BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

MONITORING OF BLUETOOTH LOW ENERGY DEVICES

MONITORING BLUETOOTH LOW ENERGY ZAŘÍZENÍ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MATEJ OLEXA**
AUTOR PRÁCE

SUPERVISOR **Mgr. KAMIL MALINKA, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

164339

Institut:	Department of Intelligent Systems (DITS)
Student:	**Olexa Matej**
Programme:	Information Technology
Title:	**Monitoring of Bluetooth Low Energy Devices**
Category:	Security
Academic year:	2024/25

Assignment:

1. Learn how to establish a connection with Bluetooth LE.
2. Study simple neural networks like Multi-layer perceptron (MLP).
3. Optimize an existing tool prototype for monitoring advertising channels on the ESP platform.
4. Capture a dataset with behavior profiles of different devices on advertising channels during passive operation and active connection.
5. Train and test selected neural networks on the connection recognition problem and identify intermittent and persistent advertising patterns.
6. According to the results of the experiment, the tool should be extended to support the most efficient method.

Literature:

- HUJŇÁK Ondřej, MALINKA Kamil a HANÁČEK Petr. Indirect Bluetooth Low Energy Connection Detection. In: *2023 International Conference on Information Networking (ICOIN)*. Bangkok: Institute of Electrical and Electronics Engineers, 2023, s. 328-333. ISBN 978-1-6654-6268-6.

Requirements for the semestral defence:
Items 1 to 4.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Malinka Kamil, Mgr., Ph.D.**
Head of Department:	Kočí Radek, Ing., Ph.D.
Beginning of work:	1.11.2024
Submission deadline:	14.5.2025
Approval date:	31.10.2024

Abstract

This thesis addresses passive monitoring of Bluetooth Low Energy (BLE) connections through hardware and algorithmic improvements. We develop a parallel monitoring system using three ESP32 microcontrollers, each dedicated to a primary advertising channel, which we use to capture a dataset of BLE advertising packets. Then, we solve generalization problems in existing detection methods by implementing multi-input neural networks (both MLPs and 1D CNNs) that process sequences of advertising data. Then, we deploy the trained model onto our newly created probe, which works as real-time connection detector. We also create a web application GUI for the probe, which allows users to visualize the detected connections.

Abstrakt

Táto práca sa zaoberá pasívnym monitorovaním pripojení Bluetooth Low Energy (BLE) prostredníctvom hardvérových a algoritmických zlepšení. Vyvíjame paralelný monitorovací systém použitím troch mikrokontrolérov ESP32, z ktorých je každý vyhradený pre jeden primárny 'advertisement' kanál, ktorý používame na zachytávanie údajov 'advertisement' paketov BLE. Potom, riešime problémy so zovšeobecnením existujúcich metód detekcie, implementáciou viacvstupových neurónových sietí (MLP aj 1D CNN), ktoré spracúvajú sekvencie 'advertisement' údajov. Potom nasadíme natrénovaný model na našu novovytvorenú sondu, ktorá funguje ako detektor pripojenia v reálnom čase. Pre sondu vytvárame aj grafické rozhranie webovej aplikácie, ktoré umožňuje používateľom vizualizovať zistené spojenia.

Keywords

Bluetooth Low Energy (BLE), passive monitoring, connection detection, advertising analysis, machine learning, neural networks, Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), generalization, sequence processing, ESP32, IoT Security, real-time detection, hardware implementation

Klíčová slova

Bluetooth Low Energy (BLE), pasívne monitorovanie, detekcia pripojenia, analýza advertising paketov, strojové učenie, neurónové siete, Viacvrstvový perceptrón (MLP), Konvolučná neurónová sieť (CNN), generalizácia, spracovanie sekvencií, ESP32, bezpečnosť IoT, detekcia v reálnom čase, hardvérová implementácia

Reference

OLEXA, Matej. *Monitoring of Bluetooth Low Energy Devices*. Brno, 2025. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Kamil Malinka, Ph.D.

Rozšířený abstrakt

Pasívne monitorovanie pripojení zariadení Bluetooth Low Energy (BLE) prostredníctvom analýzy časovania 'advertisement' paketov je sľubná nízkonákladová metóda pre bezpečnosť IoT, najmä v SOHO prostrediach s chýbajúcou infraštruktúrou. Základný koncept predstavil O. Hujňák [14], ktorý ukázal, že mnoho BLE zariadení pri pripojení menia aktivitu vysielania 'advertisement' paketov. Pripojenie, často spojené s akciou (napr. odomknutie zámku), je teda kritickým bezpečnostným predmetom. Výzvy ako preskakovanie frekvencií (FHSS) a priame spojenia však komplikujú tradičné monitorovanie.

Nadväzujúci výskum [15], na ktorom sa podieľal aj autor práce, sa zameral na spoľahlivejšie zachytávanie dát a tvorbu datasetu. Bola vytvorená paralelná monitorovacia sonda s tromi ESP32 mikrokontrolérmi, každý pre jeden 'advertisement' kanál (37-39), s cieľom minimalizovať stratu paketov. Hoci kvalita dát stúpla len mierne (približne o 5% menej strát paketov oproti štandardnému prijímaču), hlavným prínosom bol rozsiahly, verejne dostupný a anotovaný dataset. Tento dataset z rôznych typov zariadení (zámky, senzory, atď.) a prostredí (tienená komora, kancelária) slúži ako zdroj pre ďalší výskum.

Štatistické metódy klasifikácie sa ukázali ako nespoľahlivé. Práca O. Hujňáka [13] preto testovala metódy strojového učenia, vrátane algoritmov ARIMA, GMM, LOF, OCSVM, Isolation Forest a jednoduchého Multilayer Perceptron (MLP) klasifikátora. Najlepšie výsledky mal MLP s jedným vstupom (časový rozdiel medzi dvomi paketmi), ktorý bol presný pre individuálne trénované zariadenia. Kritickým nedostatkom však bola neschopnosť tohto jednoduchého MLP modelu generalizovať na iné typy zariadení – pri tréningu na viacerých zariadeniach súčasne kleslo F1 skóre často na 60% a menej. Neschopnosť adaptácie na rôzne vzory vysielania 'advertisement' paketov tak bráni univerzálnemu riešeniu.

Táto bakalárska práca rieši problém nedostatočnej generalizácie. Cieľom je navrhnúť, implementovať a vyhodnotiť architektúry neurónových sietí pre lepšiu generalizáciu detekcie BLE pripojení z časovania 'advertisement' paketov. Hypotézou je, že poskytnutie bohatšieho časového kontextu modelom – konkrétne sekvencií nedávnych časových rozdielov (delt) medzi paketmi – umožní modelom rozlíšiť normálny 'advertisement' rytmus od anomálií pripojenia, čím sa prekoná závislosť na nastavovaní manuálnych prahov. Práca opisuje hardvér vylepšenej monitorovacej sondy (3xESP32, Raspberry Pi, DTR synchronizácia), softvér pre zber dát, ako aj charakteristiky testovacích zariadení a metodológiu tvorby datasetu.

Pre overenie hypotézy sme v PyTorch Lightning systematicky skúmali rôzne konfigurácie neurónových sietí: základný Single-Input MLP, Multi-Input MLP spracovávajúce sekvencie N časových delt (N=3, 5, 7, 10, 15, 20) a 1D Konvolučnú Neurónovú Sieť (CNN). Pre Multi-Input MLP sa testovali rôzne hĺbky (1-3 skryté vrstvy), šírky (16-256 neurónov) a štandardné komponenty (ReLU, Batch Normalizácia, Dropout). Optimalizácia hyperparametrov (rýchlosť učenia, miera dropoutu, veľkosť dávky) prebehla nástrojom Optuna s F1 skóre ako primárnou metrikou pre nevyvážený dataset. Nevyváženosť tried sa riešila použitím WeightedRandomSampler počas tréningu a optimalizáciou Binary Cross-Entropy funkcie straty pomocou Adam optimalizátora.

Experimentálne overenie na zariadeniach vykazujúcich prerušovaný (intermittent) typ 'advertisement' správania potvrdilo, že využitie časového kontextu prostredníctvom sekvenčne pracujúcich neurónových sietí výrazne zlepšuje výkon a generalizáciu. Spomedzi testovaných architektúr dosiahla najlepšie celkové výsledky na validačnej sade 1D CNN s F1 skóre 96.3% (N=20), tesne nasledovaná Multi-Input MLP (N=20, 3 skryté vrstvy) s F1 skóre 95.7%. Tieto výsledky predstavujú výrazné zlepšenie oproti základným single-input modelom a predtým reportovaným generalizačným schopnostiam v [13]. Optimalizované

sekvenčné modely si udržali vysoké F1 skóre naprieč rôznymi typmi testovaných zariadení (Bentech, Danalock, Revogi, Mi Temp), čo potvrdzuje ich zlepšenú schopnosť adaptovať sa na rôznorodé charakteristiky 'advertisement' správania bez potreby individuálneho tréningu pre každé zariadenie.

Z vyvinutých modelov sme si vybrali Multi-Input MLP (N=15, 3 skryté vrstvy), ktorý mal F1 95.2%, čo je trochu menej ako modely s N=20, ale mal 25% menšiu veľkosť vstupu – čo sa odráža na oneskoreniu spracovania. Tento model bol následne nasadený na nami vytvorenú monitorovaciu sondu (Raspberry Pi 3B+ prepojenú s tromi ESP32) ako detektor pripojení v reálnom čase, čo potvrdilo ich praktickú životaschopnosť na cenovo dostupnom hardvéri. Pre vizualizáciu detekcií vzniklo aj grafické používateľské rozhranie webovej aplikácie, umožňujúce intuitívny prehľad o aktivite monitorovaných zariadení. Táto vizualizácia zvyšuje praktickú hodnotu systému pre koncového používateľa.

Cela táto práca bola experimentálne overená na vybraných zariadeniach v zaľudnenom prostredí s vysokým zarušením, pričom sme sa zamerali na detekciu pripojení a analýzu správania zariadení. Naše experimenty ukázali, že vybraná Multi-Input MLP architektúra (N=15, 3 skryté vrstvy) mala perfektné výsledky pri 3 zo 4 zariadení, pričom jedno zariadenie ukázalo správanie nepreskúmané vo vytvorenom datasete – ale napriek tomu, malo F1 nad 90%.

Záverom, táto práca úspešne demonštruje, že neurónové siete pracujúce so sekvenciami dát, najmä Multi-Input MLP a 1D CNN, dokážu efektívne riešiť kritický problém generalizácie pri pasívnej detekcii BLE pripojení. Vďaka zahrnutiu časového kontextu zo sekvencií 'advertisement' delt dosahujú naše modely robustný a adaptabilný výkon naprieč rôznymi zariadeniami. Tento výskum tak prispieva praktickejšou a lepšie generalizovateľnou metódou založenou na strojovom učení pre monitorovanie BLE, vhodnou pre zlepšenie prehľadu o bezpečnostnej situácii v heterogénnych IoT prostrediach.

Monitoring of Bluetooth Low Energy Devices

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mgr. Kamil Malinka, PhD. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Matej Olexa
May 12, 2025

Acknowledgements

I would like to thank my supervisor, Mgr. Kamil Malinka, PhD., for his guidance and feedback throughout this thesis. I also thank Ing. Ondřej Hujňák for his mentorship during the initial project practice phase. Thanks to my family and friends for their support during the completion of this thesis. Tools such as Gemini, DeepL and Claude were used for editorial, coding assistance and proofreading purposes during the writing process.

Contents

List of Figures

Chapter 1

Introduction

The growing world of Internet of Things (IoT) devices in smart homes and offices, especially those using Bluetooth Low Energy (BLE), has raised significant security concerns. These devices are often designed mainly for convenience, user-friendliness and low power consumption rather than security, making them attractive targets for attackers. As the number of connected devices continues to rise, so does the potential for security breaches [8, 21, 19].

BLE devices frequently exhibit security weaknesses due to inconsistent patching processes and vendor implementation problems. Many lack proper security features or receive infrequent updates, leaving vulnerabilities exposed – particularly concerning for security-critical devices like smart locks. Standard device logs are insufficient for security monitoring since sophisticated attackers can bypass them, creating a pressing need for independent passive monitoring solutions to detect unauthorized connection attempts [12].

Monitoring BLE devices presents unique challenges due to frequency hopping [33, 17] (detailed in Section 3.1.1) and the requirement for specialized hardware, which tends to be expensive. An alternative approach, proposed by Hujňák et al. [14], analyzes advertising packets (see Section 3.1.2) to detect connections. This method leverages the observation that many BLE devices stop sending advertising packets when a connection is established. By monitoring advertising channels, we can detect device connections by identifying characteristic gaps in the advertising pattern (see Section 3.1.3). This initial work provided a proof of concept using a single-sniffer setup and statistical analysis.

Following this discovery, research efforts shifted toward enhancing the monitoring probe and developing a more comprehensive dataset (as described in Chapter 6). The goal was to develop a multi-sniffer monitoring probe using multiple ESP32 microcontrollers (detailed in Section 5.2), each dedicated to monitoring one of the three primary BLE advertising channels, as outlined in Chapter 2. While this improved the data capture capabilities (Section 5.4), following analysis confirmed that simple statistical methods still struggled with precision and reliability for robust connection detection [13].

The research focus then evolved toward machine learning (ML) approaches. Hujňák et al. [13] evaluated several advanced techniques using the dataset from [15], including various anomaly detection algorithms and a basic supervised Multilayer Perceptron (MLP) (discussed further in Section 7.1.2). Their findings revealed that while a simple MLP delivered outstanding results when trained and evaluated on specific individual devices, it significantly underperformed in cross-device scenarios. Attempts to apply a single model across different device types resulted in poor generalization performance. This fundamental limitation – the inability of the simple MLP to generalize across different devices – is a challenge that needs to be addressed for effective passive BLE monitoring.

This thesis directly addresses this identified generalization challenge. The primary objective is to design, implement, and evaluate neural network architectures capable of accurately detecting BLE connection events based only on advertising packet timings, while generalizing effectively across a set of different devices. We hypothesize that by providing models with temporal context by sequences of recent advertising time deltas, rather than just single values as in Section 7.4, they can learn to differentiate between normal advertising patterns and anomalies, thereby finding connections.

The whole concept is illustrated in Figure 1.1. The figure shows the abstracted pipeline of our monitoring system, in which the monitoring probe captures advertising packets, then it sends them to a neural network model for processing, which then notifies the user about the detected connection.

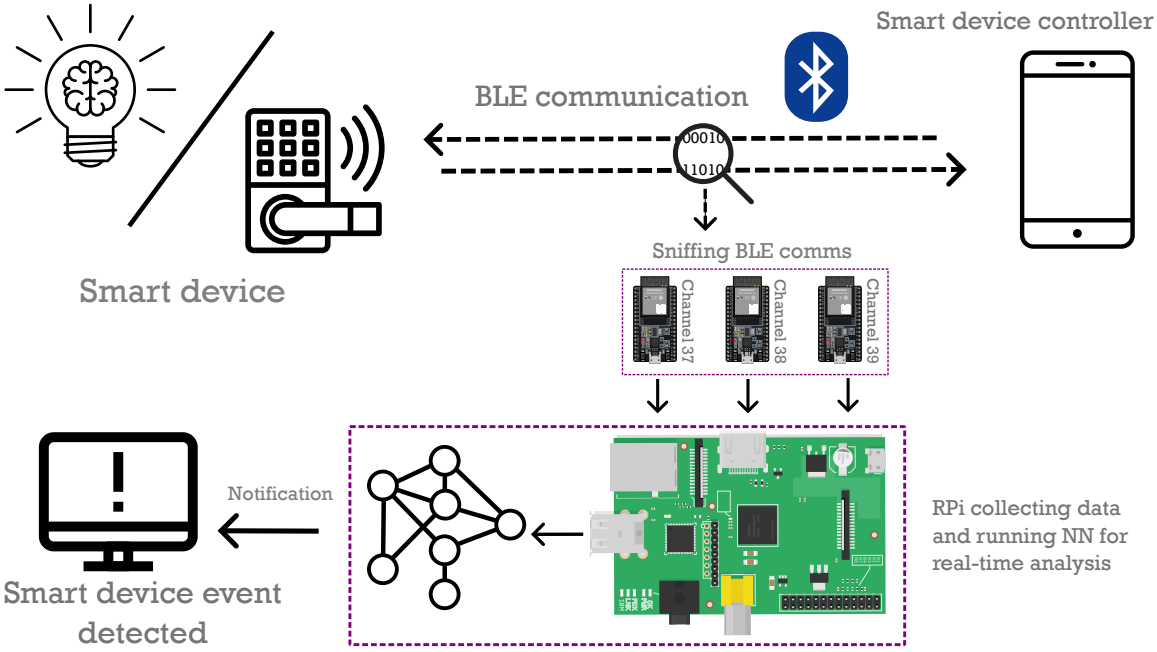

Figure 1.1: Abstracted use case diagram of our monitoring system.
Contains images from Jonathan Rutheiser under Creative Commons Attribution Share Alike 3.0 Unported license.

To test this hypothesis, we systematically explore and compare different neural network configurations implemented within the PyTorch Lightning framework [7]. This includes a baseline single-input MLP, multi-input MLPs processing varying sequence lengths (Section 7.6), and a 1D Convolutional Neural Network (CNN) architecture (Section 7.7) specifically designed for sequential data. We employ techniques like weighted sampling (Section 7.3.4) to address the inherent class imbalance in the dataset. Performance is evaluated using standard metrics (F1-score, precision, recall and accuracy [23]), with particular emphasis on assessing generalization improvements through consistent performance across different device types (Section 8.3).

Contribution to Foundational Work [15]:

- Development and validation assistance for the multi-ESP32 parallel monitoring probe that enhanced BLE advertising data acquisition
- Significant involvement in creating a comprehensive dataset for BLE connection detection research

- Contributions to prototyping, firmware/script debugging, and initial packet analysis during the dataset development phase

New Contributions of This Thesis:

- Development and evaluation of multi-input MLP and 1D CNN architectures specifically created to improve generalization in BLE connection detection using advertising time delta sequences
- Systematic investigation into how input sequence length and model hyperparameters affect performance and generalization of the proposed models (Chapter 7)
- Demonstration of significantly improved cross-device generalization compared to baseline models and previously reported results (Chapter 7.8)
- Implementation of a robust and practical machine learning-based approach for passive BLE monitoring suitable for real-world environments, including its deployment (Section 8.1)

Chapter 2

Project Context and Prior Stages

This chapter details the specific project context and prior research stages at the Faculty of Information Technology, Brno University of Technology (FIT BUT) that directly led to the work presented in this thesis. It outlines the progression of research on passive BLE monitoring via advertising analysis and clarifies the author's contributions at various stages.

2.1 Project Context and Prior Stages

This thesis was built as a direct continuation of the research conducted at FIT BUT, focusing on passive Bluetooth Low Energy (BLE) monitoring through advertising analysis. The initial proof-of-concept was developed by Hujňák et al. [14], who successfully demonstrated the ability to detect BLE connections by analyzing timing differences between advertising packets using a single-sniffer monitoring probe based around Raspberry Pi. This work created a foundation for further research in this area, including the development of a more robust monitoring probe and a comprehensive dataset. The author of this thesis was not involved in the initial work or the development of this monitoring setup.

As the next step, the focus shifted towards enhancing the monitoring capabilities and creating a dataset for BLE connection detection research. The goal was to improve the method of capturing advertising packets.

This involved developing a multi-sniffer monitoring probe using multiple ESP32 microcontrollers, each dedicated to monitoring one of the three primary BLE advertising channels. The resulting dataset was designed to simplify research on BLE connection detection using this improved capture method and work as a foundation for future research in this area. The author of this thesis was actively involved in the development of this work [15] and builds directly upon this contribution throughout this thesis. Specific contributions included involvement in the dataset creation process (6.4), participating in the prototyping phase (which involved identifying and resolving technical issues with various prototype iterations), contributing to ESP32 firmware debugging (though not its initial development), and developing the synchronized data collection Python script (5.2). The author also assisted with packet analysis and configuration of multiple test devices (6.2).

After the creation of the dataset, focus shifted towards improving the connection detection logic. As from initial work we knew, that simple statistical methods (7.1.1) are not sufficient as a general solution. So more complex statistical methods and machine learning techniques were explored. Initial investigations applying machine learning to this problem were presented by Hujňák et al. [13]. This study explored the potential use of various ML

algorithms (including Isolation Forest, Support Vector Machines, and Multilayer Perceptrons [5]) as mentioned in Section 7.1.2. It also demonstrated potential of ML while also highlighting areas for improvement, such as the model's ability to generalize across various device types with different advertising patterns. The author of this thesis was not involved in this work, but the results and findings from this study served as a foundation for the current thesis.

The primary objective of this thesis is to develop, evaluate and deploy more sophisticated machine learning models, specifically Multilayer Perceptrons (MLPs) and 1D Convolutional Neural Networks (CNNs), with an emphasis on improving detection accuracy and, crucially, achieving better generalization across the diverse range of BLE devices. While the previous work established the possible applicability of ML, this thesis dives deeper into designing neural network architectures (including single-input and multi-input MLPs, as well as 1D CNNs for sequential data processing, implemented in PyTorch Lightning framework for flexibility) explicitly aimed at overcoming potential generalization limitations observed or anticipated in earlier models.

Chapter 3

Theoretical Background

This chapter provides the theoretical background necessary for understanding the context of the work presented in this thesis. It covers the fundamental concepts of Bluetooth Low Energy (BLE) technology, machine learning principles, and the evaluation metrics used to assess the performance of the developed models.

3.1 Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a wireless communication technology designed for low power consumption and low cost, making it ideal for modern, battery-operated devices in various industries [2, 8].

With BLE, devices can communicate directly without the need for a central hub, enabling flexible and scalable architectures for Internet of Things (IoT) applications. These devices operate in one of two roles: central or peripheral. The central role is typically held by devices such as smartphones, tablets, or computers, which scan for and initiate connections with peripheral devices. Peripheral devices, on the other hand, are often low-power, battery-operated devices like wearables and sensors that advertise their presence and wait for a connection request from a central device. This client-server-like relationship allows BLE devices to engage in low-power efficient communication [3, 8, 21].

BLE is widely adopted in the Internet of Things, connecting devices like smartwatches, fitness trackers, and other wearable devices to smartphones for real-time data exchange. It also plays a crucial role in home automation and security systems, powering innovations such as smart locks, programmable thermostats, and adaptive lighting systems. Its energy efficiency, combined with an expanding ecosystem of compatible devices, has established BLE as a foundational technology for modern connected solutions [3, 8, 21].

3.1.1 Frequency hopping

BLE operates in the 2.4 GHz ISM band, which is divided into 40 channels of 2 MHz each. To avoid interference from other devices in the same frequency range, BLE uses a frequency hopping scheme. This allows devices to switch channels during communication, reducing the impact of interference from sources such as Wi-Fi (which uses the same 2.4GHz ISM band [33, 17]) or other Bluetooth devices.

Of the 40 channels, three (37, 38, and 39) are reserved for advertising, while the remaining 37 (0 to 36) are used for data transmission. The advertising channels are carefully selected to avoid overlapping with common Wi-Fi frequencies. By jumping across the data

channels, BLE improves its resistance to interference and jamming, while also enhancing security against eavesdropping. This makes BLE a reliable choice for applications in crowded RF environments [3, 4, 27].

3.1.2 Advertising

BLE devices are detected by other devices through advertising. Advertising is a process in which a BLE device broadcasts packets containing information about itself (such as device name, services offered, etc.) to other devices. Advertising devices can be in two states: advertising state and non-advertising state. In the advertising state, the device transmits packets at regular intervals to one of the three primary advertising channels, which may include all three channels or just a subset. Advertising packets are transmitted at an interval that might vary from 20 ms to 10.24 seconds according to low duty cycle specifications. After each advertising interval, a pseudo-random delay of 0 to 10 ms is added to minimize network collisions [3, 14].

3.1.3 Persistent and Intermittent Connections

Based on the advertising our advertiser device provides, we can distinct between 2 different types of connection [3].

Intermittent

This is a connection pattern where devices connect only when data need to be exchanged and terminate the connection once the exchange is complete. This pattern exposes one of the limitations of Bluetooth in IoT devices, which can maintain only a single connection concurrently. However, its advantage lies in its power efficiency and simplified radio implementation, making it suitable for battery-operated devices with intermittent communication needs.

Devices with intermittent advertising patterns will typically stop broadcasting advertising packets while a connection is active, creating a detectible gap in the advertising pattern, as shown in Figure 3.1. Once the connection terminates, the device resumes normal advertising. This distinctive behaviour forms the basis of our indirect connection detection method [3, 14].

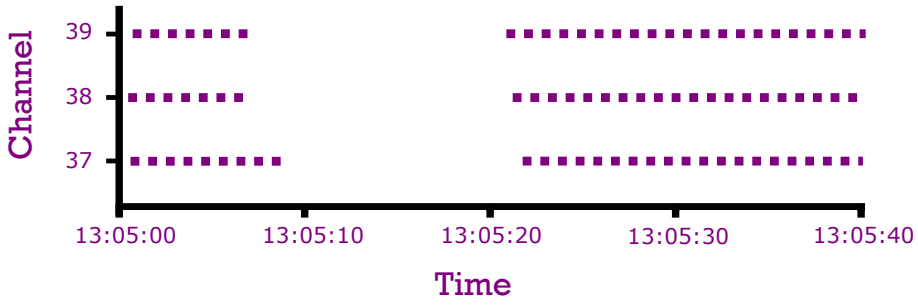

Figure 3.1: Abstracted advertising pattern of a device with intermittent advertising.

Persistent

During this connection, the devices remain continuously connected allowing real-time data exchange and interaction between the devices. This is a approach allows a connection of devices concurrently, however it requires more resources, as the device has to manage connection and advertising at the same time [3, 14].

3.1.4 Standard Detection Methods based on Advertising

Continuous monitoring of the BLE network is challenging due to Bluetooth features, such as frequency hopping during communication [2]. This thesis adapts the method proposed by Hujňák et al. (2023) [14, 12], which works with the fact that BLE devices broadcast advertising packets at regular intervals on advertising channels when idle. Once a connection is established, devices typically switch to general-purpose channels. Monitoring these advertising channels allows detection of connection states: If the advertising packets of a static device pause temporarily, it likely indicates an active connection.

This approach works with standard Bluetooth chips and requires no hardware modifications, which is a significant advantage over other methods. We can determine the average time between advertising packets and identify connections when there is a substantial gap between packets. This simplified statistical method serves as confirmation of connection states.

3.2 Machine Learning Fundamentals

Machine learning (ML) is a subset of artificial intelligence that enables systems to learn patterns from data and make predictions or decisions without without being explicitelly programmed [16].

3.2.1 Types of Machine Learning

Machine learning can be broadly categorized into three main types [16, 9]:

Supervised Learning

In supervised learning, the model is trained on a labeled dataset, which means that each training example is paired with an output label. The goal is to learn a mapping from inputs to outputs based on the provided examples. Common algorithms include linear regression, logistic regression, support vector machines, and neural networks [16, 9].

Unsupervised Learning

Unsupervised learning involves training a model on data without labeled responses. The goal is to identify patterns or structures within the data. Common techniques include clustering and dimensionality reduction [16, 9].

Reinforcement Learning

Reinforcement learning is a type of machine learning where an agent learns by taking actions in an environment to earn rewards, aiming to maximize its total rewards over time [16, 9].

For the purposes of this work, supervised learning is the most relevant type of machine learning, as the dataset contains labeled examples of BLE connection states.

3.2.2 Multi-layer perceptron

A Multilayer Perceptron (MLP) is a feedforward neural network made up of multiple layers, including an input layer, one or more hidden layers, and an output layer. It is commonly used in deep learning for tasks like classification, regression, and pattern recognition [24, 9].

Key Components

- **Layers**:
  - **Input Layer**: Receives the input.
  - **Hidden Layers**: Intermediate layers that transform the input data using weights and activation functions. An MLP can have one or more hidden layers [9].
  - **Output Layer**: Produces the final prediction or output.
- **Loss Function**:
  - The loss function tracks the error between a model's prediction and target value [24].
- **Optimization**:
  - Training an MLP involves minimizing the loss function using optimization algorithms like gradient descent or its variants (e.g., stochastic gradient descent (SGD), Adam) [28, 9].
  - Backpropagation is used to compute the gradients of the loss function [28].
- **Weights and Biases**:
  - **Weights**: Parameters that the network learns (varies) during training. They determine the strength of the connection between neurons in layers [24].
  - **Biases**: Biases are additional parameters that allow the network to shift the activation function, providing more flexibility in learning [24].
- **Activation Functions**: Activation functions calculate the output of a neuron based on its weights and biases. By introducing non-linearity, they allow the network to learn complex data patterns [9]. Common activation functions include:
  - **ReLU (Rectified Linear Unit)**: $f(x) = \max(0, x)$
    This simple function outputs 0 for all negative inputs, but returns the input value for all positive values. It is computationally efficient, helps with solving the vanishing gradient problem and introduces non-linearity without affecting the receptive fields of the convolution layer [24, 9].
  - **Softmax**: $f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$
    Used mainly in the output layer for multi-class classification, Softmax converts a vector of real values into a probability distribution. The function ensures all outputs sum to 1, making it ideal for representing categorical distributions [9].

- **Sigmoid**: $f(x) = \frac{1}{1+e^{-x}}$
  This function maps any input to a value between 0 and 1, making it useful for binary classification tasks. However, it can suffer from the vanishing gradient problem, especially for deep networks [24].

- **Dropout**: A regularization technique used to prevent overfitting by randomly setting a fraction of the neurons to zero during training. This forces the network to learn more robust features and reduces reliance on specific neurons [31].

- **Batch Normalization**: A technique that normalizes the inputs to each layer, improving training speed and stability. It helps mitigate issues related to internal covariate shift and allows for higher learning rates [18].

3.2.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a specialized type of neural network designed for processing structured grid data, such as images. They are particularly effective for tasks like image classification, object detection, and image segmentation. CNNs leverage the spatial structure of data by applying convolutional operations to extract local features, making them well-suited for visual recognition tasks [24, 22].

Their typical use case is in image processing, but they can also be used for other types of data, such as time series or audio signals using 1D convolutions. CNNs consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers [6, 9, 20, 32].

Key Components of CNNs

- **Convolutional Layers**: These layers apply convolutional filters to the input data, extracting local features. The filters slide over the input data, performing element-wise multiplication and summing the results to produce feature maps [22, 32].

- **1D Convolutions**: While 2D convolutions process data with spatial dimensions (like images), 1D convolutions are specifically designed for sequential data like time series or text. In 1D CNNs, the convolution operation slides along a single dimension (e.g., time), making them particularly suitable for processing BLE advertising deltas as a sequence [20, 9].

- **Kernel Size**: This parameter defines the width of the convolutional filter. For 1D convolutions, a kernel of size k considers k consecutive elements in the sequence. Smaller kernels (e.g., 3 or 5) detect local patterns, while larger kernels capture broader temporal dependencies [32, 25].

- **Padding**: Padding adds elements (typically zeros) to the beginning and end of the input sequence. This preserves the spatial dimensions after convolution and ensures that elements at the edges receive equal consideration. Common padding strategies include „valid“ (no padding) and „same“ (padding to maintain input dimensions) [32, 9].

- **Stride**: The stride determines how the filter shifts across the input. A stride of 1 moves the filter one element at a time, while larger strides skip elements, reducing the output size. For 1D time series, smaller strides typically preserve more temporal information [32, 25].

- **Weight Sharing**: A key feature of CNNs is parameter efficiency through weight sharing. The same filter weights are applied across the entire input sequence, allowing the network to detect the same pattern regardless of where it appears in the sequence. This significantly reduces the number of parameters compared to fully connected networks [22, 32].
- **Pooling Layers**: Pooling layers reduce the spatial dimensions of feature maps, retaining only the most important information. Common pooling operations include max pooling and average pooling [32, 9].
- **Fully Connected Layers**: These layers connect every neuron in one layer to every neuron in the next layer, similar to traditional neural networks. They are typically used at the end of a CNN for classification tasks [25, 32].

3.3 Evaluation Metrics

To evaluate the performance of the machine learning model, several metrics are used to measure its accuracy and efficiency. These metrics provide insights into the model's performance and help identify areas for improvement [23, 10, 29].

3.3.1 Accuracy

Accuracy is a metric that measures the proportion of correct predictions made by the model relative to the total number of predictions.

It is calculated as follows:

$$\text{Accuracy} = \frac{\text{correct predictions}}{\text{total predictions}}$$

Where:

- **Correct Predictions**: The number of predictions that match the actual values.
- **Total Predictions**: The total number of predictions made by the model.

A high accuracy score indicates that the model is making correct predictions; therefore, an ideal model would have an accuracy score of 1.0 (or 100%). However, when dealing with imbalanced datasets, for example, when the number of positive samples is 1% of total samples, a model that predicts all samples as negative would have an accuracy of 99%, even when it does not make any correct positive predictions [10, 23].

3.3.2 Recall

Recall measures the proportion of actual positive samples that were correctly identified by the model. It quantifies the model's sensitivity in finding all positive cases within the dataset [23, 10].

It is calculated as follows:

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Where:

- **True Positives**: The number of correctly predicted positive results.
- **False Negatives**: The number of incorrectly predicted negative results.

An ideal model would have a recall score of 1.0 (or 100%), which means that all actual positive samples were correctly identified.

3.3.3 Precision

Precision measures the proportion of predicted positive samples that were correctly identified by the model [23, 10].
It is calculated as follows:

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

Where:

- **True Positives**: The number of correctly predicted positive results.
- **False Positives**: The number of incorrectly predicted positive results.

3.3.4 F1 Score

The F1 score is the harmonic mean of precision and recall, providing a balance between the two metrics. It is a useful metric for evaluating the overall performance of a model, especially when dealing with an unbalanced dataset [23, 29].

It is calculated as follows:

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

The F1 score ranges from 0 to 1, with 1 indicating perfect precision and recall and 0 indicating poor performance [29, 23].

Chapter 4

Preparation

The practical part of this thesis is divided into four main sections. This chapter explains the overall structure and approach of the practical work done in this thesis. The primary objectives, along with their specific success criteria, were to:

1. **Collaborate on enhancing BLE monitoring capabilities:** Building upon existing research at FIT BUT [14], the aim was to improve detection reliability.
   - *Criterion:* The enhanced monitoring probe
2. **Create a comprehensive dataset:** Through collaborative research [15], develop a rich dataset of BLE advertising packets.
   - *Criterion:* The dataset must comprehensively include all chosen test devices (as detailed in Section 6.2) and cover the full range of their defined operational actions.
3. **Design and evaluate machine learning models:** Develop models for connection detection with specific performance targets, building upon insights from prior work [13].
   - *Clarification:* While an initial consideration was to develop separate neural networks for connection detection and for classifying a device's advertising pattern (intermittent vs. persistent), it was reasoned that if a primary neural network could reliably detect connection events (i.e., significant advertising gaps), this implies the device shows an intermittent advertising pattern. Devices with persistent patterns would not show such gaps during connections (Section 3.1.2). This means, that creating a separate classifier for advertising patterns is unnecessary. This allowed us to focus the machine learning efforts specifically on robust connection detection for devices where the advertising-gap principle applies.
   - *Criterion:* The primary machine learning model must achieve a detection performance (F1-score) of at least 80% across the diverse set of devices in the dataset, demonstrating effective generalization. (Further detailed in Section 4.1).
4. **Deploy an effective monitoring solution:** Implement the developed model onto the monitoring probe for practical application.

- *Criterion:* The selected model must be successfully deployed onto the existing monitoring probe hardware and operate effectively in real-time. (Further detailed in Section 4.1).

The practical implementation follows a systematic progression through these objectives. First, in Chapter 5, we detail the monitoring setups developed for capturing BLE advertising packets, including the original Raspberry Pi approach (Section 5.1) and our enhanced multi-ESP32 configuration (Section 5.2). The following chapter (Chapter 6) then describes the creation and analysis of our dataset, including the selection of diverse test devices, data collection methodology, and observed advertising patterns.

With this foundation established, Chapter 7 presents our machine learning approaches, beginning with baseline models and progressing to more sophisticated architectures, specifically designed to address the generalization challenges identified in prior work. Finally, Chapter 8 covers the real-world deployment of the selected model on embedded hardware, including the development of an intuitive monitoring interface and the results of real-time validation experiments.

4.1 Performance Metrics and Success Criteria for ML Model and Deployment

To evaluate the effectiveness of our machine learning approach and its deployment compared to existing solutions, we established the following quantitative success criteria for the developed model:

- **Detection Performance (F1-Score)**: The developed machine learning model must achieve an F1-score of at least 80% when evaluated across different devices. This target aims to significantly improve upon baseline statistical methods that typically achieve F1-scores below 50% (as shown in Table 7.1).
- **Precision Focus**: While maintaining high overall performance, the model must prioritize precision to minimize false positives, which are particularly problematic in security monitoring applications. A precision of at least 85% is targeted.
- **Generalization**: Unlike previous approaches that performed well only on specific devices upon which they were trained, our solution must maintain consistent and effective performance across multiple device types without requiring device-specific retraining.
- **Real-time Capability**: The developed model must be computationally efficient enough to run in real-time on resource-constrained platform – Raspberry Pi 3B+ in our case.

These metrics were selected to ensure that the resulting solution would provide genuine practical improvements over existing techniques, making it suitable for real-world security monitoring applications.

Throughout the subsequent chapters, we emphasize the practical challenges encountered and the incremental improvements achieved in reliability, accuracy, and generalization capabilities of BLE connection detection.

Chapter 5

Monitoring of BLE Devices

This chapter details the hardware and software components of the monitoring systems used for data acquisition. It begins by describing the original single-controller setup (5.1) and then focuses on the enhanced multi-ESP32 monitoring probe (5.2) developed to improve data capture reliability.

5.1 Original Single-Controller Setup (RPi-Based)

The initial proof-of-concept for detecting BLE connections via advertising analysis by Hujňák et al. [14] utilized a standard Raspberry Pi 3B+ with its integrated Bluetooth controller. This setup involved the RPi scanning advertising channels sequentially, a common approach for BLE sniffing with general-purpose hardware [11]. While effective for demonstrating the principle, this single-controller approach faces limitations in capturing all advertising packets due to the time taken to switch between channels (37, 38, and 39) while devices continue to advertise. This can lead to missed packets, potentially affecting the accuracy of timing-based analysis. Data captured was typically processed directly on the Raspberry Pi.

5.2 Improved Setup: Triple ESP32 Monitoring Probe

To overcome the limitations of the single-controller setup (described in Section 5.1) and enhance data capture reliability, an improved multi-sniffer monitoring probe was developed using three ESP32 microcontrollers, as also detailed in the collaborative work [15]. Each ESP32 is dedicated to continuously monitoring one of the three primary BLE advertising channels (37, 38, and 39), ensuring full coverage and minimizing packet loss due to channel hopping (as metioned in Section 3.1.1). This approach was designed to be a cost-effective and easily deployable solution for comprehensive advertising data collection.

5.2.1 Hardware Architecture

The probe's hardware architecture, illustrated conceptually in Figure 5.1 and physically in Figure 5.2, consists of three ESP32 microcontrollers. Each ESP32 acts as a dedicated sniffer for one advertising channel. These ESP32s are connected via USB to a central processing unit – a Raspberry Pi 3B+ in our case, which collects and stores the data. The Raspberry Pi also controls the synchronization of the ESP32 units.

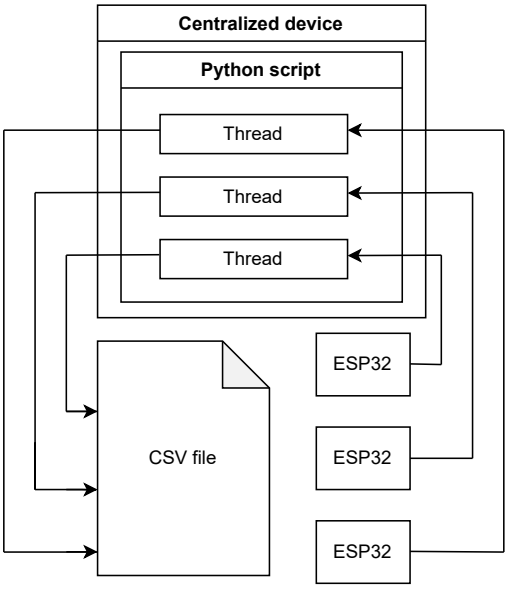

Figure 5.1: Abstract concept of the parallel monitoring probe

Figure 5.2: Physical implementation of the ESP32-based monitoring probe

5.2.2 Custom ESP32 Firmware

The core functionality of dedicating each ESP32 to a single channel was enabled by custom firmware. This firmware utilized low-level API access, provided through collaboration

with Espressif Systems, to lock each ESP32 device onto its assigned advertising channel, something not available in the standard ESP-IDF. This approach enabled continuous observation of a specific channel without the typical channel-hopping behaviour that leads to missed packets in single-controller setups.

When the author of this thesis joined the research team at FIT BUT, the custom firmware for the ESP32s, enabling single-channel locking, had been already created. The author's contributions to the monitoring probe focused on integrating the ESP32s with the Raspberry Pi, ensuring correct data transmission and synchronization, and developing the Python script for data collection (described in Section 5.2.3).

5.2.3 Data Transmission and Synchronization

Each ESP32, after capturing and processing advertising packets, transmits the relevant data to the centralized device via a serial (UART) connection. The transmitted data includes timestamps (taken from the ESP32's internal tick counter), device names (when available from advertising packets), advertising types, Bluetooth address information (address and type), RSSI values, and the specific channel number it was monitoring.

To ensure accurate timing across all channels, we implemented a synchronization system. This mechanism uses the Data Terminal Ready (DTR) signal to reset all ESP32 units simultaneously. This hardware-level reset creates a common starting point for all ESP32 internal timers at the beginning of each data collection session.

5.3 Concurrent Data Collection

To create a comparison between the original RPi-based setup and the new multi-ESP32 probe, data collection (detailed in Chapter 6) was performed concurrently using both systems. The RPi-based setup ran the original Python script from Hujňák et al. [14], while the new multi-ESP32 probe utilized the custom firmware and Python script developed for this thesis. This dual approach then allowed for a analysis of the performance and reliability of both systems in capturing BLE advertising packets.

5.4 Evaluation of Monitoring Probes

After collecting the dataset described in Chapter 6 (using both setups at the same time), a comparison of the two data collection methods was performed. This evaluated the performance of the new multi-ESP32 monitoring probe (Section 5.2) against the original method using the Raspberry Pi 3B+ internal controller (Section 5.1). The analysis was focused to determine, if the new probe could provide a significant improvement in the reliability of BLE advertising packet capture, particularly in terms of missed packets and the accuracy of timing measurements.

The results of this analysis (also discussed in the paper [15] this work was a part of) and further detailed in Ondřej Hujňák's dissertation [12], found that the measured advertising intervals (calculated using the median delta of advertising packet timings) for the target devices were almost identical across both collection methods, with differences typically less than 2%. This suggests that a singulas BLE controller is generally sufficient for capturing advertising packets.

Additionally, although the multi-ESP32 probe continuously monitored all three primary advertising channels, it only resulted in a slight reduction in missed packets (approximately

5% fewer missed packets on average, determined by clustering deltas around expected intervals) compared to the standard RPi controller, which cycled through the channels. This indicates that while the parallel probe offers some improvement in capture completeness, the nature of BLE advertising and environmental factors still lead to some packet loss.

These findings suggest that the original RPi-based setup is generally effective for BLE advertising packet capture, but the multi-ESP32 probe can provide additional reliability in specific scenarios, particularly in environments with high interference or when monitoring multiple devices simultaneously.

Chapter 6

Dataset Creation and Analysis

This chapter details the process of creating the comprehensive BLE advertising dataset used in this thesis, including the motivation, data storage methods, collection protocols, devices used, and initial analysis comparing different data capture techniques.

6.1 Dataset Motivation and Acquisition Context

The primary objective of the paper that this work builds upon [15] was to create an expanded dataset that significantly improved upon previous work [14], which served as an initial proof of concept with limited scope. The earlier work by Hujňák et al. [14] tested only 5 devices and lacked standardized methodology – some actions were performed only twice while others had more repetitions, and not all possible device actions were systematically explored.

A key limitation of the previous dataset was the absence of distinction between „clean actions“ (where the controlling application is started specifically for the action and then closed) versus normal actions (where the application remains running). This distinction is important as it can capture different connection patterns, particularly initial connection establishment overhead.

We wanted to build a complete dataset using many different kinds of devices – locks, switches, sensors, and lights. We made sure to follow the same testing steps for every device. Each action was tested exactly five times in both clean (RF shielded chamber) and normal (real office enviroment) settings. This careful approach makes our results more reliable and shows how these devices work in real-world situations.

One of the main motivations for creating this dataset was the absence of similar publicly available resources for BLE connection detection research. The dataset was designed to work as a foundation for more possible research in this technological direction.

6.2 Devices Used

The main focus of this work was security monitoring, so we selected a variety of everyday IoT devices, including door locks, padlocks, light bulbs, power plugs, and sensors. These devices were grouped into distinct categories, including locks (comprising two door locks and two padlocks), Lighting, Actuators, and Sensors. Each category addresses specific security requirements and operational characteristics.
The devices were then also classified by their power source. Some, like Philips Hue and

Revogi bulbs, plug into the mains, while most others run on batteries. This affects how often they send signals because battery-powered ones have to balance how much they use [15].

Table 6.1: Devices contained in the dataset

Device class	Device
Lock	**Bentech** FP3
	Danalock V3-BTZE
	igloohome Padlock Lite
	Nuki Smart Lock 3.0
Lighting	**Philips** Hue **white**
	Revogi Bluetooth LED bulb
Actuator	**Philips** Hue Smart **plug**
Sensor	**Mi Temp**erature and Humidity Monitor 2
	BeeWi Motion Sensor

6.2.1 Devices Excluded from Dataset

During the process of dataset creation, several devices were excluded for various technical and practical reasons. These exclusions are documented below to provide a comprehensive understanding of the dataset scope and limitations:

Compatibility Issues:

- **Locksmart**: Excluded due to application incompatibility with modern Android devices. Even after downgrading the system version, connection remained impossible. Later research revealed the manufacturer had ceased operations.
- **YEELIGHT Multicolor M2**: Despite being marketed as BLE compatible, this device required connection through a Google Nest hub rather than a connection directly through smartphone. Even after creating a proper Google Nest profile, pairing attempts were unsuccessful.
- **Vocolinc VS1 sensor**: This device exclusively supported Apple ecosystem devices, which were unavailable to the research team.

Redundancy:

- **BeeWi door sensor**: Although physically a different product, this sensor displayed advertising patterns identical to the BeeWi motion sensor already included in the dataset. Since both sensors use the same communication protocol and advertise in the same manner, including the door sensor would have been redundant and provided no additional value to the analysis.

These exclusions show some of the real-world challenges in building complete BLE device datasets, such as different systems not working together, device makers setting special requirements, and apps that don't work well.

6.3 Data Storage and Centralized Processing Script

The Raspberry Pi 3B+ serves as the central device for aggregating data from the three ESP32s (as described in Chapter 5) and storing it persistently.

6.3.1 Python Implementation for Data Collection

For data collection and aggregation on the central Raspberry Pi, the author created a Python script. This script was designed to handle concurrent data streams from the three ESP32s. To achieve robust parallel processing without blocking the main data logging operations, daemon threads were implemented within the Python script. As previously shown in Figure 5.1 (conceptually showing data flow to the RPi), each ESP32's serial connection is serviced by a dedicated thread on the Raspberry Pi.

6.3.2 CSV Data Format

The structured data received from the ESP32s (timestamp, device name, advertising type, address, RSSI, channel) is appended by the Python script to a Comma-Separated Values (CSV) file. This format was chosen for its simplicity and ease of use with various data analysis tools and libraries. Each row in the CSV file corresponds to a single advertising packet report. The specific columns are:

- **Timestamp**: The runtime of the script at the moment the data block is received from an ESP32.
- **DeviceName**: The name of the advertising device (if available in the packet).
- **Advertising Type**: The type of advertising packet.
- **Address Type**: Whether the Bluetooth address is public or random.
- **Address**: The unique MAC address of the advertising device.
- **RSSI**: The Received Signal Strength Indicator.
- **Channel**: The advertising channel (37, 38, or 39) on which the packet was captured.

6.3.3 Ensuring Chronological Order and Data Integrity

The multi-threaded Python script ensures that data packets are read from serial buffers promptly. Dedicated threads minimize processing delays between channels. Timestamps from the DTR-synchronized ESP32s are used for ordering. The script buffers data locally if needed and logs records to maintain capture order and consistency.

6.4 Data Collection Protocol and Environments

The methodology for creating the dataset involved a dual capture approach, where data collection was simultaneously performed by the standard Raspberry Pi 3B+ BLE controller (as used in [14] and described in Section 5.1) and the new triple ESP32-based monitoring probe (Section 5.2), which was introduced in [15]. This setup allowed for direct comparison and validation.

Measurements were conducted in two distinct environments:

- **RF-Shielded Chamber:** Lab Q110 at FIT BUT provided a controlled environment isolated from external signal interference, allowing for clean data collection.
- **Office Setting:** The Security@FIT research office (A222 at FIT BUT) represented real-world conditions with environmental noise and interference from other wireless devices. Office measurements were performed during normal working days.

This dual-environment approach enabled analysis of both ideal and practical device performance. The physical arrangement of equipment was carefully controlled throughout data collection. The measured device, control device (an Android tablet), and monitoring probe were positioned in a triangular formation with 50 centimeters between each component, as shown in Figure 6.1. This standardized layout ensured consistent signal strength and minimized variability in the capture conditions.

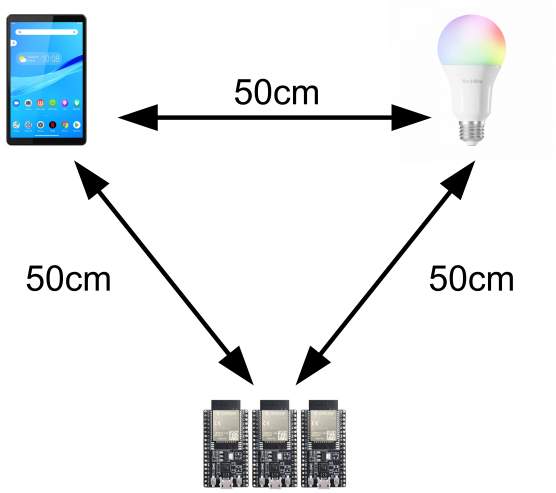

Figure 6.1: Conceptual diagram of the measurement setup

The monitoring probe (Chapter 5) was managed from a laptop connected via Ethernet, with its WiFi disabled to maintain data integrity. Each measurement session lasted one minute. After an initial 20 seconds of baseline data collection, a specific action (detailed per device in Section 6.2) was manually performed by an operator. Each session focused on a single action. Passive measurements (no actions, control app not running) were also taken. Each measurement type (active and passive) was repeated five times to minimize errors and ensure dataset reliability [15]. Measurements were repeated if initial issues were identified to ensure data cleanliness.

A representative example of the actual measurement environment is shown in Figure 6.2, which displays the real-world implementation of the triangular setup described above.

6.4.1 Dataset Analysis Methodology

To characterize the captured advertising data and compare collection methods, we employed the analysis methodology from original work [14]. This involved processing raw advertising data by calculating time deltas between consecutive packets from each target device.

These time differences helped us identify key device behaviours detailed later (Section 6.4.2). We found each device's typical advertising timing using the middle value of all

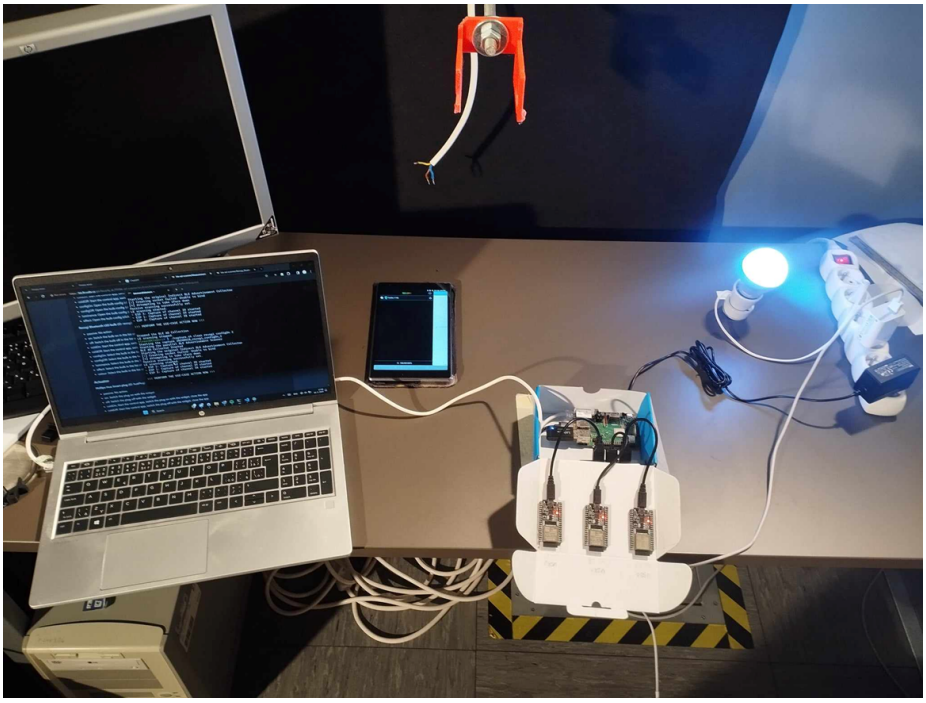

Figure 6.2: Real-world depiction of data collection setup during measurement sessions

measured deltas. We determined if devices stop advertising during connections (intermittent) or keep advertising (persistent) by looking at how these time values were spread out. We checked how many packets were missed by looking at groups of values near multiples of the normal timing. Connection duration was estimated using the longest gap between packets, and we used the basic spread of values as a way to check data quality [12, 15].

6.4.2 Detailed Device Characteristics

Below are detailed characteristics, observed behaviours, and specific actions performed for each device during dataset collection. This information is crucial for understanding the nuances in the captured data and the challenges faced during analysis. It should be noted that this detailed analysis was conducted only partially by the author of this thesis, as the primary device characterization work was established in [13].

BeeWi Motion Sensor:

- **Dataset ID:** `beeWiMotion`
- **Model:** BSMOT-EUR rev. AW11
- **BDADDR:** `04:A3:16:67:20:A9`
- **Control App:** `OtioHome`
- **Actions Recorded:**
  - *Data:* App running, motion detection triggered.
  - *ColdData:* App started, motion triggered, app closed.

  - *ExpandedData:* App running, motion sensor widget selected, motion triggered.

- **Advertising Behaviour:** Intermittent pattern with a dual advertising interval observed.
  - Normal interval: Approximately 1060 ms.
  - Post-connection phase: Lasts about 8 seconds with a faster advertising interval of approximately 200 ms.

- **Dataset Notes:**
  - Due to app start-up time issues in the office environment, the 'ColdData' action effectively became the 'ExpandedData' action.
  - Some samples were potentially affected by environmental factors or hardware issues: accidental motion triggers during measurement (office/expandedData samples 1, 4, 5) and a possible brownout causing timestamp skew on one capture channel (specific samples not identified here, but noted as a potential issue).

Bentech FP3 Lock:

- **Dataset ID:** `bentech`
- **Model:** FP3
- **BDADDR:** `DC:23:4F:A9:9A:46`
- **Control App:** `Tuya` (Note: This application tends to maintain an open connection when active, suppressing advertisements during 'Lock' and 'Unlock' actions).
- **Actions Recorded:**
  - *Lock:* App running, lock manually engaged.
  - *Unlock:* App running, unlocked via app command.
  - *ColdLock:* App started, lock manually engaged, app closed.
  - *ColdUnlock:* App started, unlocked via app command, app closed.
- **Advertising Behaviour:** Intermittent pattern.
  - Normal interval: Approximately 1003 ms.

Danalock V3-BTZE:

- **Dataset ID:** `danalock`
- **Model:** Danalock V3-BTZE
- **BDADDR:** `E8:15:D4:F5:BA:BA`
- **Control App:** `Danalock` (Multiple app versions exist: 'Danalock Classic', 'Danalock' (newer), and unofficial ones. The specific version used for control might influence behaviour, though not specified here).

- **Actions Recorded:**
  - *Lock:* App running, locked via app command.
  - *Unlock:* App running, unlocked via app command.
  - *ColdLock:* App started, locked via app command, app closed.
  - *ColdUnlock:* App started, unlocked via app command, app closed.
- **Advertising Behaviour:** Intermittent pattern.
  - Normal interval: Approximately 500 ms.
- **Dataset Notes:**
  - Accessing the activity log via the app does not trigger Bluetooth communication as logs are not stored locally on this model.

Philips Hue White Bulb:

- **Dataset ID:** `hueBulb`
- **Model:** 9290018216
- **BDADDR:** `FF:CB:D1:DC:DF:EA`
- **Control App:** `Philips Hue`
- **Actions Recorded:**
  - *On/Off:* App running, bulb toggled via main widget.
  - *ColdOn/ColdOff:* App started, bulb toggled via widget, app closed.
  - *ConfigOn/ConfigOff:* App running, entered bulb config screen, toggled state, returned to overview.
  - *Luminance:* App running, changed brightness in config.
  - *Effect:* App running, entered config, selected an effect, returned to overview.
- **Advertising Behaviour:** Persistent pattern.
  - Normal interval: Approximately 856 ms.

Philips Hue Smart Plug:

- **Dataset ID:** `huePlug`
- **Model:** 9290018216 (Note: Same model number listed as bulb, confirm if correct or typo in source data)
- **BDADDR:** `D7:3F:2B:21:A4:63`
- **Control App:** `Philips Hue`
- **Actions Recorded:**
  - *On/Off:* App running, plug toggled via main widget.

  - *ColdOn/ColdOff:* App started, plug toggled via widget, app closed.

- **Advertising Behaviour:** Persistent pattern.
  - Normal interval: Approximately 327 ms.

igloohome Padlock Lite:

- **Dataset ID:** `igloohome`
- **Model:** SP3B
- **BDADDR:** `CB:94:54:ED:5D:28`
- **Control App:** `igloohome`
- **Actions Recorded:**
  - *Lock:* App running, locked via app command.
  - *Unlock:* App running, unlocked via app command.
  - *ColdLock:* App started, locked via app command, app closed.
  - *ColdUnlock:* App started, unlocked via app command, app closed.
  - *Log:* App running, navigated to lock details > Logs tab, clicked „Get recent logs“.
- **Advertising Behaviour:** Persistent pattern.
  - Normal interval: Approximately 414 ms.

Mi Temperature and Humidity Monitor 2:

- **Dataset ID:** `miTemp`
- **Model:** LYWSD03MMC
- **BDADDR:** `A4:C1:38:E0:67:AE`
- **Control App:** `Mi Home`
- **Actions Recorded:**
  - *Data:* App running, selected sensor widget, waited for data/history load, returned to app home.
  - *ColdData:* App started, selected widget, waited for data/history load, closed app.
- **Advertising Behaviour:** Intermittent pattern.
  - Normal interval: Approximately 2191 ms (very long).

Nuki Smart Lock 3.0:

- **Dataset ID:** `nuki`
- **Model:** Nuki Smart Lock 3.0
- **Type:** 010.318
- **BDADDR:** `54:D2:72:64:1B:F4`
- **Control App:** `Nuki Smart Lock`
- **Actions Recorded:**
  - *Lock:* App running, locked via app command.
  - *Unlock:* App running, unlocked via app command.
  - *ColdLock:* App started, locked via app command, app closed.
  - *ColdUnlock:* App started, unlocked via app command, app closed.
  - *Log:* App running, navigated to lock settings to display activity log.
- **Advertising Behaviour:** Persistent pattern with a dual advertising interval.
  - Normal interval: Approximately 1010 ms.
  - Post-connection phase: Lasts about 4 seconds with a much faster advertising interval of approximately 44 ms. (Note: The 'Log' action does not trigger this faster advertising phase).

Revogi Bluetooth LED Bulb:

- **Dataset ID:** `revogi`
- **Model:** LTB012
- **BDADDR:** `E0:E5:CF:15:17:48`
- **Control App:** `DeLite` (Note: This application tends to maintain an open connection when active, suppressing advertisements during non-'Cold' actions).
- **Actions Recorded:**
  - *On/Off:* App running, bulb toggled from device list.
  - *ColdOn/ColdOff:* App started, bulb toggled, app closed.
  - *ConfigOn/ConfigOff:* App running, entered bulb config, toggled state, returned to list.
  - *Luminance:* App running, entered config, changed brightness, returned to list.
  - *Effect:* App running, entered config, selected effect, returned to list.
  - *Colour:* App running, entered config, changed hue, returned to list.
- **Advertising Behaviour:** Intermittent pattern.
  - Normal interval: Approximately 55 ms (very fast).

- **Dataset Notes:**
  - During passive captures in the office environment, the parallel collector detected exceptionally large advertising delays in several samples (Office/passive 1, 3, 4, 5 on various channels) that were not observed by the controller collector.

6.5 Evaluation of Dataset Creation

The dataset creation process involved a systematic approach to capturing BLE advertising data from a diverse range of IoT devices. By employing both the original RPi-based setup (Section 5.1) and the new multi-ESP32 monitoring probe (Section 5.2), we ensured comprehensive data collection across different environments. This dataset was then used in subsequent research [13] and in Chapter 7 of this thesis. We managed to create a dataset that was not available in the past, filling an important gap in the resources needed for BLE connection detection research. The dataset has been made publicly available[1] to facilitate further research in this area. By providing a structured resource for BLE connection detection and analysis, this dataset aims to support and enhance future research in this area (see [15] for more details).

Apart from using the dataset for machine learning model development, we have also evaluated the performance of our monitoring probes as detailed in Section 5.4. This evaluation showed that the new multi-ESP32 probe provided a slight improvement in packet capture reliability compared to the original RPi-based setup.

[1] https://github.com/hujon/BLE-ARD/

Chapter 7

Machine learning for BLE Connection Detection

Detecting BLE connection events accurately based on advertising timings presents a significant challenge due to the diverse behaviours of different devices. While the principle of using advertising gaps is established [14], simpler statistical approaches have shown limited effectiveness in detecting connections across multiple devices (described in Section 7.1.1 aswell as in referenced work [12, 13]), with many approaches failing to achieve acceptable precision or being unusable for real-time monitoring applications. More sophisticated methods, such as those based on machine learning, have shown potential in improving detection accuracy and reliability in a research made after creation of the dataset [12, 13], the best performance was achieved with a simple MLP classifier – it's largest drawback was inability to generalize across different devices.

This chapter explains the motivation (Section 7.2) for usage of machine learning in BLE connection detection, reviews the performance of prior methods (Section 7.1.1 and Section 7.1.2) and then describes the creation of a functional machine learning model for BLE connection detection. The model is trained and evaluated on the dataset created in Chapter 6 and is then, in next chapter, used for real-time detection of BLE connections in the monitoring probe (Chapter 8).

7.1 Limitations of Prior Detection Methods

Initial attempts to detect BLE connections using advertising data relied on statistical methods and later, more advanced anomaly detection and basic machine learning techniques. While some showed potential, significant limitations were found, particularly regarding performance, real-time applicability, and generalization across diverse devices.

7.1.1 Simple Statistical Methods

As detailed in initial proof-of-concept work [14] and in collaborative work [15], initial analyses employed simple statistical methods to detect BLE connections. These methods, namely SimpleStatistics and SlidingWindow, typically establish a baseline of normal advertising behaviour and then identify significant deviations from this baseline as potential connection events [12].

SimpleStatistics Method

The SimpleStatistics method maintains a moving average of advertising time intervals across the observation period [14]. It then uses an adaptive threshold based on the maximum observed difference to flag connection events, with any interval exceeding this threshold being classified as a potential connection [15].

SlidingWindow Method

The SlidingWindow method uses a fixed-size window of recent deltas to calculate a local mean and standard deviation [14, 13]. This approach is more adaptive to evolving patterns in the advertising intervals, as it considers only the most recent observations rather than the entire history. When a new interval significantly exceeds the local baseline, the SlidingWindow method flags this as a potential connection event [12]. This more localized approach can better handle devices with gradually changing advertising patterns but may be more sensitive to short-term variation [13].

These two statistical methods were also utilized in for the initial evaluation and comparison of the different monitoring setups (as discussed in Section 5.4). However, as shown in Table 7.1, their performance in accurately detecting connections was inconsistent across different environments and data collectors, often resulting in poor precision and high false positive rates, making them unreliable for practical use.

Table 7.1: Performance of simple statistical detection methods across environments [15]

Environment	**SimpleStatistics**		**SlidingWindow**	
	Precision	**Recall**	**Precision**	**Recall**
RPi BLE controller				
Shielded room	24.68%	50.37%	37.46%	65.37%
Office	24.42%	59.83%	26.12%	75.00%
Monitoring probe				
Shielded room	44.04%	72.89%	23.41%	87.22%
Office	32.90%	75.96%	23.69%	84.22%

These methods also have a significant drawback: they cannot be used for a real-time application, as they require a large amount of data to be collected before the detection can be performed. This is not suitable for a real-time application, where the detection should be performed as soon as possible after the connection is established.

7.1.2 Prior Advanced and Machine Learning Methods

Because of the limitations of simple statistical models, Hujňák et al. [12, 13] have evaluated several more sophisticated methods on the dataset [15], including time series forecasting (ARIMA), clustering (GMM), anomaly detection algorithms (Local Outlier Factor – LOF, One-Class SVM – OCSVM, Isolation Forest – iForest), and a supervised machine learning approach using a simple Multilayer Perceptron (MLP) classifier, with a single input neuron representing the current time delta.

As shown in Table 7.2 for representative devices like Bentech and Danalock, the simple MLP classifier significantly outperformed the other advanced methods when trained and

evaluated on specific devices, achieving near-perfect scores in some cases. The best performing MLP in that study used a [80, 70, 20] neuron structure with ReLU activation and a constant learning rate [13].

Table 7.2: Comparison of different detection methods on specific devices [13]

Method	Shielded Room			Office		
	Precision	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**
Bentech FP3 Lock						
ARIMA	79.9%	100%	88.8%	54.1%	100%	69.8%
GMM	59.6%	100%	74.5%	74.5%	100%	84.8%
LOF	93.7%	100%	96.8%	68.9%	77.8%	73%
OCSVM	16.9%	100%	28.8%	18.5%	100%	31.2%
iForest	100%	73.3%	84%	94.5%	66.7%	77.4%
MLP	100%	100%	**100%**	96.7%	100%	**98.3%**
Danalock V3						
ARIMA	47.9%	100%	64.7%	46.5%	100%	63%
GMM	48.1%	100%	63.7%	50.4%	100%	66.7%
LOF	93.3%	97%	95%	0%	0%	0%
OCSVM	2.1%	100%	4.1%	7.1%	100%	13.3%
iForest	89%	100%	94%	80%	100%	88.9%
MLP	97%	100%	**98.4%**	100%	100%	**100%**

However, a critical limitation emerged from this prior work: the simple MLP model, despite its success on individually trained devices (when summarised across all tested methods – 90.9% F1), failed to generalize when applied across different device types [13]. A model trained on data from multiple devices exhibited significantly degraded performance, achieving F1-scores often around 50% or lower when evaluated across the diverse dataset (Specifically shown in 7.4.3). This indicated that the simple MLP architecture, relying only on the most recent time delta, could not adapt to the varying baseline advertising intervals and patterns inherent to different BLE devices. This lack of contextual understanding, where a model cannot differentiate if a large advertising delta is normal for a slow-advertising device or indicative of a connection event for a fast-advertising device.

7.2 Motivation for Enhanced MLP Architectures

The critical limitation of the simple, single-input MLP approach, as highlighted in Section 7.1.2, became the primary motivation for this thesis. While prior work established the superiority of a supervised MLP for per-device detection, its inability to generalize when trained on a diverse dataset underscored the need for architectural improvements capable of leveraging more input context. As outlined in Chapter 4, our machine learning efforts were specifically focused on robust connection detection for devices showing intermittent advertising pattern.

Addressing the generalization gap identified with prior simple MLP approaches (Section 7.4) became the primary motivation for the work detailed in this thesis. While previous studies shown the superiority of supervised MLPs for per-device connection detection, their inability to generalize across diverse device types highlighted a critical need for architectural improvements [13]. As outlined in Chapter 4, our machine learning efforts were specifically

focused on robust connection detection for devices exhibiting intermittent advertising patterns.

Machine learning, particularly supervised neural network approaches, remains promising for identifying complex patterns in advertising data. This work therefore focuses on developing and evaluating new MLP-based (Section 7.6) and CNN-based models (Section 7.7) specifically designed to achieve improved generalization. We utilize PyTorch Lightning [7] for its flexibility in implementing and comparing different architectures, such as the single-input versus multi-input models detailed in the subsequent sections, aiming to create a detector that performs reliably across the heterogeneous landscape of real-world BLE devices by leveraging more temporal context from the input data.

7.2.1 Problem Formulation: Binary Classification

The core task of detecting BLE connections is formulated as a binary classification problem using a multilayer perceptron neural network (MLP). The MLP model is trained on a dataset containing features extracted from BLE advertising packet timings, primarily the time difference between consecutive packets (advertising delta). The model learns to predict the binary connection state (connected/not connected) based on these features. Once trained, the model can classify new sequences of advertising deltas to identify active BLE connections [28].

7.3 Data Preparation for Machine Learning

The dataset used for training and evaluating the machine learning models was created as described in Chapter 6, then it needed to be preprocessed and structured appropriately for the machine learning task. This section focuses on the data preparation steps, including the train-validation split, initial data filtering, feature engineering, and handling class imbalance.

It's important to note that the initial data filtering (Section 7.3.2) and feature engineering processes (Section 7.3.3) were not performed by the author of this thesis, as these steps were previously established in the work by Hujňák et al. [13]. The current work builds upon these existing preprocessing approaches while addressing class imbalance. The train-validation split methodology also follows a similar approach to that previous work, ensuring consistency in evaluation protocols across research efforts.

7.3.1 Train-Validation Split

For machine learning model development, we implemented a principled validation strategy. Since each action in the dataset was performed exactly five times for consistency (as mentioned in Section 6.4), we used this natural repetition structure to create our train-validation split. Specifically, for each device and action type, we reserved the first repetition (20% of the data) for the validation set and used the remaining four repetitions (80% of the data) for training. This approach ensured that our validation set contained examples of all action types across all devices while maintaining a standard 80/20 split ratio. This stratified splitting method preserved the distribution of different device types and actions in both the training and validation sets, allowing for a more reliable evaluation of the models' generalization capabilities.

7.3.2 Initial Data Filtering

Although half of the created dataset was captured in a shielded chamber to minimize external interference, it was not entirely free from noise. Minor imperfections in the shielding or internal equipment noise likely contributed to this interference. The other half of the dataset was collected in an office environment, which captured unintended advertising packets from nearby Bluetooth and Wi-Fi devices unrelated to the intended dataset.

To address these issues, the dataset was preprocessed by filtering out data from devices other than the intended target using their MAC addresses, as detailed in [15]. This initial filtering was performed as part of the dataset creation process and was not the author's contribution. The filtering ensured that only the target data was retained for analysis, eliminating irrelevant signals and unintended advertising packets.

7.3.3 Feature Engineering and Final Dataset Structure

The collected data, was stored in a CSV file containing the following columns: Timestamp, DeviceName, Advertising Type, Address Type, Address, RSSI, and Channel. For the machine learning part of this work, the dataset structure was enhanced based on the approach by Hujňák et al. (2023) [13], incorporating two additional columns crucial for the models:

- **Advertising Delta**: The time difference between consecutive advertising packets from the same device. This captures the timing patterns.
- **Connection**: A binary value indicating the state of connection of the device (1 for connected, 0 for not connected). This serves as the target label for supervised learning.

These two columns, `Advertising Delta` and `Connection`, along with potentially necessary identifiers (like Address or a processed sequence ID), formed the core features used by the machine learning models. The other columns from the original raw capture (like DeviceName, RSSI, Channel etc.) were excluded during the final feature selection for the specific models described later, as they were not directly used as input features in those particular implementations.

7.3.4 Dealing with Imbalanced Dataset

Imbalanced datasets, where certain classes significantly outnumber others, create a challenge in machine learning. In our dataset, focusing solely on the 'controller' and 'parallel' data, 324 positive instances (connected state) and 250,734 negative ones (not connected state) are provided. When 'parallel' data is further divided into its three channels, the dataset has 389 positive and 357,014 negative instances. Such high class imbalance can lead models to become biased towards the majority class, resulting in suboptimal performance on the minority class.

To mitigate this issue, two primary strategies were experimented with: weighting the loss function and implementing weighted sampling.

Weighting the Loss Function

Adjusting the loss function to assign higher penalties to misclassifications of the minority class encourages the model to learn its characteristics more effectively. In binary classification, this involves setting a higher positive class weight (pos_weight) to increase the loss

associated with misclassifying positive instances. However, during our initial exploratory trials, this approach had suboptimal results, as the model's performance on the minority class did not improve significantly. This outcome is consistent with the findings in the literature [30], where weighted loss functions may not always effectively address severe class imbalance.

Weighted Sampling

Due to the limitations of weighting the loss function, weighted sampling was used to address the class imbalance. This technique ensures that each class is represented proportionally during training. In PyTorch [26] the `WeightedRandomSampler` assigns sampling probabilities inversely proportional to class frequencies, ensuring that the model encounters a balanced representation of classes during training. This approach effectively mitigates bias towards the majority class and has been shown to improve model performance on our imbalanced dataset.

7.4 Model 1: Single-Input MLP Approach

For the initial approach and baseline benchmark, the simple Multilayer Perceptron (MLP) similar to that proposed by Hujňák et al. [13] was implemented using the PyTorch Lightning library. We use this simple implementation as a reference point against which we can evaluate our more advanced architectures. This method uses an single-input MLP, providing a comparison to make improvements in our next models (which is done in Section 7.8).

7.4.1 Implementation Details

The input feature is the 'Advertising Delta' between consecutive packets, which is scaled using scikit-learn's `StandardScaler` for normalization. The MLP outputs a binary classification: 1 if the connection state is active (True) and 0 otherwise.

A small feedforward neural network was designed with the following architecture:

- **Input Layer**: A single input neuron representing the scaled advertising delta.
- **Hidden Layer(s)**: Configurations with one to three fully connected layers (e.g., [5], [15,15], [70,30,20] neurons from our reference work [13]) using ReLU activation.
- **Output Layer**: A single neuron producing a logit, converted to a probability via a sigmoid function, which is then thresholded at 0.5 to classify the connection state.

The preprocessing pipeline for this model involved:

1. Loading data from CSV files for specific devices and environments.
2. Extracting only the `AdvertisingDelta` and `Connection` columns.
3. Standardizing the `AdvertisingDelta` values using `StandardScaler`.
4. Splitting the data into training and validation sets.
5. For the weighted approach, applying weighted sampling (`WeightedRandomSampler`) to address class imbalance as described in Section 7.3.4.

The model was trained using binary cross-entropy loss (`BCEWithLogitsLoss`) within the PyTorch Lightning framework. Training was performed both with weighted sampling and without it to compare approaches and mirror the methodology from [13].

Model training used 20 epochs with a batch size of 200 and a learning rate of 0.001, with early stopping after 5 epochs of no validation loss improvement. Training was conducted on a Ryzen 5 5600X CPU with 32GB RAM, using PyTorch Lightning for efficient model management and training.

7.4.2 Limitations of single-input MLP

Although this method demonstrates good performance on certain devices when trained specifically for them [13], it does not generalize well across different devices. The model struggles to adapt to variations in advertising intervals inherent to different devices, as it relies solely on the single advertising delta value without broader contextual information. For example, the Danalock device broadcasts ads approximately every 500 ms, while the Revogi device advertises roughly every 55 ms. Without considering the typical interval for a device, the neural network cannot reliably determine whether a given interval represents an active connection (a long gap) or simply a normal advertising event (which might be long for one device but short for another).

This lack of context implies that a single-input MLP, even if performing well when trained and tested on a specific device type, is unlikely to generalize effectively when trained on a dataset containing multiple devices and then applied across this diverse set. It would essentially require separate training for each distinct device type, reducing its practicality for real-world scenarios.

7.4.3 Experimental Results and Confirmation of Limitations

To quantify the performance of the single-input approach and its generalization capability when trained on data from all devices, various configurations were evaluated. Table 7.3 shows performance with class weighting, and Table 7.4 shows performance without it.

Table 7.3: Performance of baseline Single-Input MLP model with class weighting

Model Configuration	Precision	Recall	F1 Score	Accuracy
Single-Input MLP [5,5]	57.98%	77.30%	66.26%	99.85%
Single-Input MLP [15,15]	57.75%	76.60%	65.85%	99.84%
Single-Input MLP [70,30,20]	57.75%	76.59%	65.85%	99.84%

Table 7.4: Performance of baseline Single-Input MLP model without class weighting

Model Configuration	Precision	Recall	F1 Score	Accuracy
Single-Input MLP [5,5]	90.54%	47.52%	62.33%	99.89%
Single-Input MLP [15,15]	67.31%	49.65%	57.14%	99.85%
Single-Input MLP [70,30,20]	64.85%	75.89%	69.93%	99.87%

These results validate our concerns about the limited generalization capability of single-input models. While accuracy appears impressive (consistently >99.8%), this metric is misleading due to the significant class imbalance in our dataset. The F1-scores provide a more meaningful evaluation of performance. With class weighting (Table 7.3), we can

observe an improved recall but compromised precision, resulting in F1-scores that gets stuck around 70%. Without weighting (Table 7.4), precision improves but recall decreases substantially, leading to inconsistent F1-scores across different model configurations.

Even with the best F1-score achieved (69.93% for the [70,30,20] configuration), the model's performance is still much lower than device-specific models reported in referenced work (which achieved more that 90% F1-score [13]). This indicates that the single-input MLP approach is not suitable for generalizing across multiple devices, as it fails to capture the unique advertising patterns of each device type.

The next sections will describe the more advanced models that were developed to address these limitations and improve generalization across different device types.

7.5 Data Preparation for Sequence-Based Models

To provide temporal context to the more advanced models (Multi-Input MLP described in Section 7.6 and CNN from Section 7.7), a specific data preparation pipeline was implemented to create input sequences from the time series of advertising deltas:

1. Raw data containing `AdvertisingDelta` values for each target device was loaded.
2. Sequences of a fixed length N were constructed. For a sequence length of N=5, each input sample would be a vector $[\Delta_t, \Delta_{t-1}, \Delta_{t-2}, \Delta_{t-3}, \Delta_{t-4}]$, where Δ_t represents the time difference between consecutive advertising packets at time t from a specific device.
3. The entire sequence of N deltas within each sample vector was standardized using scikit-learn's `StandardScaler`.
4. The binary target label for supervised learning was determined by the presence of a connection (value 1) within the sequence, indicating that at least one of the deltas in the sequence was part of a connected state (3.1.2). If no connection was present, the label was set to 0.
5. The overall dataset was split into distinct training and validation sets.
6. To handle the inherent class imbalance (many more 'not connected' samples than 'connected'), weighted sampling (`WeightedRandomSampler`) was applied to the training dataloader based on the target labels.
7. Finally, the processed input sequences and corresponding labels were converted into PyTorch tensors suitable for model training and evaluation.

This pipeline generated the structured sequence data used as input for both the Multi-Input MLP and the 1D CNN models described next.

7.6 Model 2: Multi-Input MLP Approach

To leverage the temporal context prepared as described in Section 7.5, we developed and evaluated MLPs taking sequences of N advertising deltas as input. The hypothesis is that processing the sequence allows the model to learn device-specific rhythms and better identify anomalies in connection gaps, leading to improved generalization.

7.6.1 Architecture Exploration and Hyperparameter Tuning

The optimal architecture and hyperparameters for the MLP were determined through a systematic experimentation with many different configurations. The goal was to find a balance between model complexity and generalization performance across different device types.

We designed feedforward neural networks to process these input sequences. The general architecture explored included:

- **Input Layer:** N neurons corresponding to the sequence of N consecutive advertising deltas.
- **Hidden Layers:** Multiple configurations of hidden layers were tested:
  - Varying depths (1 to 3 hidden layers).
  - Different widths per layer (16, 32, 64, 128, and 256 neurons).
  - ReLU activation functions throughout.
  - Batch normalization between layers.
  - Dropout with rates ranging from 0.1 to 0.5.
- **Output Layer:** A single neuron with sigmoid activation for binary classification.

We investigated the impact of using different numbers of consecutive advertising deltas as input features, testing sequence lengths of 3, 5, 7, 10, 15 and 20. This exploration aimed to find the best temporal context window that balances sufficient historical information. For network depth and width configuration, we employed Optuna [1], an automated hyperparameter optimization framework, to systematically search the architectural space. The parameter sweep evaluated hundreds of architectural combinations, with the F1 score as our primary evaluation metric since it provides a balanced measure of precision and recall – critical for our imbalanced dataset where both false positives and false negatives carry significant implications for security monitoring. This comprehensive optimization approach helped identify architectures that balanced complexity with generalization capability across different device types.

Regarding training hyperparameters, we experimented with:

- **Optimizers:** Adam
- **Learning rates:** Ranging from 1e-4 to 1e-2
- **Batch sizes:** 32, 64, 128, 256
- **Weight decay:** Various settings to control overfitting

These networks were implemented in PyTorch Lightning and trained using binary cross-entropy loss (`BCEWithLogitsLoss`). The final architecture and hyperparameters were selected based on validation F1 score, particularly focusing on generalization across different device types.

As shown in Table 7.5, increasing the input sequence length consistently improved the best achievable F1-score. The model with sequence length N=20 achieved the highest overall performance with an F1 score of 96.7%. This suggests that providing more extensive

Table 7.5: Performance comparison of Multi-Input MLP with optimal hyperparameters for each sequence length

Sequence Length	Hidden Layers	Neurons per Layer	Precision	Recall	F1 Score
N = 3	3	[32, 16, 32]	85.9%	76.8%	81.1%
N = 5	3	[128, 128, 128]	83.6%	87.5%	85.5%
N = 7	3	[128, 128, 128]	86.0%	90.0%	88.0%
N = 10	3	[128, 128, 128]	92.0%	91.1%	92.9%
N = 15	3	[256, 128, 64]	93.9%	**96.5%**	95.2%
N = 20	3	[128, 128, 64]	**96.0%**	95.4%	**95.7%**

Note: Results reflect the best validation F1-score achieved via Optuna search for each sequence length across all tested MLP architectures (1-3 hidden layers).

temporal context, up to 20 prior advertising deltas, enables the model to better distinguish between normal advertising patterns and connection-related anomalies within this dataset.

Table 7.6: Optimal training hyperparameters for Multi-Input MLP at each sequence length

Seq. Length	Learning Rate	Dropout Rate	Weight Decay	Batch Size
N = 3	2.5e-4	0.118	4.9e-4	64
N = 5	4.7e-4	0.158	1.1e-4	64
N = 7	5.7e-4	0.12	2.1e-4	64
N = 10	5.4e-4	0.173	1.3e-4	128
N = 15	2.8e-3	0.164	1.3e-4	256
N = 20	1.2e-3	0.168	2.7e-4	256

Note: Hyperparameters correspond to the best performing run identified via Optuna search for each sequence length.

Table 7.6 presents the optimal training hyperparameters associated with the best performing model for each sequence length. Generally, as sequence length and model complexity (e.g. for N=15 and N=20 which used [256,128,64] neuron configurations) increased, larger batch sizes and slightly higher learning rates were found to be effective.

Performance by Number of Hidden Layers

To understand the impact of network depth, we also analyzed the best performance achieved for models with one, two, and three hidden layers, using a consistent input sequence length of N=10 for this specific comparison, as it represented a point of strong performance before the very long sequences.

Table 7.7: Best Multi-Input MLP performance for N=10 by number of hidden layers

Hidden Layers	Neurons per Layer	Precision	Recall	F1 Score
1	[64]	88.6%	89.2%	88.9%
2	[256, 16]	90.6%	90.8%	90.7%
3	[128, 128, 128]	**92.0%**	**91.1%**	**92.9%**

As shown in Table 7.7, for an input sequence length of N=10, increasing the number of hidden layers from one to three generally improved the F1-score, with the 3-hidden-layer configuration ([128, 128, 128]) achieving the best result (92.0%). This suggests that for this sequence length, a deeper architecture was beneficial for capturing the complex relationships within the advertising delta sequences.

However, it's important to note that while deeper architectures demonstrated better performance, they also require more computational resources. Since our final deployment target is a resource-constrained Raspberry Pi platform (described in 8), we must balance model performance against practical resource limitations. We limited our exploration to architectures with at most three hidden layers, as models with four or more layers are more likely to be computationally intense for real-time inference on the Raspberry Pi.

7.7 Model 3: 1D Convolutional Neural Network (CNN) Approach

As an alternative approach to processing the sequence data (prepared as described in Section 7.5), we also explored the use of a 1D CNN, drawing inspiration from its success in other sequence analysis tasks [20, 9]. The implemented architecture processes the input sequence of N advertising deltas as follows:

- **Input Reshaping:** The input sequence (Batch Size, N) is reshaped to add a channel dimension (Batch Size, 1, N).
- **Convolutional Layer:** A 1D convolutional layer (‘nn.Conv1d‘) with a kernel size equal to the input sequence length N and varying output channels (8, 16, 32, or 64) is applied. This layer acts as a feature extractor, identifying patterns across the entire input sequence span.
- **Pooling Layer:** An Adaptive Max Pooling layer (‘nn.AdaptiveMaxPool1d(1)‘) follows the convolution, reducing the sequence dimension to 1 by selecting the maximum activation for each feature channel.
- **Classifier:** A multi-layer classifier consisting of:
  - Dropout (rate between 0.15-0.4)
  - Linear layer mapping from output channels to hidden units (16-256)
  - ReLU activation
  - Another dropout layer
  - Final linear layer mapping to a single output logit

This CNN architecture aims to capture relevant temporal patterns across the entire advertising delta sequence while the multi-layer classifier helps prevent overfitting and improves generalization.

7.7.1 Architecture Exploration and Hyperparameter Tuning

Similar to the MLP approach, we explored variations in the 1D CNN architecture and training parameters to optimize its performance:

- **Input Sequence Length (N):** We tested the same input sequence lengths as the MLP (N = 3, 5, 10, 15, 20) to ensure a fair comparison of how each architecture utilizes temporal context.
- **Convolutional Layer Parameters:** For each sequence length N, we used a matching kernel size of N to analyze the entire sequence at once. We varied output channels (8, 16, 32, 64) to determine the optimal feature space size.
- **Network Depth:** We primarily focused on single-layer convolutional architectures, so we could keep the network simple and computationally efficient
- **Classifier Structure:** We systematically explored hidden unit counts (16, 32, 64, 128, 256) in the classifier's hidden layer and dropout rates (0.15 to 0.4) to determine the most effective configuration for preventing overfitting while maintaining good generalization.
- **Training Hyperparameters:** We used Optuna to systematically tune parameters including learning rate (1e-4 to 1e-2), weight decay (1e-5 to 1e-3), and batch sizes consistent with the MLP experiments, using the Adam optimizer through training.

This exploration aimed to identify the most effective 1D CNN configuration for this specific task. The final architecture was selected based on validation performance, and its detailed evaluation is presented alongside the MLP results in Section 7.8.

7.7.2 Model evaluation and Results

Similar to our approach with the Multi-Input MLP, we systematically explored various configurations of the 1D CNN model across different sequence lengths. The optimal architectures and their performance metrics are summarized in Table 7.8.

Table 7.8: Performance comparison of 1D CNN with different sequence lengths

Sequence Length	**Kernel Size**	**Output Channels**	**Hidden Units**	**Precision**	**Recall**	**F1 Score**
N = 3	3	32	256	90.8%	72.8%	79.2%
N = 5	5	32	16	90.2%	77.8%	80.2%
N = 10	10	64	32	93.3%	87.9%	88.5%
N = 15	15	64	256	**95.0**%	91.8%	93.3%
N = 20	20	32	256	93.3%	**96.8**%	**96.3**%

As shown in Table 7.8, the 1D CNN model with sequence length N=20 achieved the highest F1 score of 96.3%, with a precision of 93.3% and recall of 96.8%. This indicates that the model effectively captures the temporal patterns in advertising deltas and is able to find the outlier (which is the connection event) in the sequence. The model with N=15 also performed well, achieving a precision of 95.0% and recall of 91.8%, indicating that it was able to identify most of the connection events while maintaining a high precision.

The optimal training hyperparameters for each CNN configuration are presented in Table 7.9.

Table 7.9: Training hyperparameters for optimal 1D CNN configurations

Seq. Length	Learn Rate	Dropout	Weight Decay	Batch Size	Epochs
N = 3	8.5e-4	0.2	3e-4	32	5
N = 5	7.3e-4	0.28	2e-5	256	17
N = 10	1.7-4	0.16	7e-5	32	19
N = 15	8.1e-4	0.17	1e-5	64	19
N = 20	3.2e-4	0.26	1e-4	64	19

7.8 Comparison of models and discussion

Based on the results from the previous sections, we can summarize the performance of the three models (Single-Input MLP, Multi-Input MLP, and 1D CNN) in Table 7.10.

Table 7.10: Comparison of performance metrics for different model architectures across varying input sequence lengths (N)

Seq. Len.	Baseline MLP (F1)	Multi-Input MLP			1D CNN		
		Prec.	Rec.	F1	Prec.	Rec.	F1
N/A	69.9%	–	–	–	–	–	–
N = 3	–	85.9%	76.8%	81.1%	90.8%	72.8%	79.2%
N = 5	–	83.6%	87.5%	85.5%	90.2%	77.8%	80.2%
N = 7	–	86.0%	90.0%	88.0%	–	–	–
N = 10	–	93.0%	91.1%	92.9%	93.3%	87.9%	88.5%
N = 15	–	93.9%	96.5%	95.2%	95.0%	91.8%	93.3%
N = 20	–	96.0%	95.4%	**95.7%**	96.4%	97.1%	**96.3%**

Note: Single-Input MLP performance is shown as F1-score from its best configuration.

In Table 7.10, we can see that the Multi-Input MLP and 1D CNN models significantly outperform the baseline Single-Input MLP model across all sequence lengths.

The Single-Input MLP model (Section 7.4) achieved a maximum F1-score of only 69.9% when trained on the entire dataset, while both multi-input models demonstrated substantially higher performance with significantly better precision and recall metrics. The Multi-Input MLP model reached an impressive F1-score of 95.7% with a sequence length of 20, while the 1D CNN model achieved a slightly superior F1-score of 96.3% with the same sequence length.

Notably, the Multi-Input MLP model (Section 7.6) outperformed the 1D CNN model (from Section 7.7) at smaller sequence lengths (N=3,5,7,10,15), while the 1D CNN ultimately delivered better results with the largest sequence length (N=20). Both models achieved comparable precision and recall metrics, confirming their effectiveness at identifying connection events within advertising delta sequences.

An important consideration is the trade-off between performance and resources. While models with larger sequence windows (N=20) performed best overall, they require more computational resources and longer data collection times. For example, with a device having a 1000 ms advertising interval (like Bentech FP3), the system would need 20 seconds to gather the required advertising packets for N=20, compared to 15 seconds for N=15. This represents a 33% increase in waiting time for relatively modest performance gains (compar-

ing Multi-Input MLP with N=15 and 1D CNN with N=20), highlighting the diminishing returns of extending sequence length beyond certain thresholds.

Because of this performance-resource trade-off, we selected the Multi-Input MLP (Section 7.6) with N=15 as our final model for deployment on the Raspberry Pi. This model achieved a strong F1-score of 95.2% while maintaining a reasonable data collection time of (maximum) 15 seconds, making it suitable for real-time monitoring applications.

Chapter 8

Deployment

In this chapter, we describe the deployment of our trained neural network model for real-time monitoring of BLE devices (Section 8.1), the graphical user interface (GUI) developed for user interaction and visualization (Section 8.2), and the results of real-world validation experiments conducted to assess the model's performance in practical scenarios (Section 8.3).

8.1 Deployment of Neural Network

After training and validating our models (Section 7.8), we deployed the chosen architecture (Multi-Input MLP with N=15) on a Raspberry Pi 3B+. This deployment enables real-time monitoring of BLE devices in practical scenarios, allowing for immediate detection of connection events based on advertising patterns.

Although our analysis in Section 5.4 showed that the newly implemented triple ESP32 monitoring probe (5.2) did not significantly outperform the original single Raspberry Pi setup (5.1), we selected this configuration for deployment for two key reasons: First, it provides flexibility as the central data collection device can be easily replaced with more powerful hardware for neural network processing if needed. Second, ESP32 microcontrollers are more cost-effective than Raspberry Pi units, making the solution more economical for large-scale deployments.

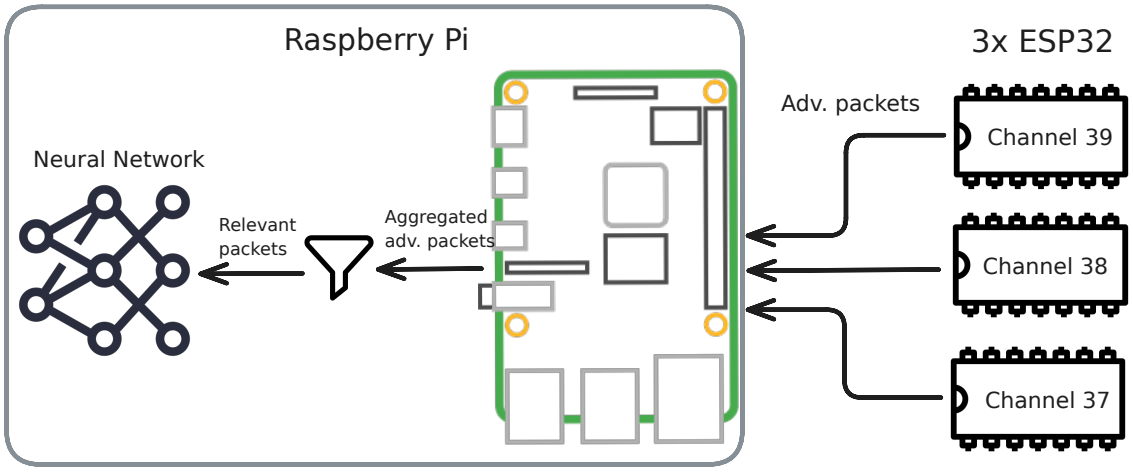

Figure 8.1: Deployment pipeline for the trained neural network on Raspberry Pi

The deployment pipeline, illustrated in Figure 8.1, integrates our trained model into a Python application running on the Raspberry Pi. This application enhances our data collection framework (from Section 5.2) to selectively process advertising packets only from user-designated devices. The system maintains a buffer of N consecutive advertising deltas per monitored device, which continuously feeds into the neural network for inference. Based on the model's output, the system registers connection events and generates notifications through the GUI (detailed in Section 8.2). This implementation allows for real-time monitoring of BLE device connections, effectively distinguishing between persistent and intermittent advertising patterns without requiring excessive computational resources.

8.2 Graphical User Interface (GUI) for Monitoring

While not the primary focus of this thesis, we developed a web-based interface to make the BLE monitoring system accessible for practical use. This interface bridges the technical complexity of the underlying system with an intuitive user experience essential for real-world security monitoring applications.

Architecture and Implementation

The GUI follows a client-server architecture with these components:

- **Backend**: Python application running on the Raspberry Pi that handles:
  - BLE device scanning and data collection
  - Neural network inference for connection detection
  - WebSocket server for real-time communication
- **Frontend**: HTML5/CSS3/JavaScript application that provides:
  - Responsive design
  - Real-time data visualization via WebSockets
  - Interactive device management controls

We implemented WebSockets for real-time communication between the backend and frontend, allowing the GUI to display live updates on device status and connection events.

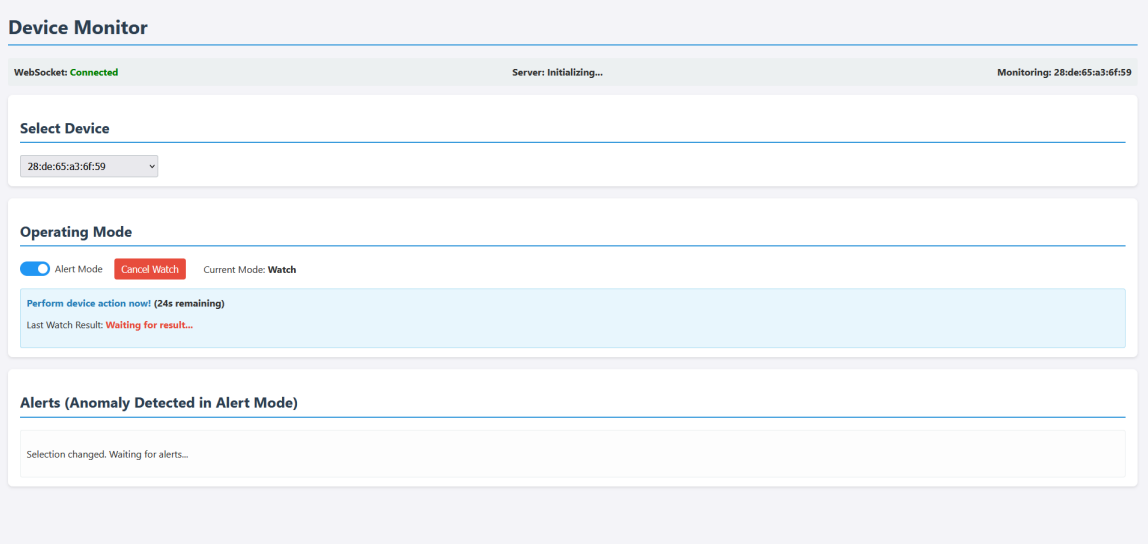

Figure 8.2: BLE monitoring interface showing device list, connection events log, and real-time status indicators

Key Features and Functionality

The GUI implements several essential features for effective BLE monitoring:

- **Device Discovery and Management**: Automated scanning identifies nearby BLE devices. Users can select specific devices for continuous monitoring, with MAC address filtering to focus computational resources on devices of interest.
- **Event Visualization**: Connection events detected by the neural network are displayed with timestamps, confidence scores, and status indicators. A chronological log maintains historical events for analysis.
- **Status Indicators**: Confidence metrics for ML predictions
- **Watch-Mode**: This feature implements a 30-second calibration process where the system:
    1. Monitors the selected device(s) to establish a baseline advertising pattern
    2. Prompts the user to do a specific action (e.g., lock/unlock)
    3. Records the advertising pattern during the action
    4. Lets the user know, if his device uses intermittent advertising or not

The interface prioritizes accessibility and usability while maintaining the technical depth required for security monitoring. It presents complex information about advertising patterns and ML-based detections in an interpretable format, eliminating the need for end users to possess specialized knowledge of BLE protocols or machine learning concepts.

8.3 Real-World Validation Results

Following the development, training, and Raspberry Pi deployment of our neural network models, we conducted real-world validation experiments using similiar methodology as described in Section 6.4 to judge its performance in a practical settings. These tests were performed in a university dormitory, an enviroment with significant traffic and interference from other devices – during our validation session, the monitoring probe detected approximately 62 other BLE devices in the proximity, perfectly simulating a real-world scenario where the system could be deployed. The primary goal of these experiments was not only to verify the detection accuracy, but also to confirm the practical usability of the implemented neural network on resource-limited hardware like the Raspberry Pi, as well as the overall pipeline effectiveness in a real-world setting.

For validation, we selected a representative subset from our comprehensive dataset (Section 6.2, Table 6.1). As shown in Table 8.1, this subset included one device with intermittent advertising pattern from each major category (lock, lighting, sensor) and one device with persistent advertising pattern (Philips Hue white). This selection allowed us to evaluate the models' performance across different device types and advertising behaviours.

Table 8.1: Devices used in validation experiments

Device class	**Device**	**Advertising Pattern**
Lock	**Danalock** V3-BTZE	Intermittent
Lighting	**Revogi** Bluetooth LED bulb	Intermittent
	Philips Hue **white**	Persistent
Sensor	**BeeWi Motion** Sensor	Intermittent

For each of the selected devices (both intermittent-pattern devices and the Philips Hue White as a persistent-pattern control), the operator performed approximately 15 connection-inducing actions at random intervals throughout each 10-minute validation test to simulate real-world usage scenarios. For intermittent devices, these actions triggered pauses in advertising that should be detected as connections. For the Philips Hue with its persistent advertising pattern, the same actions were performed but were expected to not trigger connection alerts since this device continues advertising even when connected. A „successful“ detection was defined when the system correctly responded to these initiated actions based on the device's known advertising behaviour – triggering an alert for intermittent devices or correctly not triggering one for persistent devices like the Philips Hue.

The summarized performance metrics (Precision, Recall, F1-Score) based on counting individual advertising packet classifications over the entire sessions are summarized in Table 8.2.

In terms of action-based success, the Danalock V3-BTZE and Revogi LED Bulb demonstrated a 100% success rate: all 15 connection actions that were performed for each device during testing correctly triggered a detection alarm, and there no false positive alarms for when no connection was active. This perfect performance highlights the model's effectiveness for these devices.

The Philips Hue White lightbulb, which was used as a control device with a persistent advertising pattern, did not trigger any false positive alarms during the test. This is ex-

Table 8.2: Summarized validation performance metrics per device (Multiline MLP N=15) based on packet classification

Device	**Precision**	**Recall**	**F1-Score**
Danalock V3-BTZE	1	1	1
Revogi LED Bulb	1	1	1
Philips Hue White	N/A	N/A	N/A
BeeWi Motion Sensor	0.93	0.87	0.90

pected behaviour, as the model should not detect connections for devices that continue to advertise even when connected.

Finally, the BeeWi Motion Sensor showed more complex behaviour during testing. Out of 15 motion-triggered connection events, 12 were correctly detected (True Positives), while 2 detection failures occurred (False Negatives) and 1 false positive was generated. Its important to note, that the missed detections only occurred when motion events were triggered quick after each other, suggesting the device might maintain its connection for a brief period after initial event detection – a behaviour not fully characterized during our dataset creation. Connection events labeled as „Expanded Data“ in our dataset (Section 6) were detected with perfect accuracy, as were isolated motion events with larger time separation. This pattern indicates that our model performs well for typical usage scenarios but may require refinement to handle edge cases involving rapidly repeated events. Because we are not totally sure about the behaviour of the device, we cannot say if the missed detections were caused by the model or by the device itself.

Overall, the primary objective of this validation was to verify that our neural network implementation functions effectively on resource-constrained hardware like the Raspberry Pi, and that the complete monitoring pipeline operates reliably in real-time environments with significant wireless interference. The results confirm that the deployed system maintains detection performance comparable to our offline evaluations, even if deployed in a noisy environment on a resource-limited platform.

Chapter 9

Conclusion

In this thesis, we have successfully developed an enhanced monitoring probe for capturing advertising packets from Bluetooth Low Energy devices (Chapter 5). Utilizing newly created probe at the same time as the original one, we collected a comprehensive dataset of advertising packets from many different BLE devices (Chapter 6). On this dataset, we were able to analyze behaviour of different devices and their advertising patterns, and also compare the performance of different monitoring probes (Section 6.5).

The created dataset was then used to train and evaluate multiple machine learning models for detecting BLE connection events (Chapter 7), which aimed to improve the accuracy of connection detection compared to statistical methods, aswell as to provide improvement over the single-input MLP model used in referenced work [13].

We explored various model architectures, including single-input MLPs with weighted sampling, multi-input MLPs and 1D CNNs that used multiple advertising deltas as input features for temporal context (Chapter 7). After evaluating all the different architectures (Section 7.8), we have found that all models performed much better than the simple statistical methods, with multi-input MLPs achieving an average of 90% F1-score across different configurations. The 1D CNN model with a sequence length of N=20 achieved the best overall performance, with an F1 score of 96.3% after training on the entire dataset. Other models with different sequence lengths also performed well, with almost all of them achieving F1 scores above 80%. This represents a substantial improvement over the single-input MLP model used in referenced work [13], which achieved only 66% F1 score on the same dataset. These results indicate that using multiple advertising deltas as input features for the model significantly improves the detection accuracy and generalization capability across different device types.

Then, we have chosen the best model for our use-case scenario (Section 7.8), the Multi-Input MLP with N=15, which even though was not the best performing model, was selected for its balance between performance and time required to collect the data. This model achieved a strong F1-score of 95.2% while maintaining a reasonable data collection time (5 packets less than best performing 1D CNN with N=20 while having F1 being only 1.1% lower than its superior counterpart), making it suitable for real-time monitoring applications, and deployed it on a Raspberry Pi 3B+ platform (Chapter 8). The deployment pipeline was designed to enable real-time monitoring of BLE devices, allowing for instant detection of connection events based on advertising patterns. We also developed a web-based graphical user interface (GUI) to provide an intuitive user experience for monitoring BLE devices, making the system accessible for practical use (Section 8.2).

Finally, we have done several real-world validation experiments to evaluate the performance of our deployed system in a practical setting (Section 8.3). The results confirmed that our neural network implementation functions effectively on resource-limited hardware like the Raspberry Pi, and that the complete monitoring pipeline operates reliably in real-time environments with significant wireless interference, as the system demonstrated high accuracy in detecting connection events for various BLE devices.

9.1 Future Work

While this thesis had promising results, there are several more pathways that could be explored during future research to further enhance the BLE connection detection system. Some of these directions include:

- **Expanded Device Dataset**: Expanding the dataset even further to include a much wider variety of BLE devices would improve the model's generalization capabilities.
- **Untrained Device Validation**: Our current evaluation focused on devices represented in the training data. Future work should explicitly test the system on completely new, untrained device types to validate true generalization capabilities across the broader BLE device ecosystem.
- **Artificial Dataset Generation**: Generating synthetic advertising patterns using generative models (e.g., GANs) could help augment the dataset and improve model robustness against overfitting.
- **Attention Mechanisms**: Implementing attention-based neural network architectures could potentially improve the model's ability to focus on the most relevant portions of advertising sequences, particularly for devices with variable advertising behaviours.
- **Anomaly Classification**: Extending the binary classification approach to distinguish between different types of connections (e.g., authorized vs. unauthorized) would enhance the security monitoring capabilities.
- **Recurrent Neural Networks**: Implementing RNN architectures, particularly Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) networks, could further improve the system's ability to model the temporal dependencies in advertising sequences, potentially capturing more complex patterns that our CNN architecture might miss.
- **Real-Time training and adaptation**: Creating a machine learning pipeline that allows the model to adapt to new devices and advertising patterns in real-time would enhance the system's robustness and flexibility.

These directions for future research would build upon the foundations established in this thesis and potentially lead to even more effective and practical BLE connection monitoring solutions.

Bibliography

[1] Akiba, T.; Sano, S.; Yanase, T.; Ohta, T. and Koyama, M. Optuna: A Next-generation Hyperparameter Optimization Framework. In: ACM. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, p. 2623–2631. Available at: https://github.com/optuna/optuna. Accessed: 2025-04-01.

[2] Arroyo, J. G. d.; Bindewald, J. and Ramsey, B. Securing Bluetooth Low Energy Enabled Industrial Monitors. In: *International Conference on Cyber Warfare and Security*. Academic Conferences International Limited, 2017, p. 167–176.

[3] Bluetooth Special Interest Group. *Bluetooth Core Specification, Version 6.0, Volume 1, Part A: Architecture*. Bluetooth Special Interest Group, 2014. Available at: https://www.bluetooth.com/specifications/bluetooth-core-specification/. Accessed: 2025-05-07.

[4] Bluetooth Special Interest Group. *Developer Study Guide: Bluetooth® Low Energy Security Bluetooth® Technology*. 2024. Available at: https://www.bluetooth.com/bluetooth-resources/le-security-study-guide/. [Online; Accessed 2025-03-20].

[5] Boukerche, A.; Zheng, L. and Alfandi, O. Outlier Detection: Methods, Models, and Classification. *ACM Comput. Surv.* New York, NY, USA: Association for Computing Machinery, june 2020, vol. 53, no. 3. ISSN 0360-0300. Available at: https://doi.org/10.1145/3381028.

[6] Cordeiro, J. R.; Raimundo, A.; Postolache, O. and Sebastião, P. Neural Architecture Search for 1D CNNs-Different Approaches Tests and Measurements. *Sensors*, november 2021, vol. 21, no. 23, p. 7990. PMID: 34883994; PMCID: PMC8659883.

[7] Falcon, W. et al. PyTorch Lightning. In:. GitHub, 2019. Available at: https://github.com/Lightning-AI/lightning. Accessed: 2024-08-15.

[8] Gomez, C.; Oller, J. and Paradells, J. Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. *Sensors (Basel, Switzerland)*, 2012, vol. 12, no. 9, p. 11734–11753. Available at: https://doi.org/10.3390/s120911734.

[9] Goodfellow, I.; Bengio, Y. and Courville, A. *Deep Learning*. Cambridge, MA, USA: The MIT Press, 2016. ISBN 978-0262035613. Available at: http://www.deeplearningbook.org.

[10] GOOGLE. *Google machine learning*. Available at: https://developers.google.com/machine-learning/crash-course/classification. Accessed: 2025-02-05.

[11] HEINRICH, A.; STUTE, M. and HOLLICK, M. BTLEmap: Nmap for Bluetooth Low Energy. In: *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2020, p. 331–333. WiSec '20.

[12] HUJŇÁK, O. *Security of Wireless Communication for IoT Devices*. Brno, Czech Republic, 2024. Ph.D. Thesis. Brno University of Technology. Supervisor HANÁČEK, P. Accessed: 2025-02-05.

[13] HUJŇÁK, O.; HOLOP, P.; MALINKA, K.; RES, J. and HANÁČEK, P. *Machine Learning Supported Bluetooth Low Energy Device Detection*. 2024. Unpublished manuscript, Accessed: 2025-02-14.

[14] HUJŇÁK, O.; MALINKA, K. and HANÁČEK, P. *Indirect Bluetooth Low Energy Connection Detection*. 2023.

[15] HUJŇÁK, O.; MALINKA, K.; OLEXA, M. and HANÁČEK, P. *Parallel BLE Advertising Monitoring*. 2024. Unpublished manuscript, accessed: 2025-05-05.

[16] IBM. *Machine Learning: What it is and Why it Matters*. N.d. Available at: https://www.ibm.com/think/topics/machine-learning. Accessed: 2025-05-02.

[17] IEEE. *IEEE Standard for Low-Rate Wireless Networks*. IEEE Std 802.15.4-2020. IEEE, 2020. 1–800 p. Revision of IEEE Std 802.15.4-2015.

[18] IOFFE, S. and SZEGEDY, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: BACH, F. and BLEI, D., ed. *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. PMLR, 07–09 Jul 2015, vol. 37, p. 448–456. Proceedings of Machine Learning Research. Available at: https://proceedings.mlr.press/v37/ioffe15.html.

[19] IOT ANALYTICS. *State of IoT 2023: Number of connected IoT devices growing 16% to 16.7 billion globally*. 2023. Available at: https://iot-analytics.com/number-connected-iot-devices/. Accessed: 2025-05-02.

[20] KIRANYAZ, S.; AVCI, O.; ABDELJABER, O.; INCE, T.; GABBOUJ, M. et al. 1D convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 2021, vol. 151, p. 107398. ISSN 0888-3270. Available at: https://www.sciencedirect.com/science/article/pii/S0888327020307846.

[21] KOULOURAS, G.; KATSOULIS, S. and ZANTALIS, F. Evolution of Bluetooth Technology: BLE in the IoT Ecosystem. *Sensors*, 2025, vol. 25, no. 4. ISSN 1424-8220. Available at: https://www.mdpi.com/1424-8220/25/4/996.

[22] LECUN, Y.; BOTTOU, L.; BENGIO, Y. and HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. IEEE, 1998, vol. 86, no. 11, p. 2278–2324.

[23] MANNING, C. D.; RAGHAVAN, P. and SCHÜTZE, H. Introduction to information retrieval. Cambridge University Press, 2008. Accessed: 2025-02-05.

[24] Nielsen, M. *Neural Networks and Deep Learning*. Determination Press, 2018. Available at: http://neuralnetworksanddeeplearning.com/. Accessed: 2025-02-05.

[25] Olah, C. *Understanding Convolutions*. 2014. Available at: https://colah.github.io/posts/2014-07-Understanding-Convolutions/. Accessed: 2025-05-11.

[26] PyTorch Team. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. GitHub, 2023. Available at: https://github.com/pytorch/pytorch. Accessed: 2025-05-11.

[27] Rohde & Schwarz. *Bluetooth Adaptive Frequency Hopping on a R&S CMW*. December 2016. Available at: http://www.rohde-schwarz.com/appnote/1C108. [Online; Accessed 2025-12-05].

[28] Rumelhart, D. E.; Hinton, G. E. and Williams, R. J. Learning Representations by Back-Propagating Errors. *Nature*, Oct 1986, vol. 323, no. 6088, p. 533–536.

[29] Sasaki, Y. The Truth of the F-measure. Available at: https://nicolasshu.com/assets/pdf/Sasaki_2007_The%20Truth%20of%20the%20F-measure.pdf.

[30] Sinha, S.; Khor, J.; Latha, R.; Latha, N. and Cashman, D. Class-Wise Difficulty-Balanced Loss for Solving Class-Imbalance. In: *Proceedings of the Asian Conference on Computer Vision (ACCV)*. November 2020, p. 139–155. Available at: https://openaccess.thecvf.com/content/ACCV2020/papers/Sinha_Class-Wise_Difficulty-Balanced_Loss_for_Solving_Class-Imbalance_ACCV_2020_paper.pdf.

[31] Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I. and Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research (JMLR)*, 2014, vol. 15, no. 56, p. 1929–1958. Available at: http://jmlr.org/papers/v15/srivastava14a.html.

[32] Stanford University. *CS231n: Convolutional Neural Networks for Visual Recognition*. 2023. Available at: https://cs231n.github.io/. Online course materials; Accessed: 2025-02-20.

[33] Texas Instruments. *Bluetooth and Wi-Fi Coexistence*. 2019. Available at: https://www.ti.com/pdfs/vf/bband/coexistence.pdf.

Appendix A

Poster

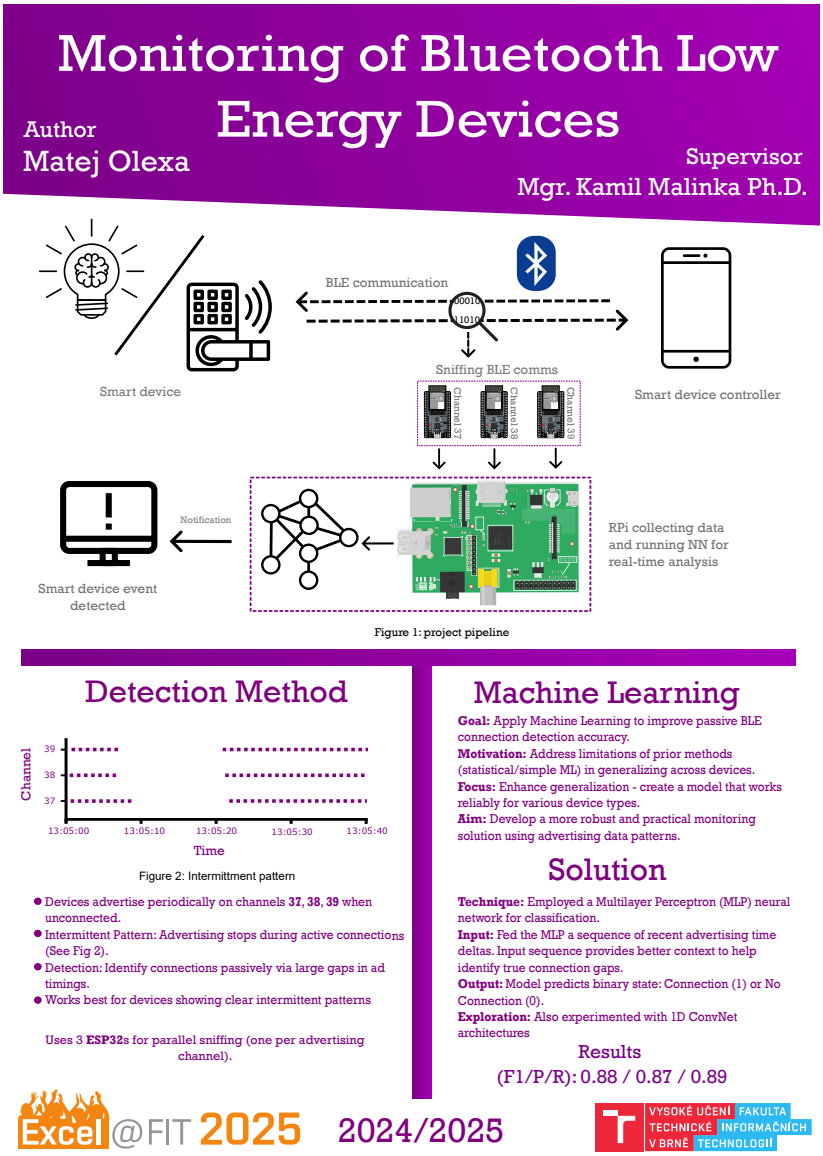

Figure A.1: Poster of the thesis used for EXCEL@FIT 2025 conference