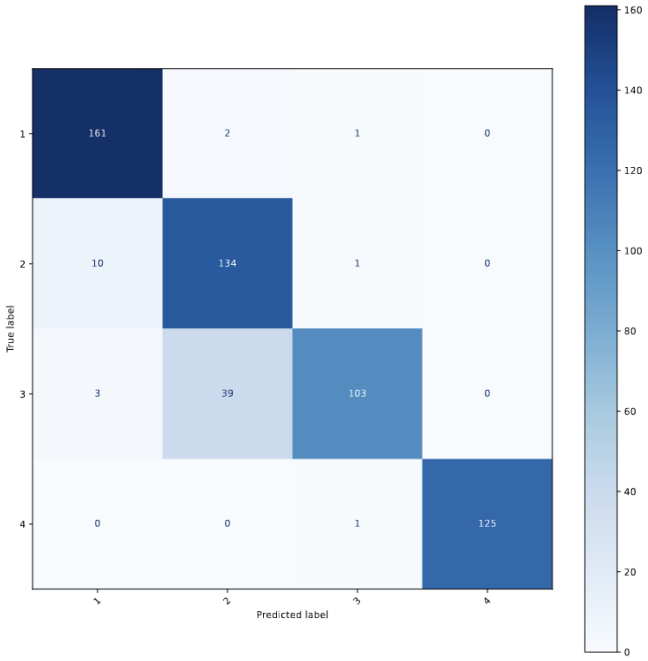

Figure A.2: Confusion Matrix for the 1 meter static testing environment.

A.1.2 Confusion Matrices for Scenario 2

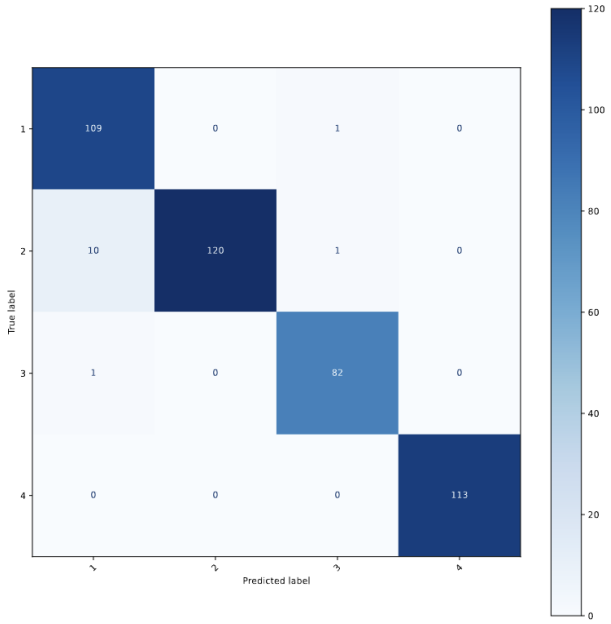

Figure A.3: Confusion Matrix for the 15 cm static training environment.

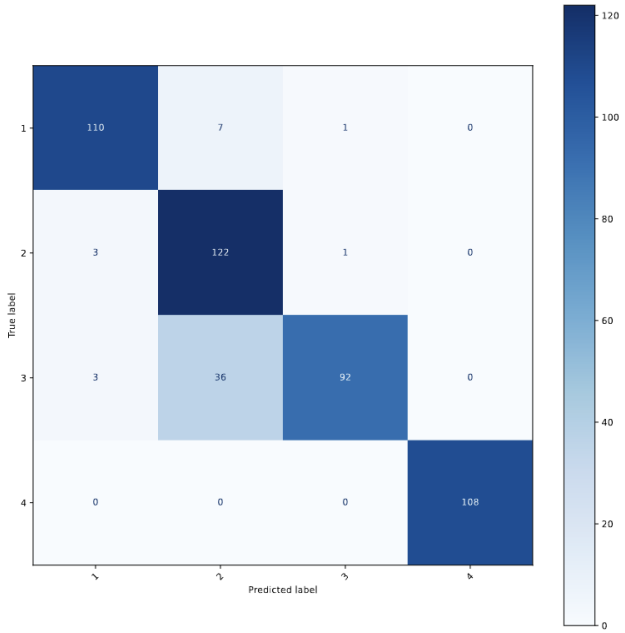

Figure A.4: Confusion Matrix for a 1m dynamic testing environment

A.1.3 Confusion Matrices for Scenario 3

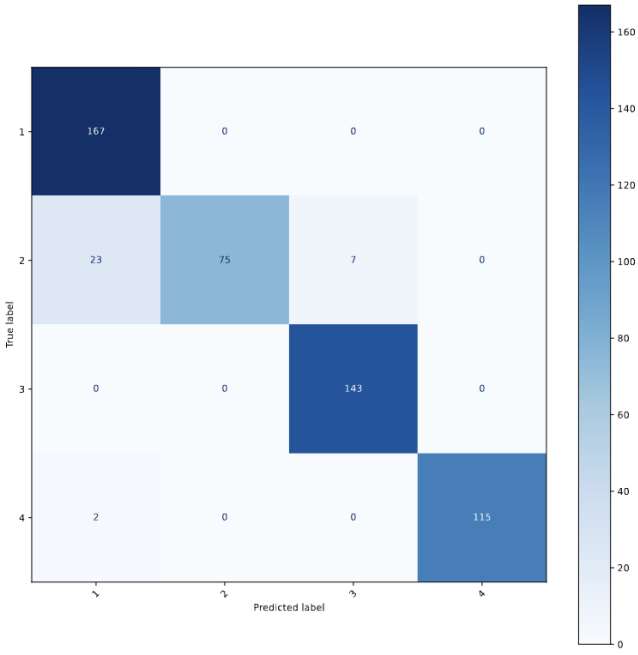

Figure A.5: Confusion Matrix for a 1m static training Environment

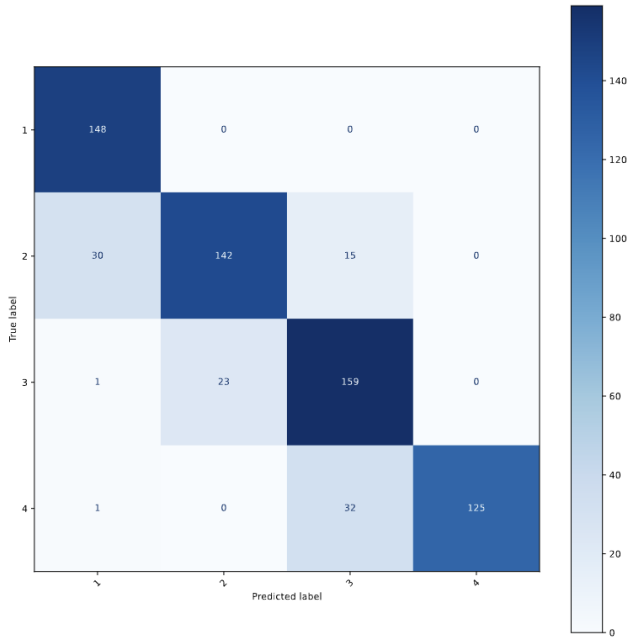

Figure A.6: Confusion Matrix for a 15cm dynamic training Environment

A.1.4 Confusion Matrices for Scenario 4

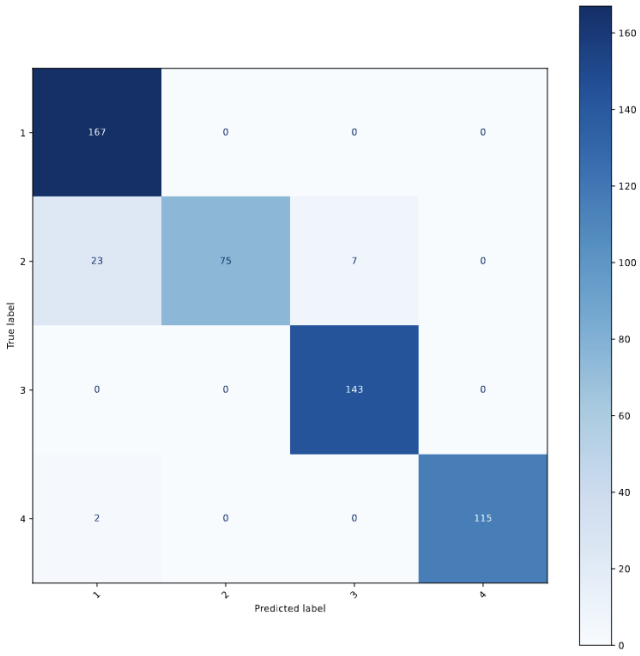

Figure A.7: Confusion Matrix for a 1m static training Environment

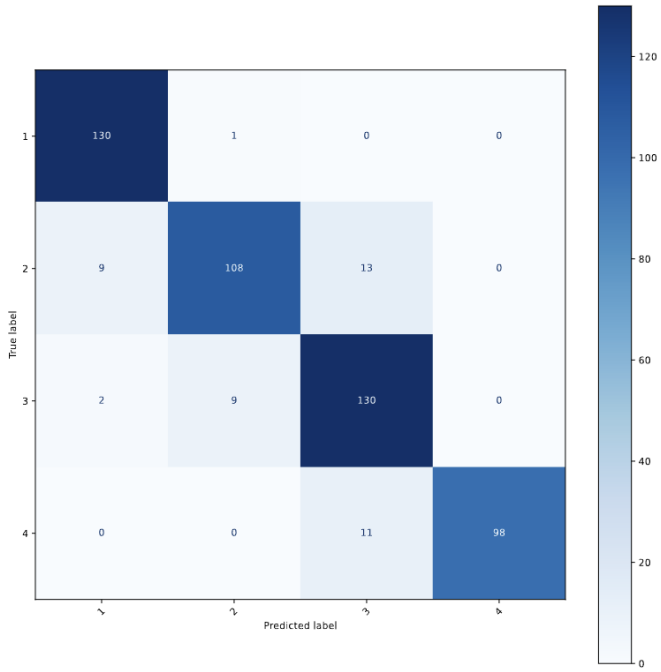

Figure A.8: Confusion Matrix for a 15cm dynamic testing environment

A.2 Environment Impact

A.2.1 Confusion Matrices for Scenario 1

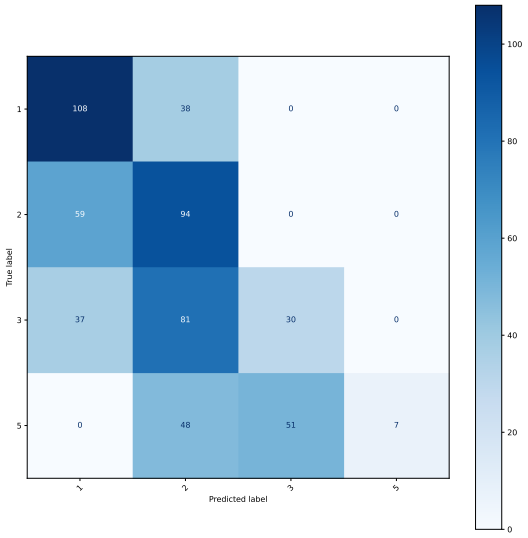

Figure A.9: Confusion Matrix for Test 1.

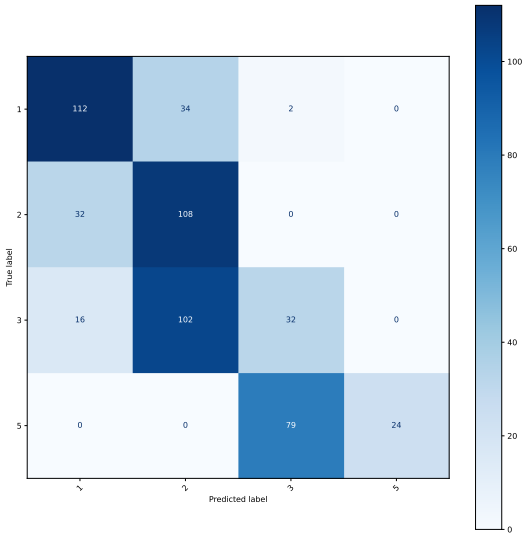

Figure A.10: Confusion Matrix for Test 2.

A.2.2 Confusion Matrices for Scenario 2

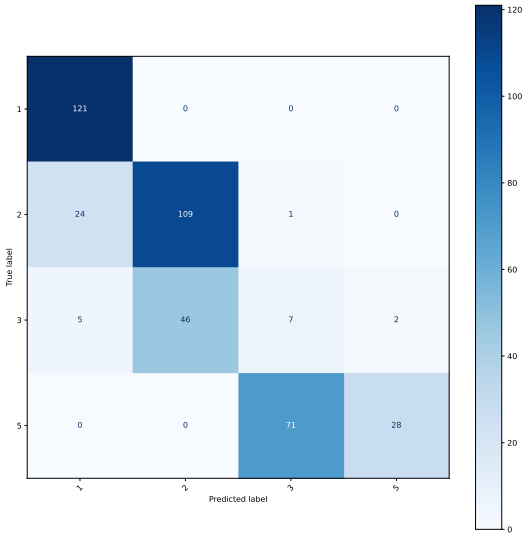

Figure A.11: Confusion Matrix for Test 1.

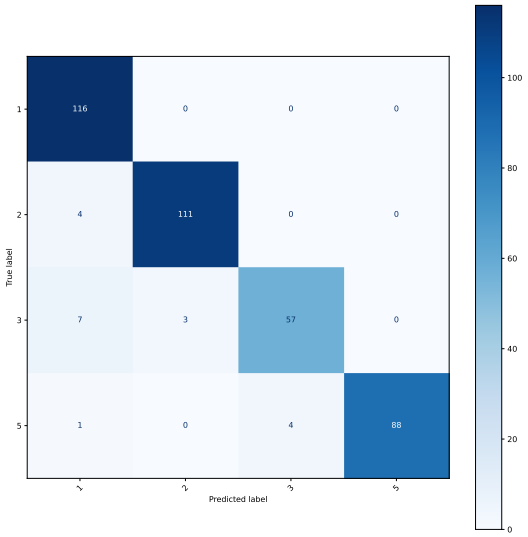

Figure A.12: Confusion Matrix for Test 2.

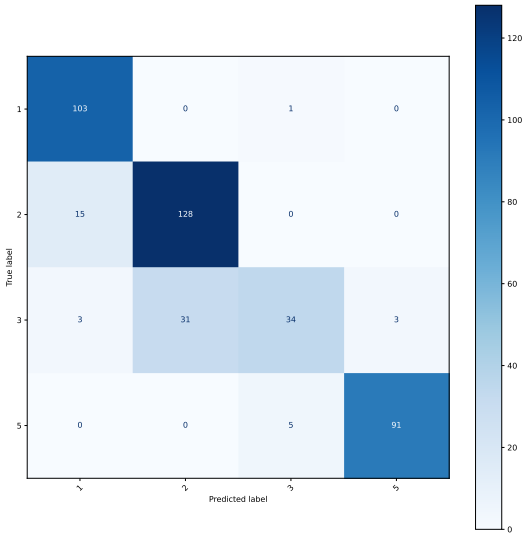

Figure A.13: Confusion Matrix for Test 3.

A.2.3 Confusion Matrices for Scenario 3

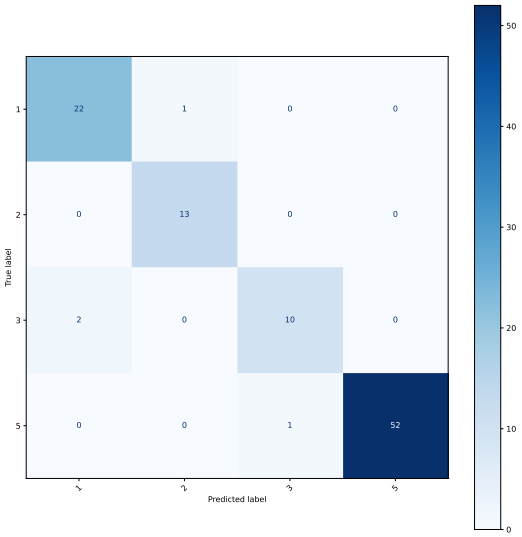

Figure A.14: Confusion Matrix for Test 1.

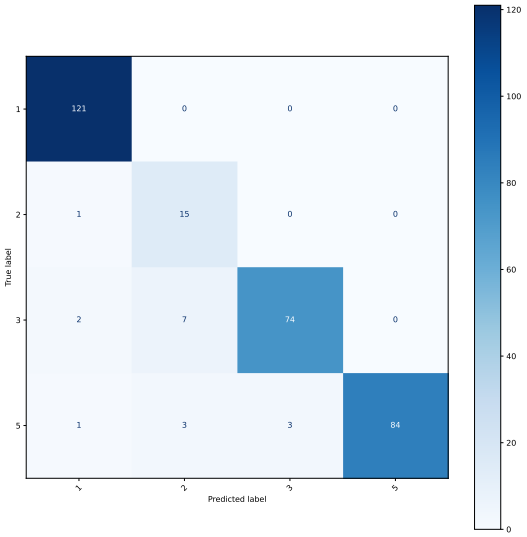

Figure A.15: Confusion Matrix for Test 2.

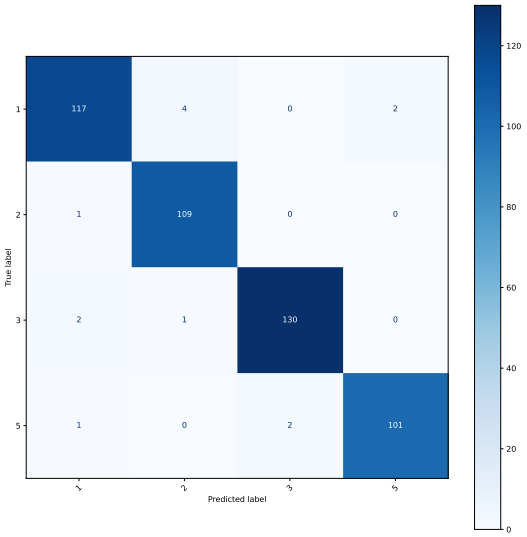

Figure A.16: Confusion Matrix for Test 3.