# A system for collecting labelled data and evaluating AI algorithms for in-ear detection in hearing protection

Dominika Agata Limanówka-Pieniak

Thesis submitted for the degree of
Master in Applied Computer and Information Technology (ACIT)
30 credits

Department of Computer Science
Faculty of Technology, Art and Design

OSLO METROPOLITAN UNIVERSITY

Spring 2024

# A system for collecting labelled data and evaluating AI algorithms for in-ear detection in hearing protection

Dominika Agata Limanówka-Pieniak

A system for collecting labelled data and evaluating AI algorithms for in-ear detection in hearing protection

http://www.oslomet.no/

Printed: Oslo Metropolitan University

# Abstract

This thesis investigates the integration of artificial intelligence (AI) algorithms to precisely determine the status of an earplug within hearing protection devices, discerning whether it is correctly inserted in the ear or not. Motivated by the global prevalence of noice induced hearing loss and the need for effective protection, the study holds the potential to contribute towards increased safety in noisy work environments. Working with Minuendo, an innovative hearing-protection solutions company, it is addressing the issue by adopting AI in practical context.

Even though AI is becoming increasingly popular, its introduction into industrial settings presents multiple challenges. This thesis addresses these challenges by designing and implementing a system on cloud infrastructure, using modern practices that streamline the setup of the necessary architecture. This allows for straightforward experimentation with machine learning algorithms, supporting iterative development and ensuring reproducibility of results. The study merges theoretical insights with real-world applications, facilitating the integration of cutting-edge technology into industrial environments.

A significant aspect of this thesis is the exploration of various machine learning algorithms for detecting the in-ear placement of earplugs. Through initial analysis, strategic data gathering, and the pilot implementation of diverse machine learning algorithms, this research provides initial insights into their effectiveness and practicality.

# Acknowledgments

I am profoundly grateful to my academic supervisor at the university, Pål Halvorsen. His support and insightful comments were crucial in shaping the final version of this thesis. I extend my heartfelt thank you for all professors and mentors who have significantly influenced my educational journey. Their guidance and encouragement have been instrumental in shaping my academic path. Each interaction has left a lasting impact, and I am genuinely grateful for the invaluable support and wisdom they have provided along the way.

Special thanks are due to Olav Kvaløy from Minuendo for his invaluable domain knowledge in auditory safety and hearing protection. His mentorship provided me with the unique opportunity to apply theoretical concepts to a real-world problem, enriching my research experience. I am also thankful to the entire team at Minuendo, who welcomed me into the company and provided the necessary data and tools for my research.

I would also like to express my deepest appreciation to my family and friends for their love and unwavering support. They have always believed in me and encouraged me to pursue my passions, providing a foundation of strength and motivation.

Lastly, I must acknowledge my husband for his willingness to accompany me on the adventure of studying for our master’s degrees in Oslo and for always being there for me, cheering me on, and believing in me when I needed it most. Oskar, I am forever grateful to have shared this experience with you by my side.

To everyone who has contributed to this journey, thank you!

# Preface

The master's studies period is undoubtedly an incredibly exciting one. Having already gained foundational knowledge in a specific field, this stage allows for a focused exploration of one's areas of interest. It offers the opportunity to delve in-depth into a chosen subject to achieve expertise while fostering a broader perspective by recognizing connections with diverse realms of science and industry.

I particularly appreciate my pursuit of a master's degree in Applied Computer and Information Technology, especially because of its applied nature. Throughout my academic journey, the vision of the direct application of knowledge in real-life scenarios attracted me the most. For this reason, I was very committed to the idea that my master's thesis would employ the knowledge and experience I had gained during the studies into a real-life, practical case. I seek to utilize academic knowledge as a bridge between academia's theoretical landscape and industry's practical domain.

This thesis has been created in collaboration with Minuendo, a company dedicated to advancing hearing protection technologies. Their work in this field is inspiring, and it is an honor to contribute even a small part to their efforts. I hope this thesis can provide valuable insights that could contribute to their ongoing efforts to improve safety in noisy work environments.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Increasingly, industrial safety relies not just on protective equipment but on the smart integration of technology to ensure that such equipment is used effectively. In this evolving landscape, several safety concerns demand attention, with hearing protection being one of the prominent examples. Especially in environments with high noise levels, addressing the issue of hearing protection becomes imperative, as exposure to loud noises can be a significant risk to individuals' long-term well-being. This thesis, developed in collaboration with Minuendo - an innovative company in hearing safety - explores the potential of artificial intelligence (AI) to enhance the effectiveness of ear protection while navigating the challenges associated with this process.

## 1.1 Motivation

According to statistics from the World Health Organization (WHO), the global prevalence of hearing loss exceeds 1.5 billion individuals [54]. Hearing loss makes communication and daily functioning much more difficult, leading to mental health challenges. Numerous studies have found a connection between hearing loss and increased depressive symptoms, especially among older adults [14, 31] and adolescents [12]. Research also indicates that hearing loss is linked to a higher risk of dementia [43, 76], highlighting the broader impact of this condition on cognitive health.

A variety of factors, including genetic susceptibility, infections, exposure to loud noises, medications, and lifestyle preferences, can contribute to hearing loss [54].

Noise-induced hearing loss stands out among this array of factors, as it could be completely prevented through early attention and care. Noise-induced hearing loss (NIHL) presents a unique challenge as its effects may not become evident until later in life. The gradual buildup of damage due to inadequate hearing care may go unnoticed until it is too late to reverse. Moreover, hearing damage is affected not only by the volume of surrounding noise but also by the duration of the exposure. Without appropriate measurement tools, it is difficult to determine whether safe thresholds for noise exposure have been exceeded, especially when people spend a significant amount of time in noisy environments, such as workplaces. The magnitude of the problem is confirmed by the fact that occupational noise-induced hearing loss is currently the most prevalent occupational disease in the world [13]. In response to this challenge, Minuendo has introduced the Smart Alert. This cutting-edge solution combines advanced hearing protection with noise monitoring technology, enabling users to receive real-time alerts when safe noise levels are exceeded and track their exposure values in the cloud application for comprehensive monitoring of their hearing health.

The Smart Alert solution is a product that already helps protect the hearing of workers, but its potential could be further enhanced by integrating artificial intelligence to improve earplug placement detection. Correctly detecting whether earplugs are in the ears is important for proper data analysis and determining actual noise exposure. In the current landscape, AI has gained extensive popularity for its transformative capabilities across various sectors. AI has the exceptional ability to improve efficiency, reduce costs and support the decision-making process. Both businesses and individuals are increasingly integrating AI into their operations. For Smart Alert, AI could be applied to analyze data from the earplugs to either replace or provide insights for the existing algorithm, thereby ensuring a more accurate assessment of the earplug status. This use of AI could enhance user safety by providing more precise and reliable detection which is crucial for accurate safety alerts and enhanced protection for users.

## 1.2 Problem Statement

The integration of AI into various sectors promises significant enhancements in efficiency and capability. However, the adoption of AI technologies is not without its challenges and risks. Key hurdles include substantial initial investments required for infrastructure, training, and development, as well as the inherent risks associated with uncertain returns on investment. These challenges can discourage organizations from fully exploring the potential of AI technologies.

Furthermore, the complexity of deploying AI systems involves technical and operational challenges that can complicate the direct application of AI in products. The risks associated with data privacy, the need for tailored solutions, and the management of continuous learning and adaptation of AI systems in dynamic environments pose significant barriers. In the context of hearing protection, the potential of AI for enhancing in-ear detection has been acknowledged. However, realizing this potential hinges on obtaining accurately labeled data, selecting suitable algorithms, and rigorously testing these algorithms to build trust.

This study addresses these issues by aiming to facilitate the adoption of AI methods for in-ear detection. Therefore, this thesis focuses on two main objectives:

- **Design and Implement a System**: To develop and implement a robust system that facilitates experimentation with AI technologies specifically for in-ear detection.
- **Evaluate AI Classification Algorithms**: To evaluate the performance and limitations of popular classification algorithms in accurately determining the in-ear placement of Smart Alert hearing protection devices.

This dual approach not only tests the practical applications of AI for the Smart Alert solution but also contributes to its ongoing development and refinement in an applied scenario.

## 1.3 Scope

The project aims to navigate an introduction of an AI-based system hosted on a cloud infrastructure to facilitate the exploration of AI capabilities for the real life case of a small company. This section outlines the scope of this thesis, focusing on the key components of the project. The scope of this research covers delving into several aspects of AI integration.

**Initial analysis** Before delving into the specifics of AI integration and creating cloud infrastructure, the research conducts an initial analysis to understand the context in which the AI would be introduced. This involves learning about the currently existing infrastructure and processes, digging deeper into the problem that AI would solve and identifying potential obstacles. All of this is intended to help develop the optimal solution and is expected to lead to more informed decision-making.

**Data collection** Data collection is one of the most important factors for introducing supervised learning methods. In order to do this, it is necessary to design a process for gathering data, create an infrastructure for its uploading and labeling, and implement preprocessing of this data and a manner for storing it.

**Process design** All steps should make use of current knowledge in the area of effective software development as much as possible. Process design should be embedded into existing workflows as much as possible and follow the principles of agile development.

**Infrastructure preparation** The project also covers the technical side of preparing the necessary infrastructure for managing the data, running the AI algorithms provided in the open sourced, commercially usable libraries, and visualizing the results.

**Utilizing existing AI algorithms** In this study, existing AI algorithms from open-source libraries will be utilized for in-ear detection. These algorithms will be tuned by adjusting parameters to optimize their effectiveness for this specific application. The performance of each algorithm will be evaluated based on accuracy and other metrics.

**Evaluation of the project** Among other things, the project's evaluation will present what has been accomplished, as well as the problems encountered, along with suggestions for their solution.

## 1.4 Limitations

Despite the comprehensive nature of this study, there are several limitations that should be acknowledged. It is also important to highlight the weaknesses of the project, which were identified at the start of the work.

**Algorithm Research and Development** The project assumes using existing AI algorithms implemented in the open-sourced libraries for model development, without getting into creating new algorithms.

**Regulatory Compliance** Compliance with specific regulatory requirements or industry standards is beyond the scope of the project. Ensuring compliance may require additional legal and regulatory expertise, which is not covered in this project.

**Limited Diversity and Volume of Data** Due to privacy aspects and the necessity of existence of labels, the data for the project will be collected by volunteers from Minuendo and the project developer. For this reason, there is a risk that a relatively small amount of data will be collected, which may not be very diverse.

## 1.5 Research Methods

The study serves as a comprehensive attempt to combine academic knowledge with real-world cases and theoretical insights with practical implementations. Research methods, based on principles of theory, abstraction and design, are expected to navigate complexity in both academic and practical contexts. This approach to computer science problems was drawn from the report "Computing as a Discipline", which discusses the nature of computer science as a discipline and proposes such a framework for handling this very broad and diverse field [19]. In this study, a holistic approach encompassing all three paradigms—theory, abstraction, and design—will be employed.

**Theory** The theory paradigm involves gathering information and understanding the existing knowledge base related to the topic of AI-based systems introduction in the industry. A comprehensive literature search is conducted at the outset to establish the theoretical framework and identify potential challenges and available solutions relevant to the study. This step lays the foundation for further exploration and analysis. It is also expected to facilitate informed decision-making when it comes to planning and implementing the other aspects of the project.

- The theory paradigm is directly applied to our work as it forms the foundational knowledge upon which all algorithmic and architectural decisions are based, ensuring that the development of AI systems for in-ear detection is grounded in well-established computer science principles and relevant AI research.

**Abstraction** Abstraction, akin to experimental science, forms hypotheses and analyzes the outcomes. The experiments are conducted to gather data on model training and validation metrics. This method simplifies complex phenomena, enabling for the investigation and empirical testing of the ideas. In my project, abstraction allows for the systematic testing of different AI algorithms, making it possible to distill complex data interactions into understandable models that predict in-ear detection efficacy.

- Abstraction is applied to this project as it allows for the formulation and testing of hypotheses about AI performance in in-ear detection to empirically validate and refine our AI models.

**Design** The design paradigm is the one that mainly guides the design and implementation of this research methodology, facilitating a seamless integration of theoretical ideas into practical applications. Within the scope of the project, a scalable infrastructure is being set up to conduct experiments and a process is being designed to collect data that will later be used by AI algorithms.

- The design paradigm is applied to my work as it guides the practical application of theoretical and abstracted insights into a tangible AI system. This involves creating an infrastructure that can support continuous experimentation with AI algorithms.

Building upon the established frameworks of design, theory, and abstraction, elements of both qualitative and quantitative research are to be incorporated to provide a comprehensive view of the created system. The system will be evaluated not only through the collection of quantitative data, such as algorithm accuracy metrics, but also through the gathering of qualitative feedback from domain experts within the company. This dual approach ensures that a holistic understanding of both the system's and algorithms performance and its practical efficacy in real-world applications is achieved.

Additionally, the research will employ iterative experimentation, a methodology favored in modern software development for its flexibility and rapid feedback integration. This technique allows for continuous refinement and optimization of the AI solution, ensuring that the development process remains dynamic and responsive to emerging needs and findings throughout the duration of the thesis.

## 1.6 Ethical Considerations

Ethical issues related especially to the use of data are paramount in any AI project. Given the sensitive nature of the data, discussions of ethical issues are key to ensuring responsible and transparent practices throughout the duration of the project.

The data collected during the project comes from earplug sensors. Among them are data from the microphones in the form of frequency values, from which, however, it is not possible to reproduce recordings of actual sound and conversations - the data only stores information about noise levels. Therefore, the data was not classified as confidential information and could easily be stored and processed on Cloud services at the department of computer science.

Volunteers inside the company are in charge for data recording, keeping a collaborative approach. Privacy issues are handled with care, protecting confidentiality and adhering to ethical norms. Every participant has agreed to the procedure of data collecting and has been properly informed about its purpose. They have given their consent knowing exactly how their data will be used, so there is accountability and openness at these issues.

There is also the matter of whether a potential bias exists. Because one of the study's drawbacks is the relatively small and non-diverse research sample, the issue of bias should be acknowledged while evaluating the project's outcomes.

## 1.7 Main Contributions

At the heart of this thesis lies a commitment to applying academic knowledge in practical context. This thesis adopts a comprehensive approach to enable the introduction of AI for in-ear detection for Smart Alert data. This creates an opportunity to leverage the AI technology to improve the product that protects the hearing of employees working in noisy environments and prevents NIHL, which is the most common occupational disease.

The main contribution of this thesis is establishing a systematic framework and processes essential for conducting experiments with AI, while utilizing an agile methodology to streamline experimentation. The devised system not only enables seamless implementation of AI but also incorporates a robust data gathering process, recognizing the paramount importance of high-quality data in AI applications.

Moreover, the project evaluates the efficacy of common AI algorithms in classifying data from Smart Alert sensors. By conducting thorough comparison and analysis, it uncovers valuable insights into how these algorithms perform and their suitability for improving hearing protection devices. The evaluation provides a summary of the experiments conducted.

## 1.8 Thesis Outline

This thesis is divided into six chapters, each of which serves a particular purpose. The following summary provides an overview of the remaining five chapters.

**Chapter 2 - Background and Related Work** The Background chapter covers three major sections. To begin, it presents a detailed review of the Smart Alert product and the associated problem, which could possibly be handled through the use of AI technology. Second, it provides a comprehensive overview of common AI algorithms that are suitable for addressing the classification problem. Finally, the chapter delves into the necessary Cloud-related background, covering the foundational concepts and technologies required for setting up the infrastructure to support the proposed AI solutions effectively.

**Chapter 3 - Design and Implementation** This chapter defines the methodological framework used to accomplish the stated objectives of the study. It includes a detailed plan for conducting the research, along with a description of each step.

**Chapter 4 - Results** This chapter presents the evaluation of created system and its components. It also includes the analysis of the pilot iterations.

**Chapter 5 - Discussion** The Discussion chapter serves as a platform to share insights derived from the research and its relevance to the broader field. It delves into the analysis of findings, focusing on the applicability of integrating AI for in-ear detection problem.

**Chapter 6 - Summary and Conclusions** The final chapter draws conclusions from the study, followed by a concise summary highlighting key points. Finally, the chapter concludes by suggesting potential directions for future research.

# Chapter 2

# Background and Related Work

This chapter is structured into three main sections, each providing a comprehensive background for this thesis. In the initial part, we will introduce the Smart Alert product and its specifications, followed by a detailed exploration of the problem and the associated data. The second section will present the key concepts of AI that are relevant to this thesis, laying the groundwork for understanding their practical applications. It will explore classification techniques in detail and discuss prominent tools and technologies in the field. In the final section of the chapter, modern operational strategies for developing and managing ML projects are discussed, starting with an exploration of challenges in AI integration within the industry. Emphasizing efficiency in workflow integration and the use of advanced tools, these strategies aim to facilitate the effective development of ML models.

## 2.1 Smart Alert

Smart Alert [6] is a complete solution that consists of a cloud service, a wearable neckband with earplugs, and a dock for charging the earplugs and syncing the data to the cloud. All these components are shown in Figure 2.1.

This solution is intended to provide effective hearing protection. Earplugs developed for workers should be put into their ears to give hearing protection in noisy situations, and microphones in the earplugs and on the collar record and continuously analyze noise levels in real time to notify workers when their total exposure during the day exceeds a safe threshold. Importantly, the device records only noise levels and does not capture or record actual sound. This data is sent to the

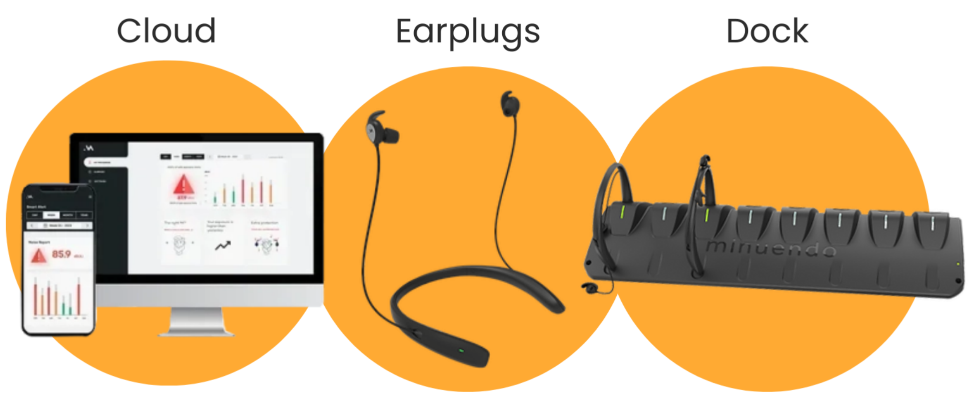


Figure 2.1: Smart Alert Components

cloud when inserted into the dock and can be viewed on dashboards available on a web-based application accessible to employees and safety managers.

### 2.1.1 Product Specification

At the epicenter of this project lies the wearable neckband with earplugs. This device was designed to be worn during daily work activities and it is equipped with three microphones: one in each earplug and one on the neckband, labeled M1, M2 and M3 respectively in Figure 2.2. These microphones are intended to record environmental noise levels and the noise reaching the ears in order to control daily noise exposure and prevent future hearing damage.

In addition to the microphones, the band features an accelerometer, labeled A in Figure 2.2. In addition there is haptic feedback, enhancing the user's interaction with the device. The band also includes a visual indicator, labeled V in Figure 2.2, that offers visual cues for battery status or operational modes, a data port, labeled D in Figure 2.2, for connectivity and firmware updates, and a rechargeable battery.

A key aspect of the band's design is its lightweight and unobtrusive nature, allowing for seamless integration into the user's daily routine without causing discomfort. Moreover, the use of patented acoustic filter technology in the earplugs guarantees sound clarity without the muffled effect that is common with traditional earplugs. This technology not only enhances situational awareness but also prevents the risk of overprotection, ensuring that users can remain alert to their surroundings while still protecting their hearing.

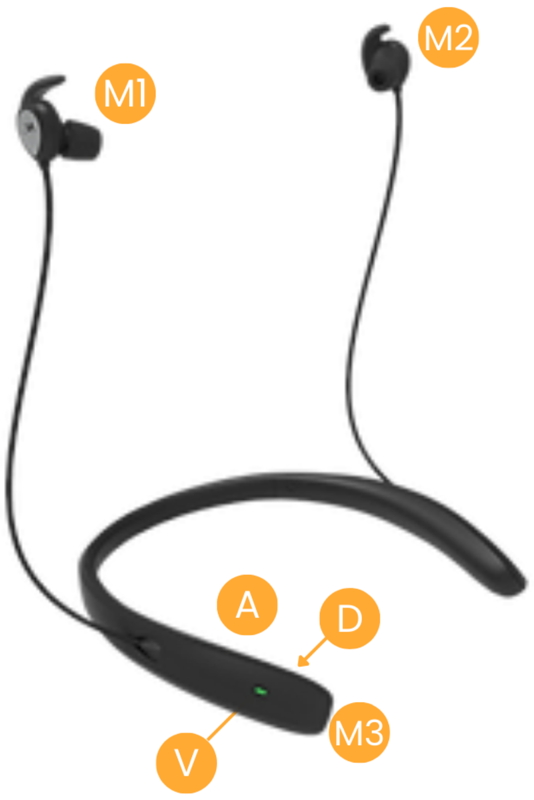


Figure 2.2: Earplug Specification

### 2.1.2 In-ear Detection

The incorporation of AI within this project is driven by the primary objective of accurately identifying the status of an earplug - whether it is inserted into the ear or not. This distinction is crucial because it dictates the internal processing of the data, which is critical to effectively ensuring the user's hearing safety. Wearing earplugs attenuates noise to a certain extent, and this is accounted for when assessing hearing safety and determining if noise levels surpass safe thresholds. Consequently, inaccurately indicating that earplugs are inserted when they are not can potentially introduce errors in data processing, potentially leading to inaccuracies in tracking the user's actual noise exposure.

The issue of in-ear detection has also appeared recently in the case of modern earbuds. It is increasingly used to enhance user experience by accurately determining whether the earbuds are in the user's ears and based on that manage audio playback more efficiently. This technology typically relies on either optical or physical sensors to determine the state of earbuds [46]. In the case of Smart Alert, though, it will be required to rely only on the microphones' data.

Each Smart Alert microphone captures data across five distinct frequency bands. At present, the status of the earplugs is assessed by comparing the readings from the microphones on the earplugs with those from the collar microphone, using predefined conditional expressions. However, implementing AI could provide a more sophisticated and accurate analysis of the data.

### 2.1.3 Data Structure

This section contains a description of the data under study, that is, the data collected from the earplugs. Data can be recorded on the device at varying intervals, with the units selected for this study logging data to a file every second. Given this frequent recording interval, the device can store approximately two hours of data before requiring upload to the cloud via the existing sync interface. It is essential to note that while the device logs noise data, it does not record actual sound.

Current processes allow the downloading of earplug data from the cloud in the form of a CSV file. The dataset consists of 31 features that represent the different information gathered by the device. The following paragraphs will describe the features of this dataset. The column descriptions will be organized into groups, linking related variables for a clear and concise presentation of the data.

**Timestamp** The Timestamp column stores the information about the exact date and time of each recorded observation. It is expressed in Unix timestamp format, which represents the number of seconds that have elapsed since midnight on the 1st of January, 1970 (UTC).

**Data from the microphones** The dataset contains data from three microphones: left, right, and collar. Each microphone provides 7 columns of data. In order to distinguish which microphone the data comes from, there are prefixes added to the feature name: „L" or „l" for the left microphone, „R" or „r" for the right microphone and „C" or „c" for the microphone located on the collar.

- **mic_leq_a**: This column represents the LAeq equivalent sound level during the logging period, measured in dBA. This value is derived from the current algorithm used to determine the status of the earplugs. Our analysis covers assessment of the earplug status so we need to use data that is independent from the current algorithm. Therefore, this column will not be used in our analysis.

- **mic_peak_c**: The column presenting peak dBC values from the microphone represents the maximum decibel level measured within a certain time interval.
- **Diag1, Diag2, Diag3, Diag4, Diag5**: These columns represent the dB values from the microphones int the five different frequency bands.

**Data from the accelometer** The device is equipped with an accelerometer, and its data is also provided in a csv file. However, the available information is limited because only the vector magnitude is stored, omitting data from all three axes of the accelerometer. Nevertheless, this data offers insight into the overall movement of the device. The dataset features two columns containing accelerometer data:

- **accel_mean**: It contains the average value of vector magnitude of the device's movement in a given time range.
- **accel_peak**: It contains the peak, maximum value of the device movement during this time.

**Other data** The dataset also includes several other values that are not directly relevant to the analysis conducted in this study. These values, primarily for internal use, provide additional information about the device and its environment. They include details such as the battery level of the device and system events that have occurred. While these data points may have significance in other contexts, they are not relevant to the specific analysis undertaken here.

## 2.2 AI Fundamentals and Classification Techniques

In today's technological landscape, AI is a frequently encountered term, representing efforts to equip machines with capabilities that mirror human cognition [55]. This expansive field covers various applications, from understanding languages to recognizing images, appealing to a broad audience with diverse levels of technical understanding. AI has grown in popularity and use across a multitude of sectors due to its versatility and effectiveness. Its methodologies encompass a range of techniques tailored to address diverse problems, demonstrating its transformative potential in solving complex challenges across industries.

### 2.2.1 Definitions and Core Concepts

To better understand the framework of AI, it is crucial to define several underlying concepts that support its complex structure. This section aims to lay the groundwork by outlining these key principles, allowing for a more in-depth examination of how they are applied in real-world scenarios.

#### Classification

In this study, our focus shifts to a specific computational challenge: classification. Classification is the task of identifying which category or class a new observation belongs to using a training set of data containing observations whose category is already known [24]. This involves organizing data into distinct categories based on recognizable patterns or attributes, a fundamental component in many AI applications where data categorization directly impacts functionality and outcomes.

#### Binary Classification

It is a specific type of classification where there are only two possible outcomes for each input [29]. An example pertinent to this thesis is determining the insertion status of earplugs, categorized simply as IN-EAR or OUT-OF-EAR. Binary classification simplifies decision-making processes in various applications, making it a widely used technique.

#### Machine Learning

Machine Learning (ML) is a subset of AI that focuses on developing algorithms capable of learning from and making predictions or decisions based on data. ML enables computers to perform specific tasks from data without using explicit instructions, relying instead on patterns and inference [24]. It encompasses a range of methods enabling machines to enhance their performance by learning from data over time. In the context of this study, where we are examining the categorization of data into distinct classes, ML is a more precise term that specifically addresses our case of enhancing algorithmic intelligence and functionality through experience and data. However, we will use the terms „AI" and „ML" interchangeably throughout this thesis to encompass the broader scope of AI technologies while focusing on ML applications.

#### Supervised Learning

This type of task, where models are trained on labeled data to categorize new inputs, is referred to as supervised learning. Supervised learning is a more precise term within ML, a subset of AI focused specifically on algorithms that improve automatically through experience. In supervised learning, each piece of training data includes an input and a corresponding output label, which the model uses to learn how to predict the correct label for new, unseen data [29].

#### Overfitting and Underfitting

These terms describe common challenges in ML model training. Overfitting occurs when a model learns too much detail from the training data, including noise, which harms its performance on new data. On the contrary, underfitting occurs when a model is overly simplistic, incapable of capturing the intrinsic patterns within the training data, thus displaying subpar performance even on its training dataset. Both have a negative impact on model accuracy, with overfitting due to excessive complexity and underfitting due to insufficient model complexity [24].

### 2.2.2 The Machine Learning Workflow

To effectively implement ML techniques, it is essential to understand the structured process that guides the development and deployment of models. The ML workflow encompasses a series of strategic steps, each critical for ensuring the accuracy, efficiency, and applicability of the solutions derived. This section outlines the key stages of this workflow, as illustrated in Figure 2.3 (adapted from [11]). However, despite the structured nature of the pipeline, the stages are often intertwined [11].

#### Data Acquisition

The Data Acquisition step involves gathering, loading, and integrating data to form a cohesive dataset [11], a foundational component in any ML project. In this case, it entails collecting labeled data from various users and earplug units, and integrating them into one dataset.

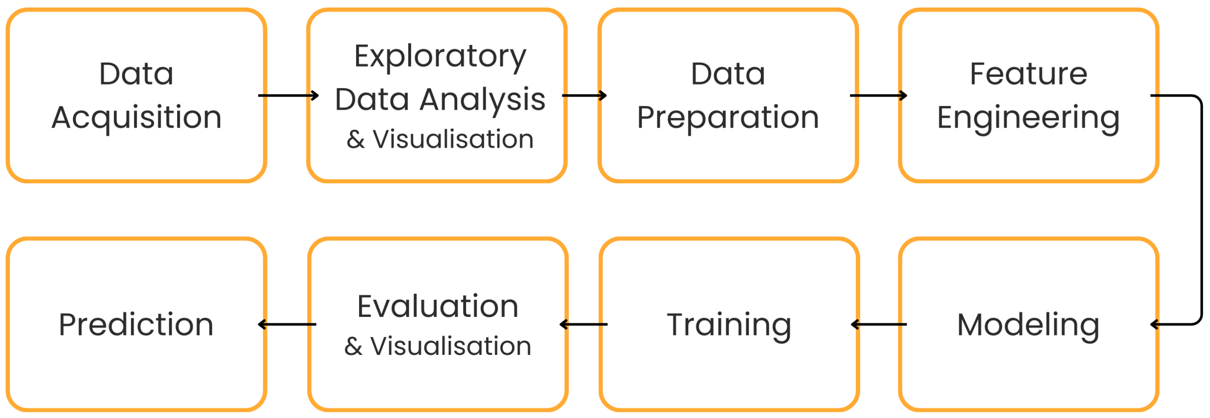


Figure 2.3: Machine Learning Flow

**Exploratory Data Analysis and Visualisation**

The Exploratory Data Analysis and Visualization step is instrumental in gaining insights into the data and facilitating model development. It is focusing on thorough data exploration to unveil hidden structures, detect anomalies, and validate hypotheses [77]. By computing feature importance and visualizing patterns, this step helps in understanding the dataset thoroughly. Visualizations can include histograms, scatter plots, and box plots to explore data distributions, relationships between variables, and potential outliers.

**Data Preparation**

The Data Preparation stage is responsible for converting raw data into a format that is suitable for analysis purposes. This could include tasks data transformation, cleaning data, and standardizing formats to ensure accuracy. Effective data preparation provides a reliable foundation for subsequent tasks [11].

**Feature engineering**

In the Feature Engineering stage the focus is on enhancing the quality and effectiveness of the feature set to improve model performance [11]. This involves a variety of techniques aimed at transforming and augmenting the original dataset to extract meaningful information and reduce computational complexity.

- **Creating New Features**: This technique involves generating additional columns based on existing ones to capture additional insights or enhance model performance. For example, creating interaction terms, polynomial features, or derived features based on domain knowledge can provide valuable information for modeling [85].

- **Encoding Categorical Variables**: Categorical variables, which represent qualitative data, often need to be converted into a numerical format for ML models to process them effectively. Techniques such as one-hot encoding, label encoding, and target encoding are commonly used to encode categorical variables while preserving their meaningful information [29].
- **Feature Scaling**: Feature scaling techniques are applied to ensure that all features have a similar scale, preventing certain features from dominating the model training process. Common scaling methods include standardization and normalization, which rescale features to a standard range [29].
- **Dimensionality Reduction**: High-dimensional datasets may suffer from the curse of dimensionality, leading to increased computational complexity and reduced model performance. Dimensionality reduction techniques such as Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE) [24, 29] are employed to reduce the number of features while preserving as much information as possible, thereby improving model efficiency and interpretability.

#### Modeling

The Modeling stage involves the construction and refinement of the models. During this phase, various modeling techniques and algorithms are explored, selected, and implemented to analyze the data effectively. Model building could encompass tasks such as planning the model architecture, selecting appropriate algorithms, and fine-tuning model parameters to optimize performance. Additionally, the modeling stage often iterates multiple times, allowing for the refinement and improvement of models through experimentation and evaluation.

#### Training

During the Training stage of the ML workflow, the focus moves to optimizing the model's performance using labeled data. The model learns from the provided data through iterative training, modifying its parameters to enhance the precision of its predictions.

**Evaluation and Visualization**

In the Evaluation and Visualization stage, the model performance is assessed using various evaluation metrics and visualization techniques. Understanding and using evaluation metrics enables informed decision-making in model selection, parameter tuning and overall model optimization.

One of the fundamental tools in this stage is the **confusion matrix**. It is often used for binary classification tasks. It presents a clear breakdown of model predictions versus actual class labels, with elements like true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) [21]. This matrix offers valuable insights into the performance of the model, especially in scenarios where class imbalances exist. The example of a two-dimensional confusion matrix is presented on Figure 2.4.

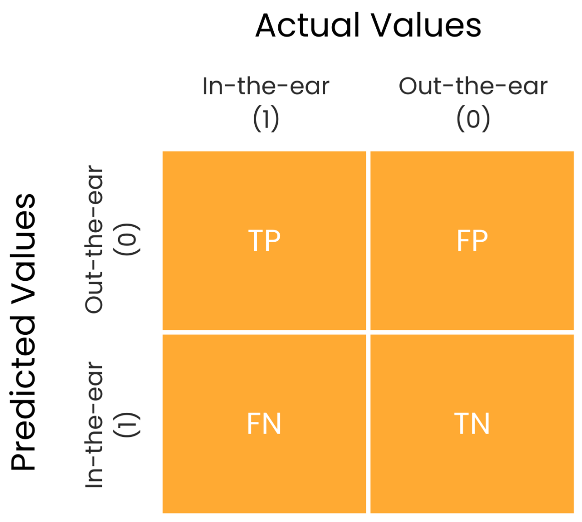


Figure 2.4: Two-Dimensional Classification Matrix

In addition to the confusion matrix, other commonly used evaluation metrics for the classification problems include:

- **Accuracy**: The accuracy score reflects the fraction of correct predictions made by a classification model. It measures how accurately the model's predictions match the true labels of the samples. If the predicted labels match the true labels for a sample, it contributes to the accuracy score, otherwise, it does not [60]. It can be calculated from the confusion matrix as follow:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} \quad [21]$$

- **Precision**: Precision is calculated as the ratio of true positives to the sum of true positives and false positives. It can be calculated from the confusion matrix as follow:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad [21]$$

- **Recall**: Recall, also known as sensitivity, is determined by dividing the number of true positives by the total number of actual positive samples (encompassing all samples that should have been accurately identified as positive). It can be calculated from the confusion matrix as follow:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad [21]$$

- **Specificity**: Specificity is determined by dividing the number of true negatives by the total number of actual negative samples (all samples that should have been identified as negative). It can be calculated from the confusion matrix as follow:

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \quad [21]$$

- **F1 Score**: It represents the harmonic mean of precision and recall, offering a balance between the two metrics. It is calculated as the ratio of twice the product of precision and recall to their sum. This metric ranges between 0 and 1, where a higher score indicates better model performance. It is especially useful in situations where there is an imbalance between the classes or when both false positives and false negatives need to be minimized simultaneously. It can be calculated from the confusion matrix as follow:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad [21]$$

There are situations where false negatives are more critical, the goal is then to minimize false negatives and prioritize sensitivity (which is true positive rate), whereas in scenarios where false positives are more tolerable, the focus lies on maximizing specificity (which is true negative rate). The confusion matrix helps with making informed decisions about the trade-offs between these metrics based on the specific requirements of the problem domain [61].

#### Prediction

In the Prediction stage of the ML workflow, the model's performance in making predictions in unseen data is assessed. After undergoing rigorous evaluation, the model is deployed to address the specific problem and evaluate its effectiveness.

### 2.2.3 Examining Classification Algorithms

In this section, we focus on straightforward and effective binary classification algorithms that are easy to implement and require minimal tuning. We emphasize methods that are well-documented and widely accessible, suitable for quick deployment across a range of applications. This approach ensures practical solutions can be quickly integrated and utilized effectively, without the complexity and extensive configuration often associated with more advanced models.

Following subsections will explore various binary classification algorithms, delving into how they function, their efficiency with different-sized datasets, and their adaptability to multi-dimensional environments. We will examine the preprocessing requirements essential for each algorithm to perform optimally. This detailed analysis aims to highlight the distinct features and operational frameworks of each method, providing a clear understanding of how they process and categorize data, which is crucial for applying these algorithms in real-world scenarios.

Additionally, each algorithm will be evaluated for its suitability in addressing a specific problem — detecting in-ear placement using dB values from multiple frequency bands. Particular attention will be given to the interpretability of the results provided by each model. This is vital for building trust and for the practical application of these algorithms, as decisions that are based on ML models need to be justifiable and comprehensible to end-users, particularly in domains where accuracy and reliability are critical.

#### k-Nearest Neighbors

The k-Nearest Neighbors (k-NN) algorithm is very popular method used in ML, particularly for classification tasks. It belongs to the category of instance-based methods, meaning it does not construct an explicit model but rather stores instances of the training set. When classifying a new observation, k-NN identifies the $k$ number of data points in the training dataset that is closest to it (hence the name *nearest neighbors*) and assigns the majority class among them to the new observation. The choice of the parameter $k$, representing the number of neighbors, is crucial and depends on the dataset characteristics. [66]

- **Handling Small/Big Datasets**: It can work effectively with different dataset sizes. While it doesn't necessarily require a lot of data, its performance generally improves with larger datasets due to the availability of more diverse neighbors for classification [39]. However, with small datasets, k-NN can still produce reliable results [28].

- **Handling Multiple Dimensions**: It operates effectively in low-dimensional spaces because the distance between points (used to determine nearest neighbors) is less likely to be distorted. As the number of dimensions increases, the distance metric may become less meaningful due to the curse of dimensionality, potentially leading to less accurate predictions [39].
- **Preprocessing Needs**: Preprocessing typically include feature scaling to ensure equal importance, handling missing values through imputation or exclusion, addressing outliers, and encoding categorical variables for numerical ones [41, 72], all aimed at optimizing distance calculations and enhancing the algorithm's performance.
- **Interpretability**: The decision-making process of k-NN is clear-cut: it examines the labels of neighboring data points to ascertain the majority class, resulting in easily understandable and transparent outcomes.
- **Suitability for in-ear Detection**: It can be suitable due to its simplicity and interpretability, although considering dimensionality reduction techniques could further optimize its performance given the modest dimensionality of our dataset.

#### Support Vector Classifier

The Support Vector Classifier (SVC) is another supervised learning classification method. Unlike k-NN algorithm, it is not instance-based. Instead, it seeks to find the optimal hyperplane that separates different classes in the feature space in a way to maximize the margin between them. It is useful for both binary and multi-class classification tasks. [68]

- **Handling Small/Big Datasets**: SVC is well-suited for small datasets, offering efficient training. However, it becomes less efficient as dataset sizes increase. [3]
- **Handling Multiple Dimensions**: SVC is highly effective in handling multi-dimensional data due to their ability to operate effectively in high-dimensional spaces, even when the number of dimensions exceeds the number of samples [68].

- **Preprocessing Needs**: SVC requires careful preprocessing which includes feature scaling [68] to ensure that the model isn't biased towards variables with larger scales to maintain the model's accuracy and robustness.
- **Interpretability**: The interpretability of SVC is generally limited[53], but using a linear kernel can yield coefficients [68] that clarify the importance of each feature in decision-making. For non-linear kernels, however, complex transformations obscure this direct understanding.
- **Suitability for in-ear Detection**: This algorithm is powerful and well-suited for in-ear detection task, as it can effectively handle small datasets by focusing on the most informative data points, optimizing performance even with limited data.

### Logistic Regression

Logistic Regression is a method commonly used for binary classification tasks, leveraging a logistic (sigmoid) function to convert regression outputs into probabilities constrained between 0 and 1 [84]. This transformation enables the prediction of class membership probabilities, making it particularly valuable in scenarios where outcomes are distinctly categorical. It is designed to provide outputs that are directly interpretable as probabilities, offering a solid foundation for decision-making based on the likelihood of different classifications.

- **Handling Small/Big Datasets**: It can be effectively applied to both small and large datasets. It generally requires a sufficient number of samples to ensure robust parameter estimates and to avoid overfitting, especially when dealing with multiple predictors [84].
- **Handling Multiple Dimensions**: Logistic Regression can be utilized for high-dimensional tasks, showing that it can handle datasets with a large number of attributes effectively [18]. However, it is crucial to ensure that the dataset has more observations than dimensions [42].
- **Preprocessing Needs**: It does not necessarily require preprocessing, but scaling can be helpful to ensure faster convergence and more stable optimization during model training [18].
- **Interpretability**: One of the major strengths of Logistic Regression is its interpretability. The model's coefficients offer insights into how predictor

variables influence the probability of class membership by indicating the change in the log odds of the dependent variable for a one-unit change in an independent variable [84].

- **Suitability for in-ear Detection**: It seems to be well suited for in-ear detection (which is binary classification problem) due to its ability to model the probability of class memberships. This makes it particularly useful in fields where probabilities need to be clear and interpretable.

### Decision Trees

Decision Trees (DTs) are a supervised ML method that splits data into branches to form a tree structure based on the purity of nodes, using criteria like information gain or the Gini index to choose attributes at each decision node [21]. Each node in the tree represents a decision that helps distinguish between the classifications, leading to a leaf node that provides the final outcome.

- **Handling Small/Big Datasets**: DTs generally perform well with both small and large datasets. However, with large datasets, they can become overly complex and are susceptible to overfitting, necessitating meticulous tuning. Conversely, with very small datasets, there is a risk that the trees might not capture complex patterns effectively [21].
- **Handling Multiple Dimensions**: DTs are prone to overfitting, particularly when confronted with datasets characterized by a substantial number of features compared to the number of samples. To mitigate this risk, it is crucial to maintain a balanced ratio of samples to features, and implementing dimensionality reduction techniques may prove beneficial [63].
- **Preprocessing Needs**: DTs require minimal data preparation. For instance, scaling of features is unnecessary, and their performance is not affected by outliers. Furthermore, certain decision tree algorithms can handle missing data effectively [62].
- **Interpretability**: A significant advantage of DTs is their high interpretability. They produce a clear and understandable model structure that can be easily visualized, facilitating straightforward explanations of the predictions made by the model [63].
- **Suitability for in-ear Detection**: This algorithm appear to be well-suited for in-ear detection. They can generally handle both large and small datasets

efficiently, do not necessitate extensive data preparation, and offer a clear, easily understood model structure, making them a practical choice for applications where understanding and interpreting the model's decisions is important.

**Random Forest**

The Random Forest (RF) algorithm is a powerful tool for solving classification problems. It works by constructing a collection of decision trees, each trained on a different subset of the data and using a random selection of features. By combining the predictions from all these trees, it produces a more accurate and robust final prediction. This approach helps to reduce overfitting and improves the model's ability to generalize to new, unseen yet data. [64]

- **Handling Small/Big Datasets**: RF handle both small and large datasets efficiently. They utilize bootstrap sampling from the training set to reduce variance and combat overfitting, making them robust even as datasets scale up [65], making this algorithm a better choice for large datasets in comparison to the DTs.

- **Handling Multiple Dimensions**: RF performs effectively in multidimensional cases by using a proximity measure that assesses how closely related samples are within the forest. This approach groups similar samples based on their paths through the ensemble of trees, providing a robust method for managing high-dimensional datasets [45].

- **Preprocessing Needs**: RF generally requires minimal preprocessing of data. It also maintains accuracy despite missing data and is resilient to outliers [21].

- **Interpretability**: The RF is easy to use, though analyzing it can be challenging [21]. Nevertheless, it offers valuable insights by evaluating the impact of each feature on the predictions through feature importance metrics [64].

- **Suitability for in-ear Detection**: RF is suitable for in-ear detection as it is straightforward to use, requires little preprocessing, and performs well even with large datasets. Although it generally achieves excellent performance, it can be somewhat difficult to interpret.

### Multi-layer Perceptron

The Multi-layer Perceptron (MLP) is a versatile neural network algorithm used for classification tasks. It consists of multiple layers of interconnected nodes, including an input layer, hidden layers, and an output layer. During training, the MLP adjusts the connection weights between nodes to minimize a specified loss function, allowing it to learn complex patterns in the data. In classification tasks, the output layer typically has nodes corresponding to each class, with a softmax activation function to produce class probabilities. [67]

- **Handling Small/Big Datasets**: The MLP is capable of performing with various sizes of training sets, effectively handling both small and large datasets [7].
- **Handling Multiple Dimensions**: MLPs can handle multi-dimensional input data, which is necessary for applications where inputs are continuous and multi-dimensional [7].
- **Preprocessing Needs**: MLP models can greatly benefit from standardization. It ensures that each feature contributes equally to the model's calculations by scaling the data uniformly and prevents any single feature from dominating the training process [52].
- **Interpretability**: he interpretability of MLPs is generally considered low, especially when compared to methods like decision trees. MLPs, with their complex network structures and weight adjustments, do not easily lend themselves to intuitive understanding or visualization of how decisions are made [7].
- **Suitability for in-ear Detection**: MLP models are commonly utilized for binary classification tasks in experimental setups due to their versatility and effectiveness so might potentially perform very well for in-ear detection problem. However, their complex architectures and non-linear nature often make them challenging to interpret easily.

## 2.2.4 Tools and Technologies

This section will focus on fundamental tools and technologies that are vital for ML workflows. From core development environments to specialized frameworks and visualization libraries, each component is considered integral for robust research and innovation. A comprehensive review of available technologies will be provided, offering insights into their functionalities and applications in ML.

### Programming Language

Programming languages form the backbone of ML development, providing the necessary tools and frameworks to implement algorithms and build predictive models. In this subsection, we explore some of the key programming languages commonly used in the field of ML, drawing insights from a comparative analysis [34] of Python, Java, and R.

- **Python** [71]: Python can be characterized by user-friendly syntax, extensive range of libraries tailored for tasks such as ML and data analysis, and widespread adoption within the research community.
- **Java** [5]: With a focus on reliability, security, and platform independence, Java finds its niche in enterprise-level solutions and Android app development. Its robust object-oriented paradigm and extensive community support make it suitable for building scalable ML applications.
- **R** [69]: Specifically designed for statistical computing and data analysis, R offers a comprehensive suite of tools and packages tailored for ML tasks. Its extensive collection of statistical algorithms and visualization features renders it the preferred option for statisticians and researchers operating within the ML domain.

In summary, Python, Java, and R are among the leading programming languages utilized in ML development, each bringing unique strengths and catering to diverse use cases. Python's simplicity and extensive library support make it the preferred choice for most ML practitioners, while Java and R continue to play significant roles in specific domains and applications within the field [34].

### Frameworks and Libraries

This section investigates the essential frameworks and libraries used to facilitate ML development. With so many possibilities accessible, the focus is focused to Python, the dominating programming language in the ML field.

- **Data Manipulation**: Data manipulation libraries play a crucial role in ML development, enabling efficient preprocessing and manipulation of datasets. These libraries provide essential tools for tasks as data cleaning, transformation, and analysis.
  - **Pandas** [49]: It is an open-source data manipulation and analysis tool in Python, well-suited for handling large data sets with its DataFrame structures for tabular data. It excels in tasks like data cleaning, manipulation, and reading various file formats.
  - **NumPy** [23]: It is a core library for numerical computing in Python, providing efficient operations on large, multi-dimensional arrays and matrices. It offers extensive mathematical functions, speed, and memory efficiency, which are essential for high-performance scientific computing.
- **Machine Learning Libraries**: ML libraries provide a wide range of algorithms and tools for building, training, and evaluating ML models. These libraries streamline the development process by offering implementations of various algorithms, as well as utilities for data preprocessing or model evaluation.
  - **Scikit-learn** [58]: It offers a wide array of algorithms, simplifies complex tasks and grants access to diverse ML methods. This versatility makes it an indispensable tool for exploring and deploying advanced techniques in research projects.
  - **TensorFlow** [1]: TensorFlow is a powerful open-source ML library developed by Google Brain. It provides comprehensive support for deep learning and neural network models, along with tools for deployment across various platforms.
  - **PyTorch** [57]: It is an open-source deep learning framework developed by Facebook's AI Research lab. PyTorch offers dynamic computational graphs, making it flexible and suitable for research experimentation.
  - **Keras** [15]: Keras is a user-friendly, high-level neural networks API written in Python. Its design prioritizes simplicity and modularity, facilitating rapid experimentation with deep learning models. Keras allows for rapid prototyping of neural networks and can run seamlessly on top of TensorFlow.
- **Visualization Libraries**: Data visualization enables gaining insights from data and communicate findings effectively. Visualizing data allows for the exploration of patterns, trends, and relationships, facilitating the understanding

of complex datasets. Some popular visualization libraries that provide tools for creating insightful plots and charts are:

- **Matplotlib** [26]: A versatile library for creating static and interactive visualizations in Python, offering a wide range of plotting functions.
- **Seaborn** [81]: Built on Matplotlib, Seaborn simplifies statistical data visualization with high-level functions for creating attractive and informative plots.
- **Plotly** [27]: An interactive visualization library supporting static and dynamic plotting. It enables the creation of interactive charts, dashboards, and web applications with support for various chart types and cross-platform compatibility.

### Integrated Development Environments

An Integrated Development Environment (IDE) is a software application that provides programmers with tools for software development, including a source code editor, build automation, and debugging capabilities, all within a single interface. IDEs supporting ML development integrate built-in ML libraries, interactive notebooks, and data visualization tools, streamlining the process of experimenting with ML algorithms and analyzing results.

- **Jupyter Notebook**[37]: Jupyter Notebook stands is a vital tool for AI experimentation, offering an interactive space for data analysis and collaboration. Its compatibility with the Python language enables the effortless exploration and application of various AI techniques. With its user-friendly interface and multimedia features, experimentation, analysis, and dissemination of findings are simplified.

- **Visual Studio Code** [51]: Visual Studio Code is a lightweight yet powerful source code editor developed by Microsoft. It supports various programming languages and provides features such as debugging, syntax highlighting, and intelligent code completion, making it a popular choice for ML development.

- **PyCharm**[32]: PyCharm is a robust IDE specifically designed for Python development. It offers a wide range of features tailored to Python programming, including code analysis, debugging, and version control integration. PyCharm's intelligent code editor and extensive plugin ecosystem make it well-suited for ML projects.

## 2.3 Practical Aspects of AI Adoption

In the realm of software, operations entail a diverse set of activities crucial for the development, deployment, and maintenance of software systems. These tasks, spanning infrastructure management, environment configuration, and performance monitoring, are essential for the smooth operation of software applications.
With organizations relying heavily on software to support their operations, the effectiveness of these operational practices is key to delivering value to customers and stakeholders.

### 2.3.1 AI Implementation Challenges in Industry

Integrating AI solutions into industrial operations presents numerous organizational, technological, and ethical challenges. Drawing on current literature, this section identifies the key obstacles to implementing AI technologies in real-world industrial settings.

#### Organizational Challenges

Organizational factors are one of the most common types of issues. These include managerial, cultural, and strategic obstacles that could hinder the successful implementation of AI. Challenges include a lack of understanding of AI's business potential [10, 25, 79], inadequate top-level support [4, 10, 44, 78], and uncertainty about return on investment [10, 30, 35, 74].

#### Data Challenges

The second category is the Data Challenges. They were mentioned most frequently in the literature. AI needs a huge amount of data, so ensuring the quality, quantity, and accessibility of data is paramount for effective AI implementation. These challenges encompass various aspects, including the lack of quality and volume of available data [4, 10, 22, 30, 35, 36, 38, 74, 79], data governance issues [4, 10, 38, 74], security risks [10, 22, 30, 36, 74], acquiring, processing, and linking data effectively [30, 36, 74], data latency and transport issues [30], model training and testing difficulties due to data shortage and bias [36, 38, 74], and challenges in model interpretation and deriving business value [4, 10, 30, 35].

### Technology Challenges

Integrating AI into industrial applications presents several technological challenges. Managing complex AI and ML models requires sophisticated version control systems for data, code, and hyperparameters [33, 47, 48, 73], and the need for computationally intensive hardware like GPUs [4, 47]. Continuous monitoring is crucial to detect and swiftly address biases or adversarial attacks, ensuring the integrity and performance of AI applications [22, 30, 33, 38, 48, 75]. Additionally, the rapid deployment of AI models into existing workflows necessitates an adaptable infrastructure that supports frequent updates without disrupting operations [10, 22, 33, 36, 40, 70, 74].

### Ethical and Legal Challenges

Finally, in the last category, Ethical and Legal Issues, the analysis underscores the critical importance of addressing ethical and legal considerations in AI adoption. These encompass various concerns, including transparency and fairness in AI adoption [4, 22, 35, 38, 79, 83], as well as the immaturity of the legal environment and the potential for legal and regulatory violations [4, 38, 74, 79]. Furthermore, ensuring privacy, security, consent, and bias considerations in data usage is paramount, alongside concerns about the suitability of AI technology for the industry and adherence to laws and regulations [4, 22, 35, 38].

## 2.3.2 Machine Learning Operations

Developing and deploying ML models efficiently and effectively can be a challenge. Traditional software development practices often fall short when applied to ML workflows due to the unique challenges posed by the iterative nature of model development, the complexity of managing data and models, and the necessity for rapid adaptation to new data. This has spurred the development of practices that cater specifically to the needs of ML projects.

### Popular Methodologies in Modern Software Development

Several methodologies have revolutionized modern software development, each bringing unique principles that enhance various aspects of the software development lifecycle. These methodologies include:

- **Agile**: Agile practices promote an iterative and incremental development process, enabling teams to respond to changes quickly and deliver work in small, manageable increments [8]. This flexibility helps maintain a focus on delivering value to customers through continuous feedback and adjustment.
- **Lean**: Originating from manufacturing principles, the Lean methodology emphasizes generating greater value for customers while utilizing fewer resources. This includes minimizing unnecessary tasks and optimizing efficient practices [59].
- **DevOps**: This methodology integrates software development (Dev) and IT operations (Ops), emphasizing automation, continuous integration (CI), and continuous delivery (CD) to speed up and enhance the reliability of software deployment [20].

Together, these methodologies create a robust environment for developing, testing, and releasing software that better aligns with business goals and customer needs. They provide a solid foundation for developing and deploying ML models.

#### MLOps

A specialized framework, known as Machine Learning Operations (MLOps), has been conceptualized to address the specific needs of ML production processes. Emerging from the integration of traditional DevOps practices with the distinctive demands of ML, MLOps is positioned at the crucial intersection of three key areas: data engineering, ML, and DevOps, as illustrated in the Figure 2.5. Each of the areas is responsible for different tasks within the realm of ML models development and deployment:

- **Data Engineering**: Effective handling of data through meticulous collection, analysis, and visualization.
- **Machine Learning**: Efficient management of model development, feature engineering, and parameter tuning.
- **DevOps**: Adaptation of continuous integration, automated testing, and performance monitoring specifically tailored for ML workflows.

Despite promising performances in the experimental stage, most models fail to progress due to the complexities of deployment and maintenance, leading

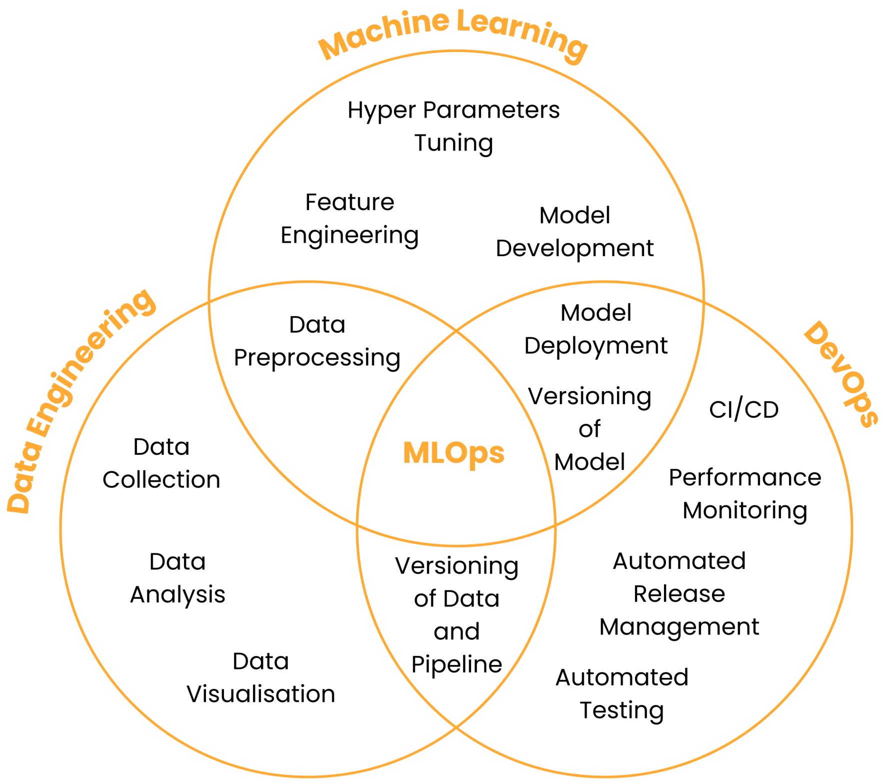


Figure 2.5: Machine Learning Operations Overview

to stagnation at the prototype phase [2]. MLOps offers automated processes and structured frameworks to overcome these obstacles, enabling organizations to experiment with AI technologies more effectively and drive innovation.

### 2.3.3 MLOps Tools and Technologies

MLOps is characterized by the integration of ML with operational practices to facilitate the deployment and maintenance of ML models in production environments. This section is dedicated to the exploration of examples of tools and technologies that underpin successful MLOps implementations. Covered are aspects of data engineering, ML development, and DevOps practices.

#### Data Engineering Tools and Technologies

The goal of data engineering is to prepare and manage data to ensure its quality, availability, and consistency for use in ML applications. This section examines examples of essential tools and technologies found in the literature that facilitate data collection and integration, storage and management, and versioning. These processes are critical for ensuring that data is reliable and accessible, supporting robust MLOps framework. By exploring how each category of tools contributes to effective data engineering, we provide a foundation for understanding the technologies that enhance and streamline the handling of data in ML deployments.

- **Data Collection and Integration**: Data collection can involve gathering information from diverse sources which may include databases, sensors, or web APIs, to create comprehensive datasets used in ML. Effective integration is crucial, particularly when these sources provide varied formats and data types, requiring sophisticated tools to consolidate and normalize data into a unified format. This ensures that the datasets are consistent and usable for analysis. Tools like Apache Kafka [40] and RabbitMQ [9] , which facilitates real-time data streaming, and Apache NiFi [56], known for its ability to automate and manage data flows across multiple systems, are commonly employed to support these tasks.

- **Data Storage and Management**: Data Storage and Management in ML integrates various technologies to handle large and diverse datasets efficiently. Distributed file systems like Apache Hadoop manage vast volumes, while databases such as SQL and NoSQL—including key-value, columnar, document, and graph stores—offer flexible storage solutions tailored to specific data types and queries [17]. Querying platforms like Hive, Impala, Spark SQL, and Drill enhance data retrieval, supported by cloud platforms such as Amazon S3, Microsoft Azure, and OpenStack, which provide scalable storage options [17]. Additionally, fog computing extends processing capabilities closer to data sources, improving response times and efficiency for ML projects [17].

- **Data Versioning**: Data versioning is crucial for ensuring the reproducibility and maintainability of ML models. Among the most popular tools, DVC (Data Version Control) and Delta Lake stand out as open-source options that provide robust solutions for managing and tracking changes in data and models [75]. Additionally, LakeFS offers similar capabilities with a focus on ease of merging and branching, akin to traditional software version control systems [75]. On the private side, Pachyderm offers an integrated platform that includes data versioning, catering to enterprises needing comprehensive data management systems [75].

### Machine Learning Tools and Technologies

ML Tools form a critical component of MLOps, providing the necessary functionalities for building, training, and evaluating models. These tools range from libraries and frameworks that facilitate the construction of ML algorithms to platforms that assist in model training and hyperparameter tuning.

- **Frameworks and Libraries**: Frameworks and libraries in ML serve the crucial goal of providing foundational tools and pre-built algorithms that facilitate the development and training of ML models. These resources were previously explored in detail in section 2.2.4, Tools and Technologies. Among the most notable are Scikit-learn, TensorFlow, and PyTorch.
- **Hyperparameter Tuning**: Effective tuning of hyperparameters can significantly enhance model performance. Tools that automate this process, such as HyperOpt, SigOpt, and Optuna [75], allow practitioners to explore more parameter combinations faster than manual tuning.

### DevOps Tools and Technologies

DevOps tools and technologies streamline the continuous integration, delivery, and deployment of ML models into production environments. These tools enhance collaboration between development and operations teams and ensure that ML models are deployed efficiently and reliably.

- **Continuous Integration and Continuous Delivery (CI/CD)**: CI/CD automates the integration of code changes from multiple developers and their deployment to production environments. This reduces errors and accelerates deployment cycles, allowing for quicker feedback and more iterative updates. Tools like Jenkins, GitLab CI/CD, and GitHub Actions are instrumental in managing these processes efficiently [40, 70].
- **Containerization and Orchestration**: Containerization encapsulates an application and its dependencies into a container, ensuring consistent execution across any infrastructure. Orchestration manages these containers across different environments, ensuring they scale and operate efficiently. Docker is a popular choice for containerization, while Kubernetes is extensively used for orchestrating containers [40, 47, 70, 75, 82].

- **Infrastructure as Code (IaC)**: IaC manages and provisions the infrastructure through code rather than manual processes, enhancing both productivity and consistency across environments. This approach allows for the automated setup of servers, databases, networks, and other infrastructure elements needed for ML deployments. Tools such as Terraform and AWS CloudFormation enable teams to define their infrastructure in scripts which can be versioned and reused [16, 47, 80].
- **Configuration Management**: Configuration management tools help maintain consistency of the application's operational environment, ensuring that software performs as expected on all systems. This is crucial for ML applications where consistency across data processing and model training environments is essential. Ansible, Puppet, and Chef are popular choices that automate the software configuration process across various environments [47, 50, 80].
- **Monitoring and Logging**: Effective monitoring and logging are critical for maintaining the health and performance of ML models once they are in production. These tools help identify and diagnose issues in real-time, ensuring high availability and performance. Prometheus is commonly used for monitoring, while the ELK Stack (Elasticsearch, Logstash, and Kibana) is utilized for powerful logging and visualization capabilities [40].

## 2.4 Summary

This chapter has laid a comprehensive foundation for the thesis, encompassing essential domain knowledge in the field of hearing protection, AI, and cloud computing principles. It has explored the specifics of the Smart Alert system and the in-ear detection problem, demonstrating the potential of ML to enhance detection accuracy and ensure user safety. Additionally, this chapter addressed the challenges of integrating AI into practical applications and highlighted the significance of MLOps in managing the lifecycle of AI models efficiently. Moving forward, the next chapter will delve into the design and implementation of the system, detailing its architecture and the methodologies used for pilot development.

# Chapter 3

# Design and Implementation

This chapter serves as a framework outlining the systematic approach taken to achieve the defined goals which include designing and implementing a system to explore the potential of AI algorithms for hearing-protecting equipment. A comprehensive strategy was established at the beginning to streamline the research process and facilitate progress monitoring.

The methodology is based on a project plan divided into four distinct stages, each strategically designed to achieve specific objectives critical to meeting the research goals. A visual representation of the plan is presented in Figure 3.6. These stages encompass a wide range of activities, including:

1. **Initial Analysis**: It is the initial stage of the research, which consists of understanding the data, defining the problem, and aligning project objectives with company needs through in-depth discussions with company experts.

2. **Data Acquisition**: The main goal of this stage is to design and implement the process for generating, labeling, storing, and processing the required data.

3. **Iterative Pilot Development**: This stage involves setting up the infrastructure to enable AI experiments and putting in place an iterative pilot development process to enable gaining hands-on experience with AI and a better opportunity to determine the potential challenges, costs and benefits from introducing AI into the company's production environment.

4. **Evaluation**: It is the final stage of the project which consists of evaluating the created pilot version, along with identifying the potentially existing obstacles and proposing ways to overcome them.

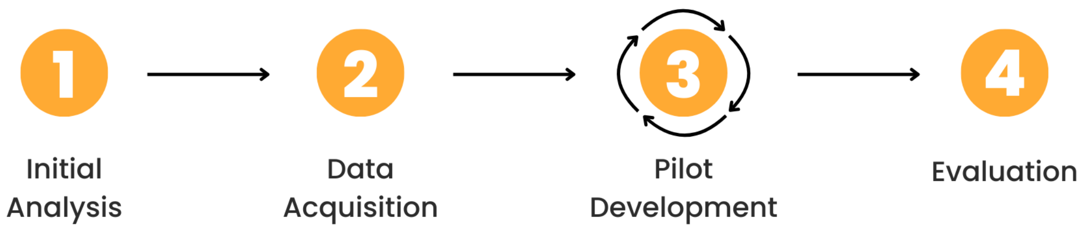


Figure 3.1: Stages of the Project Plan

The subsequent sections will elaborate on each stage of the project plan, identifying the expected outcomes and the necessary activities, tools and technologies for each phase of the research process.

## 3.1 Initial Analysis

The project begins with an initial analysis phase, where the primary focus is on understanding the data, comprehensively grasping the problem, and engaging in discussions with experts from the company. This stage serves as a project kickoff, facilitating a mutual understanding of expectations and objectives. It involves in-depth conversations with company specialists to ensure alignment of the project's direction with the company's needs and to gain insights into the specific challenges and requirements of the AI application in hearing protection.

### 3.1.1 Data Assessment

Data is the cornerstone of this AI project, with its quality directly influencing the models' effectiveness. While there is a substantial amount of data collected from earplugs, the usability of this data for the project is limited because it includes labels indicating whether an earplug is IN-EAR or OUT-OF-EAR, generated by an existing algorithm. The project aims to develop a new algorithm and will not use these existing labels, focusing instead on creating a system that can generate more accurate and reliable labels. Furthermore, since this data originates from clients, there are significant privacy and permission constraints preventing its use for this project. Therefore, there is a need to establish a new system that can gather and label data independently, ensuring the labels are accurate and the data is collected in compliance with privacy standards. This approach will facilitate the development of more advanced and reliable AI models by providing clean, well-labeled, and usable datasets.

### 3.1.2 Clarifying Objectives

In the Initial Analysis phase of the project, setting clear and achievable objectives is paramount for guiding the development and ensuring the project aligns with the strategic goals. This section outlines the main goals of the project that were agreed to with the company.

**Primary Objective** The primary objective of this thesis is to develop a comprehensive system designed to facilitate the experimentation with AI algorithms and establish a robust process for the collection of accurately labeled data. This system aims to:

- **Enable AI Experimentation**: Provide a structured environment where different AI algorithms can be tested and analyzed to determine their suitability and effectiveness for specific tasks within the project.
- **Streamline Data Collection**: Implement a methodical approach for gathering high-quality, accurately labeled data, which is critical for training and validating AI models. This involves designing user interfaces (UI) and backend processes that integrate seamlessly with existing workflows, ensuring that data collection is both efficient and scalable.

**Secondary Focus** In addition to setting up a dynamic framework for AI experimentation and data collection, there is a significant secondary focus on the actual creation and evaluation of models using available open-source algorithms. This aspect of the project will involve:

- **Utilizing Open-Source Libraries**: Leverage existing implementations of ML algorithms from open-source libraries, which offer a wide range of robust tools for AI development.
- **Model Development and Evaluation**: Use of available algorithms, adjustment of them and evaluation of their performance specifically for ear detection tasks. This includes thorough testing of model accuracy, precision and recall to identify the most effective algorithms.

- **Parameter Optimization**: Explore different parameter settings and feature engineering techniques to optimize the models. This iterative process aims to enhance model performance, adapting to the nuanced requirements of in-ear detection, and potentially extending the models' applicability to broader scenarios.

### 3.1.3 Domain Experts Consultation

Throughout the development of the thesis, there will be ongoing consultations with domain experts from the company. These consultations are designed to ensure that the project remains aligned with the operational realities and technical requirements of the earplug data systems. Experts will provide insights into the nuances of earplug data, help refine problem definitions, and validate the practicality of the proposed solutions. Their input will be crucial in guiding the development process, ensuring the solutions developed are not only theoretically sound but also practically viable.

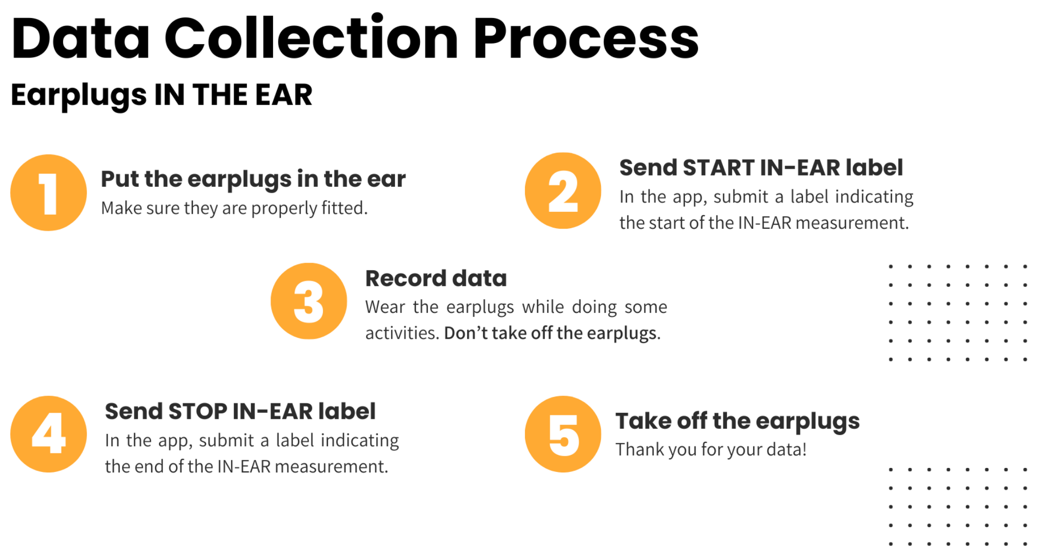
## 3.2 Data Acquisition

Data is essential for the implementation of AI. While the company already has some data storage mechanisms in place, the unique nature of the problem, requiring existence of reliable labels of the measurement, demands the collection of new data. It is important to ensure that the data collected is relevant and of high quality as it can directly impact the performance of the AI model. The data needs to be labeled accurately to train the model effectively.

### 3.2.1 Overview

The specific challenge in collecting data for the project lies in the necessity for reliable labels that clearly indicate the status of the earplugs. It was decided that the data would be collected by company employees and that integrating this process into existing workflows could help streamline data collection and ensure a steady supply of data for the project.

To facilitate the data collection process, an additional menu tab has been implemented in the Smart Alert application to enable the earplug data synchronization and sending labels to the system. The features have been made available and visible only to Minuendo employees. Several employees expressed their willingness to be volunteers and assist in collecting the labeled data. They were provided with a purpose-built multimedia presentation explaining the data collection procedure. Figure 3.2 displays a slide from the presentation that outlines the procedure for recording data with earplugs in place. It details the steps from fitting the earplugs correctly to submitting start and stop labels through the app during an activity session, concluding with the removal of the earplugs after data collection.



Data Collection Process

Earplugs IN THE EAR

1 **Put the earplugs in the ear** Make sure they are properly fitted.

2 **Send START IN-EAR label** In the app, submit a label indicating the start of the IN-EAR measurement.

3 **Record data** Wear the earplugs while doing some activities. **Don't take off the earplugs.**

4 **Send STOP IN-EAR label** In the app, submit a label indicating the end of the IN-EAR measurement.

5 **Take off the earplugs** Thank you for your data!

Figure 3.2: Slide Illustrating In-Ear Data Collection Steps

### 3.2.2 Integration with Smart Alert Application

The Smart Alert application is an integral part of the product's system, extensively used by employees. The application is a web application built using Flutter, which allows it to operate efficiently on browsers both on computers and mobile platforms.

There are two main functionalities relevant for data gathering process:

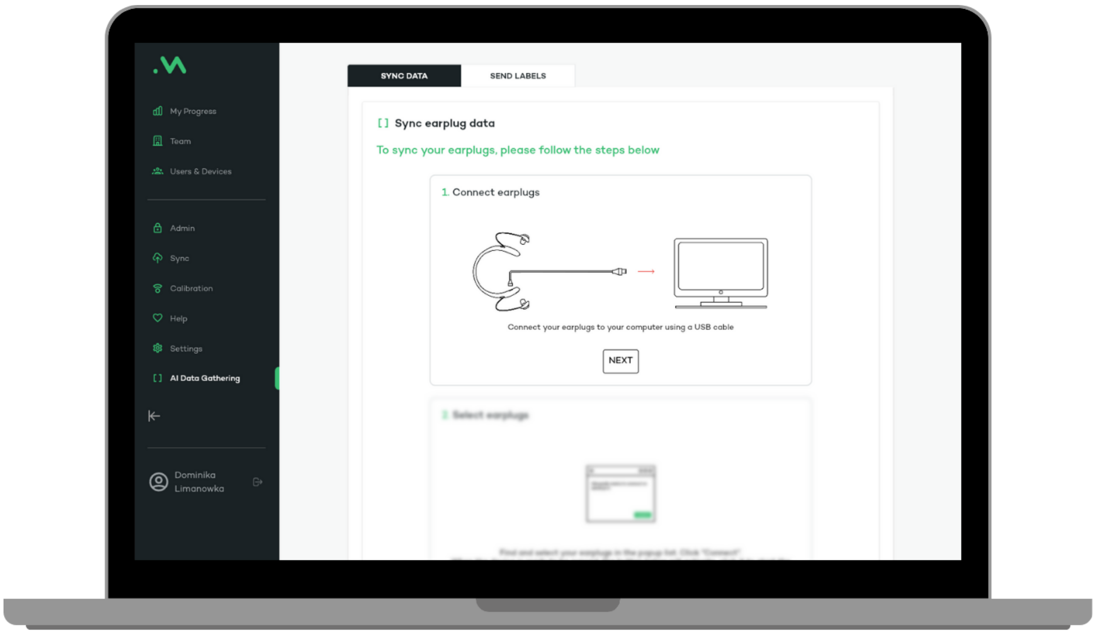
1. **Data Syncing**: The application already includes a data syncing feature that is crucial for the data gathering process. This feature requires the use of a computer as it involves uploading earplug data through a USB connection directly into the system.

2. **Labelling Interface**: A new labeling feature is planned to be developed and integrated into the application. This feature will allow employees to label the

state of earplugs as IN-EAR or OUT-OF-EAR at precise moments. The interface will be designed to be user-friendly and intuitive, merging smoothly with the existing application to ensure ease of use without adding complexity.

### 3.2.3 Earplug Data Synchronization

This functionality was based on the existing interface for synchronizing plug data available in the production version of the Smart Alert application. This screen provides instructions to guide the user through the whole process. To synchronize data, users need to connect the earplugs to a computer and select the appropriate device in the serial port window to validate the connection. Then click the „START" button to start transferring data. When the process, illustrated by a progress bar, is complete, a message is displayed saying whether the data synchronization was successful - then the device can be unplugged. A screenshot showing the data synchronization interface is shown in Figure 3.3.



[ ] Sync earplug data

To sync your earplugs, please follow the steps below

1. Connect earplugs

Connect your earplugs to your computer using a USB cable

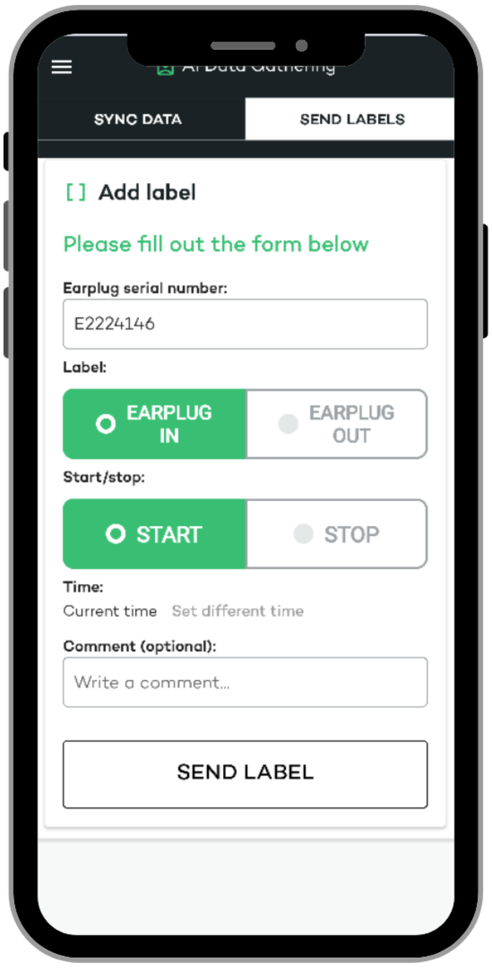
Figure 3.3: Interface for Uploading Earplugs Data to the Cloud

### 3.2.4 Sending Labels to the System

A new user-friendly interface has been developed to streamline the data labeling process, featuring a simple label upload form accessible from both computers and mobile devices. A screenshot showing the sending labels interface is shown in Figure

3.4. The interface has been designed for ease of use, ensuring seamless labeling even while performing manual tasks. Key elements of the form include:

- A text field for inputting the device's serial number, pre-filled with the user's assigned device number to eliminate need of manual entry.
- A label button indicating whether the earplugs are in or out of the ear.
- START/STOP buttons for marking the beginning and end of each measurement.
- An optional text field for users to add comments about the measurement.
- An optional button for adjusting the measurement time, with default settings of current time.



Earplug serial number: E2224146

Label: [✓ EARPLUG IN] EARPLUG OUT

Start/stop: [✓ START] STOP

Time: Current time Set different time

Comment (optional): Write a comment...

Figure 3.4: Interface for Data Labeling

### 3.2.5 Obtaining the Data

The data collected from the Smart Alert application is directly transmitted to the Minuendo server, where it could be exported as CSV files for each measurement. The CSV files include standard columns for this data, as described in section 2.1.3 Data Structure, along with four additional columns:

- **labelId** - This column contains information regarding the ID of the label assigned to each measurement.
- **serial** - This column provides details about the serial number of the device from which the measurement originates.
- **userId** - This column includes the ID number of the user who submitted the measurement.
- **inear** - The „inear" column serves as a crucial in this case label, specifying whether the earplugs were inserted (1) or removed (0) from the ears during the measurement.

## 3.3 Pilot Implementation

The iterative Pilot Development stage marks an important phase in the project. It focuses on establishing the necessary infrastructure to facilitate AI experiments and piloting a development process aimed at gaining practical experience with AI technologies for in-ear detection cases. By following this structured approach, the Pilot Implementation stage lays the groundwork for iterative experimentation and refinement, ultimately paving the way for the successful integration of AI technologies.

### 3.3.1 System Overview

This section provides a comprehensive overview of this project stage, highlighting its key components. It covers the system objectives, provides a description of the general design and infrastructure, and introduces the cloud environment utilized.

### System Objectives

The system's key objective is to prioritize simplicity, ensuring that chosen solutions are straightforward and easy to use and facilitates quick experimentation with ML algorithms. While the focus is on experimentation with ML, developing a full deployment system is not required. However, integrating certain MLOps practices, such as automation, is essential to enhance the system's efficiency.

Additionally, the system will include a data management platform to streamline working with acquired CSV data, enhancing the overall workflow. Experiments with ML algorithms will follow an iterative development approach, underscoring the necessity for version control for both code and data to ensure reproducibility across experiments. This systematic approach aims to provide a robust foundation for conducting ML experiments efficiently and effectively within the context of in-ear detection research.

### System Design

The project's infrastructure comprises a well-organized network of servers, each serving a distinct purpose to support the various aspects of the AI development process and data management. Figure 3.5 shows a system architecture diagram, which includes the key components and interactions that exist between them. The diagram is organized into several key areas:

- **Master Server**: The Master Server acts as the command center for the entire infrastructure setup. It hosts the Terraform and Puppet configuration files, making it the primary point for orchestrating and initializing the infrastructure. This server is essential for deploying and managing all other servers in the system, ensuring that configurations are consistently applied and maintained.
- **Central Data Management Repository**: The repository contains all the files needed to deploy the Central Data Repository Management Tool, including the SQL database init schemas, API code in Python, Dockerfile for building the API image, and a docker-compose file for orchestrating the deployment.
- **Central Data Repository Server**: The Central Data Repository Server is dedicated to storing all CSV data files. It operates a custom-made Docker-based Data Repository Management Tool system with an API and a database. The API provides an endpoint for uploading data files, which analyzes each

sent file, stores its metadata in the database, and saves the file in a dedicated Docker volume. Both the database and CSV file volumes are mounted directly on the server to ensure data persistence and resilience. The docker compose is deployed on the server using the GitLab CI/CD pipeline to automate the process.

- **In-ear Detection Repository**: The In-ear Detection Repository houses Jupyter notebook for ML experiments and leverages DVC for tracking training and validation datasets.

- **Developer Server**: The Developer Server hosts the Jupyter environment, maintains a clone of the in-ear detection repository, and synchronizes files from the Central Data Repository Server. It is also integrated with a DVC remote storage server, facilitating access to all necessary data for ML experiments.

- **DVC Remote Server**: It serves as a remote storage for datasets versioned by DVC, providing a centralized location for storing and retrieving various versions of the datasets.

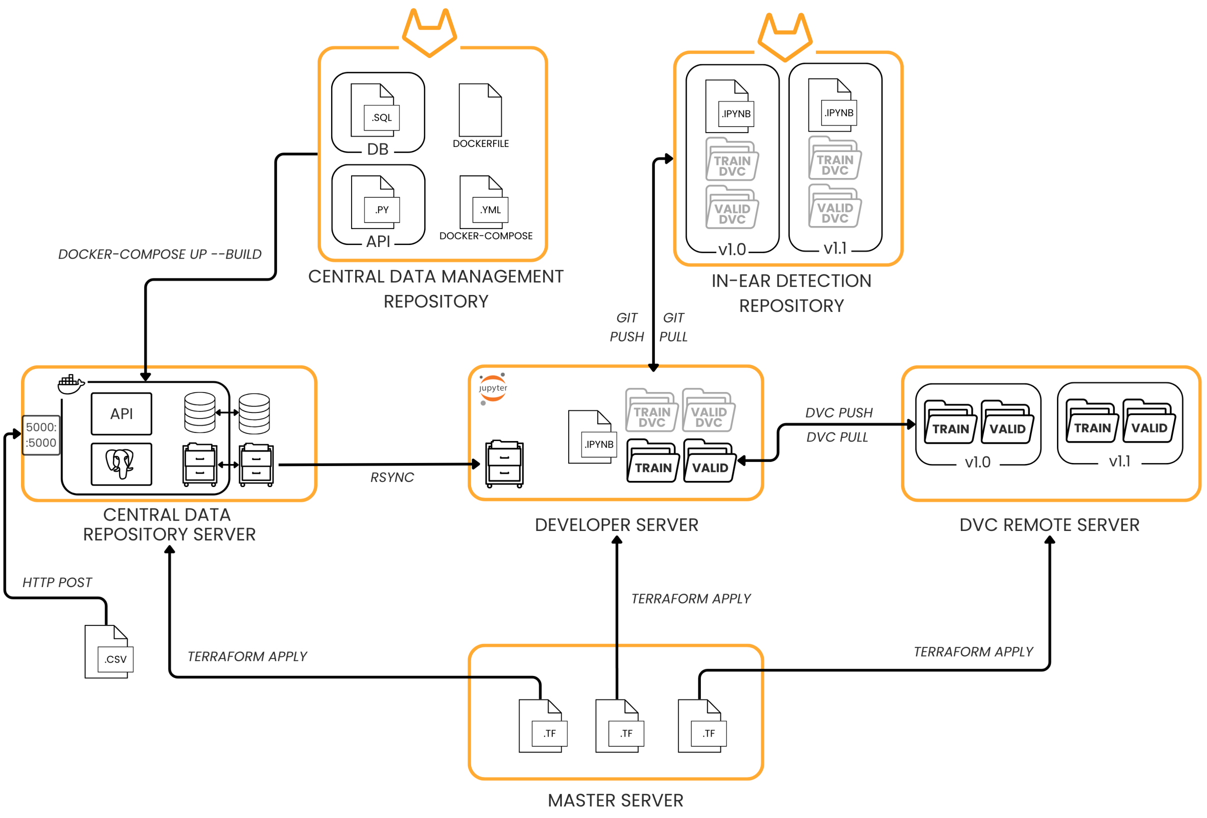


Figure 3.5: Detailed System Architecture Diagram

### Cloud Environment

All work for the ML project is conducted on the university's servers, which are part of a secure cloud infrastructure based on OpenStack and exclusively accessible within the university's network. Access to the server is regulated through SSH key authentication, ensuring only authorized individuals, such as myself, can access and manage the data. For consistency and reliability across all instances created for the project, Ubuntu 20.04 was chosen as the source image.

### Backup

The OsloMet's cloud solution utilizes Ceph storage technology, which provides built-in redundancy and fault tolerance. This ensures data resilience and reliability, with automated backups safeguarding against data loss or corruption. However, the architecture employed by the university involves storing all data in a singular location which could lead to a potential risk of losing the data in case of any on-site incident.

To address these concerns and ensure that the data and configurations remain safe and recoverable, a system has been devised to regularly back up important data to an external location. This system integrates a scheduled synchronization process, utilizing rsync, a powerful file-copying tool, which is particularly efficient in transferring and synchronizing files across multiple systems over SSH. The choice of rsync for this task stems from its ability to efficiently handle large datasets by transferring only the changes made to the data, thereby reducing bandwidth usage and improving backup times.

The backup mechanism involves secure SSH communications, enabled through key-based authentication, eliminating the need for passwords and securing data transfers. Backups are automated through a cron job scheduled to run daily at 2 AM, optimizing performance by operating during off-peak hours.

```
# Run the backup script daily at 2 AM and save the logs into log file
0 2 * * * /home/user/backup_script.sh >> /home/user/backup_log.log 2>&1
```

Listing 3.1: Setting up CRON job

```
#!/bin/bash
# Backup CSV Data
rsync -avzhe ssh central_data_repo:/srv/data/backup_directory
# Backup PostgreSQL Database
```

```
rsync -avzhe ssh central_data_repo:/srv/postgres/backup_directory
# Backup Developer Repository
rsync -avzhe ssh dev_server:/home/dev/inearaidetection/backup_directory
# Backup DVC Remote Server
rsync -avzhe ssh dvc_server:/srv/dvc/data/backup_directory
```

Listing 3.2: Backup script

### 3.3.2 Central Data Repository

This section introduces the purpose and fundamental operation of the Central Data Repository server. It outlines the server's role as the primary storage location for all CSV files used within the project. This server stores raw data files and optimizes their accessibility through an effective data management strategy.

The data is maintained in CSV format and is uploaded to the Centralized Data Repository Server through a dedicated API endpoint. This method facilitates ease of data management while ensuring data integrity. Each file's upload triggers the storage of relevant metadata in a PostgreSQL database, which includes details such as file name, earplug serial number, the date range covered by the file, upload timestamp, and file size. The complete source code for all components described below is provided in the appendix.

#### API

The server is equipped with a Flask-based API, created to facilitate efficient data handling and management. To enhance deployment consistency and streamline environment management, this API is fully dockerized, encapsulating its dependencies and runtime environment within Docker containers. This setup ensures that the API can be deployed uniformly across different development and production environments without compatibility issues. The API implements following endpoints:

- File Upload Endpoint (POST): This endpoint handles the uploading of CSV files. Upon receiving a file, the endpoint first validates the file format and then utilizes the Pandas library, a powerful tool for data analysis in Python, to parse and extract necessary information from the file. This extracted data includes key metrics and identifiers which are essential for data categorization and retrieval.

Following the analysis, the relevant data is stored in the database, with connection parameters securely sourced from environment variables provided by Docker. This ensures that each interaction is secure and contextually configured to its deployment environment.

- Data Retrieval Endpoint (GET): This endpoint allows querying files based on specific criteria such as measurement or upload date, earplug's serial number, label and more.
- System Health Check Endpoint (GET): This endpoint could be used for maintaining API reliability by continuously monitoring the health of the API, providing essential diagnostics and ensuring that the system remains fully operational without disruptions.

**Database**

The metadata extracted by the API is stored in a PostgreSQL database. This setup is implemented within a Docker container to leverage the benefits of containerization, such as portability and consistency across different environments. The database contains a single table, csv_metadata, which is structured to store comprehensive metadata for each CSV file uploaded. The schema for this table is defined on Listing 3.3. It contains a range of data about each file:

- file_name: The name of the file, ensuring no duplicates.
- label: A boolean that indicates whether the measurement was classified as IN-EAR or OUT-OF-EAR.
- label_id, earplug_id, user_id: Identifiers linking the file to specific labels, earplug devices, and users, facilitating detailed tracking and analysis.
- timestamp_start and timestamp_end: Timestamps defining the data coverage period within each file.
- file_path: The storage path of the file on the server, ensuring each file's location is uniquely identified.
- file_length: The amount of rows in the CSV file.
- upload_date: The timestamp marking when the file was uploaded to the system.

```
CREATE TABLE IF NOT EXISTS csv_metadata (
    id SERIAL PRIMARY KEY,
    file_name VARCHAR(50) NOT NULL UNIQUE,
    label BOOLEAN NOT NULL,
    label_id INT NOT NULL UNIQUE,
    earplug_id VARCHAR(8) NOT NULL,
    user_id INT,
    timestamp_start TIMESTAMP NOT NULL,
    timestamp_end TIMESTAMP NOT NULL,
    file_path TEXT NOT NULL UNIQUE,
    file_length INT NOT NULL,
    upload_date TIMESTAMP NOT NULL
);
```

Listing 3.3: Database table definition

### Containerization and Volume Management

As previously mentioned, our system employs Docker to streamline the deployment and management of our services. This containerization technology enables us to encapsulate the API within a Docker container, making it straightforward to expose the API on a specified port. Docker Compose orchestrates this setup through configurations specified in the docker-compose.yaml file, which outlines the necessary services, networks, and, importantly, volumes for data persistence. It is presented on the Listing 3.4.

```
version: '3.8'
services:
  api:
    build: .
    volumes:
      - .:/app
      - /srv/data:/api/uploads
    ports:
      - "${API_PORT}:5000"
    depends_on:
      - db
    environment:
```

```
    - DATABASE_URL=postgres://${POSTGRES_USER}:${POSTGRES_PASSWORD}@db/$
       {POSTGRES_DB}
 db:
   image: postgres:latest
   environment:
     POSTGRES_DB: ${POSTGRES_DB}
     POSTGRES_USER: ${POSTGRES_USER}
     POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
   volumes:
     - ./db/init:/docker-entrypoint-initdb.d
     - /srv/postgres:/var/lib/postgresql/data
   ports:
     - "${DB_PORT}:5432"
```

Listing 3.4: Configuration of docker compose

This setup utilizes Docker volumes for storing uploaded CSV files and PostgreSQL database files, ensuring continuity and integrity of data across the lifecycle of containers. Mounting these volumes directly on the host system enables seamless backups and secure data access during updates and routine maintenance. This strategic use of Docker not only simplifies the deployment process but also enhances the application's overall security and scalability.

**Application Deployment**

Application deployment for the Central Data Repository server involved a process facilitated by GitLab CI/CD. The repository containing the API code, database initialization scripts, Docker Compose configuration, and Dockerfile was stored on GitLab. The deployment process was orchestrated using GitLab CI/CD pipelines configured to automate the deployment workflow.

In the deployment workflow, a simple pipeline was created with a single step to manage the deployment process, as shown in Listing 3.5. This snippet initiates the deployment by bringing down existing containers and rebuilding the application using Docker Compose. While this pipeline step serves as a foundational element, it can be extended with additional steps such as integration testing, security checks, artifact publishing, and rollback mechanisms to enhance reliability and efficiency.

```
deploy_step:
  stage: deploy
```

```
script:
  - echo "Starting deployment..."
  - docker-compose down
  - docker-compose up --build -d
```

Listing 3.5: Deployment Step in GitLab CI/CD Pipeline for Central Data Repository

**Technology Selection Justification**

Selecting the appropriate technologies was paramount, with a focus on balancing performance, flexibility, and ease of implementation. Below, the rationale behind the choices are presented and some proposed alternatives are outlined.

- **API**: For the API development, the Flask framework was selected. It was chosen for its simplicity and flexibility in building lightweight web applications. Alternatives could include the Django REST framework for more complex projects requiring additional features and built-in functionalities.
- **Database**: For the database, PostgreSQL was selected. It offers robust features, reliability, and adherence to SQL standards, making it suitable for handling relational data. However, since the database consists only of one table, alternatives such as SQLite could be considered for simpler implementations due to its lightweight nature and ease of use or any NoSQL databases.
- **Containerization**: Docker was chosen for containerization. It provides a standardized way to package and deploy applications, ensuring consistency across different environments. Although alternatives like Kubernetes offer advanced orchestration capabilities, it was decided that they might be too complicated for this case, and Docker alone would suffice for our needs.

### 3.3.3 Developer Server

The Developer Server is a crucial element of the system infrastructure, specifically designed to facilitate the development and testing of AI models for in-ear detection. The server provides an environment equipped with the necessary tools and resources, such as Jupyter Notebooks and necessary libraries. It hosts a version-controlled repository cloned from GitLab, which contains all necessary scripts and notebooks for developing and refining AI algorithms.

The Developer Server is configured to sync seamlessly with the Central Data Repository Server, ensuring access to the most up-to-date datasets for training and testing. This configuration allows developers to selectively include specific datasets in their work by placing them in the data directories within the repository. These directories, while not tracked by Git, are managed with Data Version Control (DVC) to facilitate the efficient handling of large datasets and to maintain a clear historical record of data used in various experiments. This setup optimizes the development process, allowing for robust testing and iterative improvements of AI models while ensuring data consistency and reproducibility in experiments.

**In-ear Detection Repository**

The project repository is structured to support efficient development and version control of used data, adhering to the goal of transparency and reproducibility. The repository is organized as follows:

- **/dataset**: This directory contains the training data files, which are essential for developing the ML models. Managed by DVC, changes in these data files are meticulously tracked without bloating the Git repository with large files.
- **/validation**: Holds the validation data files used for testing and evaluating the models. Like the training data, modifications to validation data are tracked using DVC to ensure precise version control.
- **pilot_development.ipynb**: A Jupyter notebook that encapsulates the entire process of data exploration, model training, and evaluation. This notebook serves as the central document for explaining the methodology and results.
- **dataset.dvc** and **validation.dvc**: DVC tracker files for the training and validation datasets. These files store metadata about the datasets' versions, facilitating robust data versioning.
- **.dvc/**: A directory automatically generated by DVC, containing metadata and configuration details crucial for dataset tracking and project reproducibility.
- **.DVC/config**: Contains DVC configuration settings that link the project with remote storage and define data management workflows.
- **.gitignore**: Specifies intentionally untracked files that Git should ignore, including DVC-managed data files and other non-source code elements.

This organization ensures that each component of the ML workflow is clearly defined and easily accessible, promoting an efficient and transparent development process. The complete source code for all components of this repository is available in the appendix.

### In-ear Detection Development

Development is conducted within Jupyter notebooks, chosen for their versatility in allowing extensive room for experimentation and iterative refinement. These interactive environments are ideal for exploring various hypotheses, visualizing data, and incrementally building models.

The main notebook guides the development of AI models for in-ear detection and begins by importing essential Python libraries and modules, setting up configurations like file paths. Functionalities are encapsulated into reusable functions to keep the code organized and maintainable.

The remainder of the notebook aligns with the steps defined in the In-ear Detection Pilot Methodology, described later in Section 3.3.6. It consists of data aggregation, where multiple CSV files are merged into a single dataframe for comprehensive analysis. This stage sets the foundation for the subsequent data handling processes, which include splitting the data, applying optional scaling, and conducting feature engineering.

Model training follows these preparatory steps, involving selecting and tuning algorithms, evaluating their performance on a validation set, and visualizing the results to assess predictive accuracy. This structured approach ensures that each model development phase is methodically executed to maintain consistency and reliability.

### Data Management

Due to the iterative nature of developing AI models for ear-in detection, keeping track of changes in the dataset is not just useful but essential. Although it is recognized that storing large datasets directly in the repository alongside the code is not a best practice, it remains important to track dataset versions and match them with specific model iterations for reproducibility and systematic improvement.

To efficiently manage this challenge, it was decided to utilize DVC. DVC allows versioning datasets without bloating the code repository and linking specific datasets to model iterations. This ensures clarity on which model was trained with which version of the dataset, a critical factor for reproducibility. This approach streamlines the development, testing, and, in the future, deploying of AI models, fully adapting to the demands of iterative model development.

Despite the advantages of using DVC, managing it alongside regular Git operations can be cumbersome, requiring commands such as *dvc push/pull* every time changes need to be synced with the remote data storage. To streamline this process and reduce the overhead of managing both Git and DVC commands separately, scripts named *full_checkout* and *full_push* were created. These scripts consolidate Git and DVC operations into single commands, thereby simplifying the version control workflow and ensuring that dataset and code versions remain synchronized effortlessly.

These scripts were added to the *.bashrc* file on the development server to enhance usability, allowing them to be used as commands directly from the terminal. This integration facilitates seamless execution of version control tasks within the development environment. The code for these scripts is provided in the appendix for detailed reference.

**Technology Selection Justification**

This section briefly overviews the justifications for choosing specific technologies used on the Developer Server, emphasizing how they contribute to the project's objectives and enhance the development environment.

- **IDE**: Jupyter Notebooks were utilized as the IDE due to their interactivity, ease of use, and widespread adoption within the data science community. Their unique capability to combine code, output, visualization, and narrative into a single document streamlines the process of exploratory analysis and iterative model development. Popular alternatives to Jupyter for interactive development include RStudio for development with R programming language, and Visual Studio Code, which supports numerous programming languages and extensions tailored for data science workflows.

- **Data Versioning** DVC was chosen for its efficiency in managing large datasets and maintaining reproducibility without enlarging the Git repository. It mimics Git commands, simplifying the learning curve for users already familiar with version control systems, and allows for external storage of large files while keeping track of their versions. A popular alternative for handling data versioning is MLFlow, which offers a broader range of features but is generally more complex compared to DVC.

### 3.3.4 DVC Remote Server

The DVC Remote Server functions as a dedicated storage and management system for datasets used in projects versioned by DVC. It primarily stores the actual data files, where each file is identified by a unique hash that acts like a fingerprint of the file's content. This hashing mechanism ensures that every version of a file is stored distinctly, allowing for precise retrieval based on its version. Additionally, the server maintains metadata associated with these files, which includes versioning details that correlate directly with the revisions of the project in the DVC system, much like commits in a Git repository. This setup is crucial for tracking the evolution of data alongside code changes, ensuring consistency and reproducibility in data-driven projects.

#### Alternative Storage Options

While the project employs an on-site hosting solution with SSH for security and control, other viable alternatives include cloud solutions like AWS S3, Azure Blob Storage, and Google Cloud Storage. These platforms offer scalable and secure data storage options. Google Drive is another alternative, offering quick setup and easy access.

### 3.3.5 Master Server

The Master Server in this project architecture plays a pivotal role in orchestrating the setup and management of the entire network of servers used for the AI development. This server acts as the central command point from which all other servers are provisioned and managed. The following sections outline the key functionalities of this server.

**Infrastructure and Configuration Management**

Centralized around the Master Server, the infrastructure setup and configuration management utilize Terraform and Puppet. These tools automate the provisioning and configuration of the servers, streamlining the deployment process and enhancing the system's reliability and scalability. The Master Server oversees the setup of components such as the Central Data Repository, Developer Server and DVC Remote Storage Server.

Creating a strategic plan for the deployment and management of servers was a critical step in the infrastructure setup and configuration management. This plan involved defining specific deployment requirements for each server to ensure that they are optimized for their respective roles within the system. Below are the tailored requirements essential for the functionality of each server in the setup:

- **Central Data Repository Server**:
  - Must have Docker and Docker Compose installed to manage containerized applications.
  - Requires a GitLab Runner installed and running to handle CI/CD processes, allowing for automated deployment of the application.
- **Developer Server**:
  - Requires installation of DVC for data version control, Jupyter for interactive development, and Python for running the application.
  - Must be provisioned with custom scripts *full_checkout* and *full_push* to simplify DVC and Git operations. These scripts should be added to the *.bashrc* file to make them available as commands directly from the command line.
- **DVC Remote Storage Server**:
  - Needs to have a dedicated user account set up for managing data versions with DVC.
  - An *authorized_keys* file must be created in the *.ssh* directory to manage SSH access. This file will store the public SSH keys of servers that are authorized to use this storage, ensuring secure and controlled access to the data.

The deployment and configuration of these servers were largely automated using Terraform and Puppet, with the complete code for these processes available in the appendix. However, it's important to note that the automation was not all-encompassing. Manual interventions were necessary for tasks such as the registration of the CI/CD GitLab Runner and the management of SSH keys, which are essential for maintaining secure and authorized access to GitLab repositories. These manual steps are crucial for ensuring the security and integrity of the system.

#### Potential Features and Extensions

While the current setup effectively supports the core functionalities, future enhancements could include integrated monitoring and alert systems. Incorporating tools like Prometheus for monitoring and Grafana for analytics could provide deeper insights into system performance and help preemptively address potential issues.

#### Technology Selection Justification

Other options such as Ansible and Chef were considered for infrastructure automation and configuration. However, Terraform and Puppet were ultimately recognized as a sufficient and well-suited fit for the project's needs. These tools provided the necessary capabilities for robust and scalable infrastructure management, aligning effectively with the project's specific requirements.

### 3.3.6 In-ear Detection Pilot Methodology

The methodology for the in-ear detection pilot is designed to ensure a robust and flexible approach to developing and validating ML models for earplug detection. This process is inherently iterative, allowing for continuous refinement and adaptation in response to experimental insights and evolving data patterns. Each iteration of the in-ear detection pilot methodology has been presented on Figure 3.6 and consists of the following steps:

- **Data Acquisition**: This step involves selecting which files to include in the dataset to ensure it is diverse and representative for robust model development.
- **Exploratory Data Analysis and Visualization**: Conducts initial data analysis to uncover patterns and identify potential issues, using visualization tools for

clarity.

- **Data Preparation and Feature Engineering**: Describes necessary preprocessing activities, such as scaling features, to prepare the data for modeling. Additional techniques may be applied to enhance model relevance.
- **Modeling and Training**: Discusses the setup and adjustment of AI models, including algorithm selection and parameter configuration, followed by the training process.
- **Evaluation and Visualization**: Assesses model performance using metrics like Accuracy, Precision, Recall, Specificity, and F1 Score, with visual aids to illustrate results from test and validation sets.

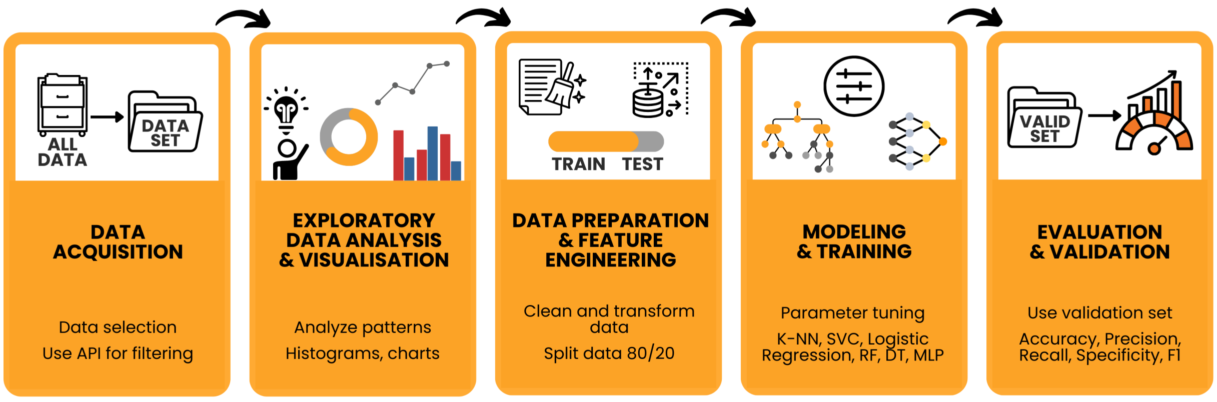


Figure 3.6: Stages in the Pilot Development Methodology

### Iterative Development Process

The iterative nature of this project facilitates flexibility in exploring various hypotheses and adapting methodologies based on intermediate results. This dynamic approach supports the progressive refinement of models and techniques, enhancing the overall effectiveness of the AI systems developed. Each iteration involves revisiting the data handling, feature engineering, model training, and evaluation stages, enabling systematic improvements and fine-tuning of the algorithms.

### Data Acquisition

The process of setting up the data for training involves strategically placing the selected files into the *dataset* directory. This directory serves as the primary data source for training the ML models. To facilitate the selection of suitable data, details about the available files can be retrieved using the GET API endpoint provided by the central data repository. This step is crucial as the choice of data significantly impacts the model's performance; a dataset that is too small or lacks diversity can adversely affect the outcomes of the models, leading to less robust predictions. Thus, ensuring a well-curated and representative dataset is essential for effective model training.

#### Exploratory Data Analysis and Visualization

During the exploratory data analysis phase, various methods are employed to scrutinize and understand the dataset thoroughly. Examples of techniques used include:

- **Histograms**: These are used to visualize the distribution of variables within the dataset, which helps understand the spread and skewness of data points.
- **Distribution of Labels**: Analyzing how data is distributed across different categories can highlight imbalances or peculiarities that might affect model training.
- **Printing Statistics**: Summarizing data through descriptive statistics provides foundational insights into the central tendencies and variability of the data.

#### Data Preparation & Feature Engineering

This step involves describing the essential preprocessing activities to ready the data for modeling. It can include cleaning to remove duplicates and correct errors, along with scaling to standardize feature ranges. Specific iterations might also employ additional techniques to enhance model relevance and efficiency. Additionally, data is split into training and testing sets using an 80/20 ratio. This setup allows for testing the models on new, unseen data using performance metrics to assess their effectiveness.

#### Modeling and Training

In the model selection and training step, six algorithms were identified and selected for their suitability in handling binary classification tasks, specifically tailored for in-ear detection. These algorithms include k-NN, SVC (with a linear kernel chosen for its ease of interpretation), DT, RF, Logistic Regression, and MLP. Each algorithm has its unique advantages and capabilities, making them strong candidates for effectively determining whether earplugs are correctly in place. The applicability of these algorithms to the problem of in-ear detection was thoroughly discussed in Section 2.2.3 where their relevance and potential benefits for this specific application were detailed. This diverse selection was driven by the objective to explore various approaches in the initial experimentation phases of the project, allowing for a comprehensive evaluation of different methodologies to address the in-ear classification challenge.

#### Evaluation and Visualization

This step involves assessing the performance metrics such as Accuracy, Precision, Recall, Specificity, and F1 Score for each model using test sets and validation sets, supported by visual aids to illustrate the models' performance. Assessing these quantitative performance metrics allows for the comparison of results across different model iterations. The validation set, comprising 9 files not used in the training set and chosen to represent diverse conditions, serves to mimic real-life predictions. This evaluation on unseen data provides a measure of the models' generalizability and effectiveness.

#### Technology Selection Justification

This section outlines the rationale for selecting specific technologies for the In-ear Detection Pilot. This justification encompasses the choice of programming language, ML library, and visualization tools, which together form the essential technical framework for the project.

- **Programming Language**: Python emerged as the preferred choice over highly-statistical R or Java. Its simplicity and robust libraries facilitate rapid prototyping and experimentation, crucial for iterative model development and testing.

- **ML Library**: Scikit-learn was selected for this pilot project due to its extensive documentation and comprehensive functionality, making it well-suited for rapid development. Despite its relative simplicity, it offered all the necessary functionalities required for the pilot's experimentation phase

- **Visualisation Libraries**: Due to their simplicity and popularity, it was decided to use Matplotlib and Seaborn for data visualization in the project. Their widespread usage and rich functionality enable quick creation of clear and attractive plots. However, Plotly provides interactive and web-based plotting capabilities, making it very compelling alternative.

## 3.4 Summary

This chapter has outlined the comprehensive project plan, encompassing initial analysis, data acquisition, pilot implementation, and evaluation, providing detailed insights into each phase. It delved into the architecture of the proposed system, explaining the rationale behind technology choices and detailing the methodology used for pilot development, which explores the potential of ML algorithms for in-ear detection. The next chapter will present the results of these efforts, evaluating the effectiveness of the system and analysing the pilot development outcomes.

# Chapter 4

# Results

This chapter provides a comprehensive analysis of the results from the project aimed at facilitating the introduction of AI for in-ear detection problem. The primary goal was to develop and assess a system capable of integrating AI effectively. The evaluation is structured to first consider the overall system, focusing on data gathering processes, infrastructure setup, and testing of system components. Following this foundational assessment, the chapter delves into the pilot iterations analysis, detailing the developmental progress and enhancements made to the AI models.

## 4.1 System Evaluation

The first part of this chapter is dedicated to examining the outcomes of the primary objective of this thesis, which was to design and implement a system that enables experimentation with AI technologies in a real-world setting.

### 4.1.1 Data Gathering and Labeling Process

The initial phase of this project emphasized the establishment of a rigorous data collection and annotation system, critical for ensuring the integrity and quality of the data necessary for training the ML models. In total, approximately 258 MB of data were collected, comprising 62691 rows of measurements.

### User Interface for Data Labeling

The UI for data labeling was designed to be user-friendly to facilitate efficient and accurate data collection. Recognizing the importance of simplicity and accessibility, the initial UI was adapted to include larger buttons that were easier to click, especially for users who might be working in challenging conditions or wearing gloves.

This design choice was directly influenced by user feedback, which indicated that the original smaller radio buttons were too difficult to navigate quickly during data collection sessions. The feedback highlighted the need for clearer and more accessible labeling options to reduce errors and streamline the data entry process. The updated UI with larger, distinct buttons for different data states such as IN-EAR and OUT-OF-EAR improved the overall impression of the data collection process. Figure 4.1 below illustrates both the initial design with small radio buttons and the subsequent redesign featuring larger, easy-to-click buttons. This comparison highlights the improvements made to facilitate a more user-friendly and efficient data collection process.

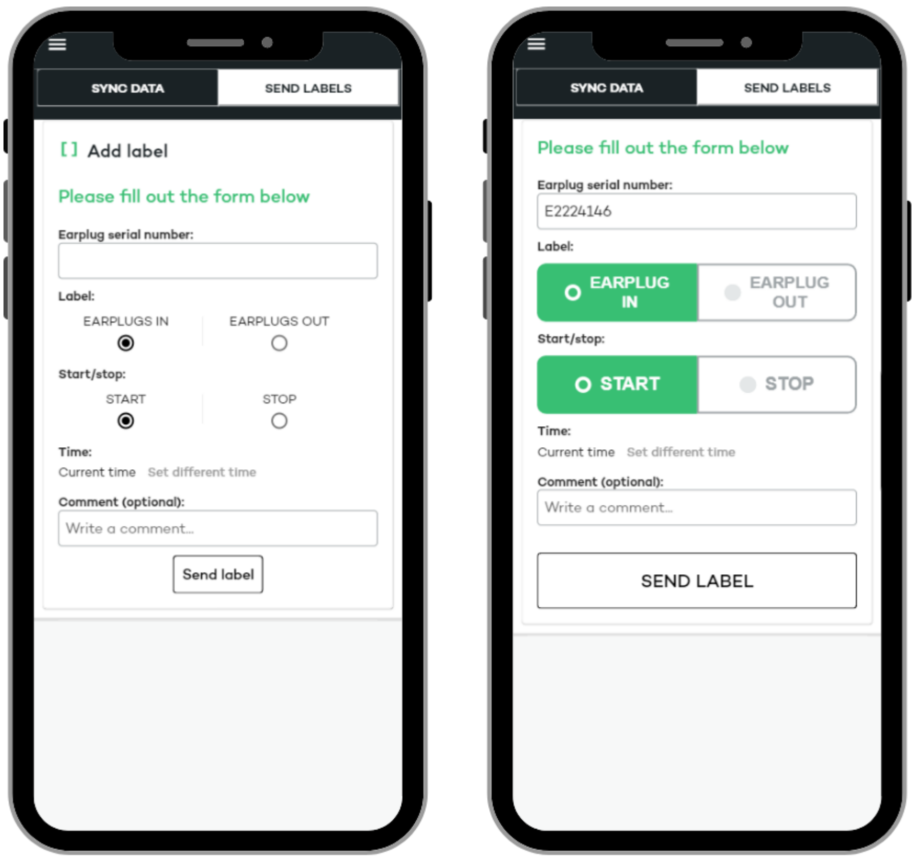


Figure 4.1: Data Labeling Interface: Before and After User Feedback

**Data Labelling Process**

The data collection process depended on users to mark start and stop of each recording session. They had to clearly indicate when data recording began and ended, ensuring the accuracy of the collected data. For instance, when recording IN-EAR data, users sent start measurement label with IN-EAR value when they put in the earplugs and sent stop label before taking them out. This strict adherence to procedures aimed to prevent any mixing or mislabeling of data, maintaining the reliability of the collected information.

Throughout the project, volunteers from the company provided valuable feedback on potential ways to simplify the data collection process. One significant suggestion was to automatically end recording sessions when starting new measurements, like transitioning from IN-EAR to OUT-OF-EAR. This proposal aimed to make operations smoother and improve user experience during data collection.

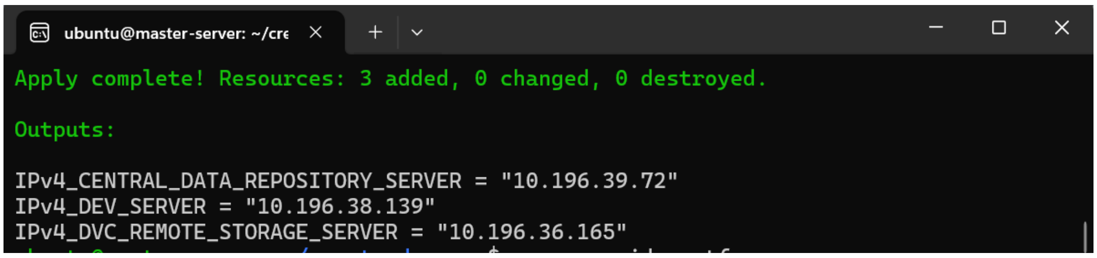
However, there were discussions about the possible effects of such a change, especially concerning the introduction of potential data uncertainties during transitions. After careful consideration, the decision was made to keep the manual demarcation process. This choice was based on prioritizing data accuracy and methodological rigor, emphasizing the project's dedication to maintaining the integrity of collected data, particularly in its early stages. Therefore, despite recognizing the benefits of simplifying procedures, the project chose to stick with the detailed manual process to ensure the reliability of the collected datasets.

### 4.1.2 Infrastructure Setup

The setup of the project's infrastructure was accomplished using the *terraform apply* command, executed on the Master Server. This command was run within the *system_management* directory, where Terraform was already initialized.

Upon execution, the infrastructure setup process systematically created each server. The successful creation of these servers is visually confirmed through the output in the console, which indicates that the servers were successfully instantiated and also displays their respective IP addresses. The screenshot of the received feedback is illustrated in Figure 4.2.

Following the deployment, a comprehensive verification process was undertaken to ensure that each server met its specific deployment requirements. This verification involved checking whether all necessary applications and configurations, as defined in the server specifications in Section 3.3.5, were correctly installed and configured.


```
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

IPv4_CENTRAL_DATA_REPOSITORY_SERVER = "10.196.39.72"
IPv4_DEV_SERVER = "10.196.38.139"
IPv4_DVC_REMOTE_STORAGE_SERVER = "10.196.36.165"
```

Figure 4.2: Successful Infrastructure Setup: Console Feedback

### Central Data Repository Server

The Central Data Repository Server is equipped to handle the storage and management of CSV data files crucial to the project. This server requires Docker and Docker Compose for container management and a GitLab Runner for handling CI/CD processes. Following the automated infrastructure setup executed via Terraform, all necessary software installations and service configurations, including the running GitLab Runner, were successfully implemented without manual intervention. Figure 4.3 presents a screenshot confirming these installations and the operational status of the GitLab Runner.

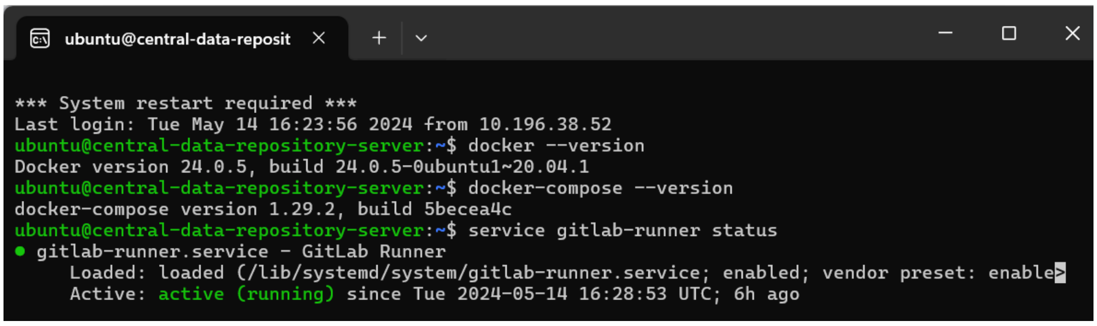

```
*** System restart required ***
Last login: Tue May 14 16:23:56 2024 from 10.196.38.52
ubuntu@central-data-repository-server:~$ docker --version
Docker version 24.0.5, build 24.0.5-0ubuntu1~20.04.1
ubuntu@central-data-repository-server:~$ docker-compose --version
docker-compose version 1.29.2, build 5becea4c
ubuntu@central-data-repository-server:~$ service gitlab-runner status
● gitlab-runner.service - GitLab Runner
     Loaded: loaded (/lib/systemd/system/gitlab-runner.service; enabled; vendor preset: enable>
     Active: active (running) since Tue 2024-05-14 16:28:53 UTC; 6h ago
```

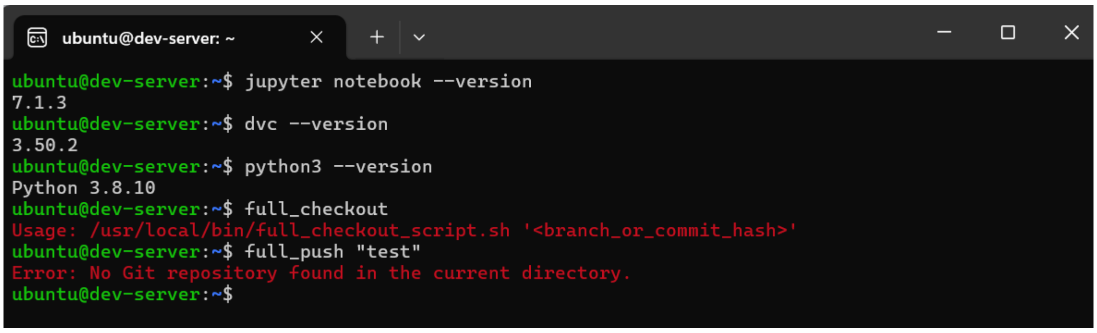
Figure 4.3: Successful Infrastructure Setup: Central Data Repository Server

The next step involves registering the GitLab Runner to fully activate the CI/CD pipeline, which will manage the deployment of the API application automatically. This setup ensures the server is prepared to support CI/CD activities, enhancing operational efficiency and reliability.

### Developer Server

The Developer Server should be equipped with DVC for version control, Jupyter for interactive development, and Python to run applications. It also should include custom scripts *full_checkout* and *full_push* added to the *.bashrc* file for ease of use directly from the command line.

The screenshot presented in Figure 4.4 clearly demonstrates that the Developer Server has successfully installed and verified versions of Jupyter, DVC, and Python components. The screenshot also shows the usage of custom scripts, *full_checkout* and *full_push*. Notably, an attempt to run *full_checkout* without specifying a branch or commit hash displays its usage prompt, indicating the script is operational, though it is not used in the proper context here. Similarly, the *full_push* command reveals an error due to the absence of a Git repository in the current directory, suggesting that these scripts are ready but require correct environmental setup to function fully.



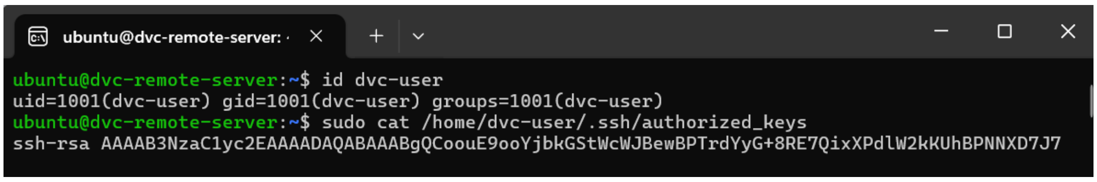
```
ubuntu@dev-server:~$ jupyter notebook --version
7.1.3
ubuntu@dev-server:~$ dvc --version
3.50.2
ubuntu@dev-server:~$ python3 --version
Python 3.8.10
ubuntu@dev-server:~$ full_checkout
Usage: /usr/local/bin/full_checkout_script.sh '<branch_or_commit_hash>'
ubuntu@dev-server:~$ full_push "test"
Error: No Git repository found in the current directory.
ubuntu@dev-server:~$
```

Figure 4.4: Successful Infrastructure Setup: Developer Server

To fully operationalize the server, manual steps include adding the server's SSH key to GitLab for repository access, and to the DVC Remote Storage Server for data access, ensuring secure and comprehensive integration into our development environment.

### DVC Remote Storage Server

The DVC Remote Storage Server is required to have a dedicated user, referred to as dvc-user, for managing data versions. This user must have an *authorized_keys* file in the .ssh folder to securely manage SSH access. Figure 4.5 successfully demonstrates that this requirement has been met, showcasing the presence of the user and the file.



```
ubuntu@dvc-remote-server:~$ id dvc-user
uid=1001(dvc-user) gid=1001(dvc-user) groups=1001(dvc-user)
ubuntu@dvc-remote-server:~$ sudo cat /home/dvc-user/.ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCoouE9ooYjbkGStWcWJBewBPTrdYyG+8RE7QixXPdlW2kKUhBPNNXD7J7
```

Figure 4.5: Successful Infrastructure Setup: DVC Remote Storage Server

### 4.1.3 Operational Testing of the Central Data Repository

In this section, the operational efficacy of the central data repository is examined, emphasizing the API as an essential component of the system. It is facilitated by the API that both data storage and metadata retrieval are managed, ensuring comprehensive handling of information. To showcase the functionality and robustness of the API, manual testing is conducted based on four distinct use cases.

#### POST Endpoint Testing

The POST endpoint allows users to upload a CSV file containing earplug data. The API extracts relevant data from the file and saves it into the database.

- **Case 1 - No File Provided Testing Case**:
  - **Description**: Access the POST endpoint without uploading any file.
  - **Desired Response**: The API responds with an error message indicating the requirement for a file upload.
  - **Actual Response**: The API prompts the user to upload a file by returning an error message indicating that no file was provided, guiding them to take the necessary action.
- **Case 2 - Non-CSV File Testing Case**:
  - **Description**: Attempt to upload a file in a format other than CSV (e.g., PDF, TXT).
  - **Desired Response**: The API detects the unsupported file format and returns an error message indicating the need for a CSV file.
  - **Actual Response**: The API prompts the user to upload a file by returning an error message indicating that no file was provided, guiding them to take the necessary action.
- **Case 3 - Incorrect CSV Format Testing Case**:

  - **Description**: Upload a CSV file with incorrect formatting, such as missing columns or incorrect data types.
  - **Desired Response**: The API identifies the formatting errors and returns an error message specifying the issues encountered.
  - **Actual Response**: The API detects the formatting errors within the CSV file and returns an error message detailing the specific issues encountered, ensuring data integrity.

- **Case 4 - Valid File Testing Case**:
  - **Description**: Upload a properly formatted CSV file containing relevant data.
  - **Desired Response**: The API successfully extracts the data from the file and stores it in the database.
  - **Actual Response**: The API extracts the data as expected and returns a success message along with extracted metadata object and status code indicating successful processing.

**GET Endpoint Testing**

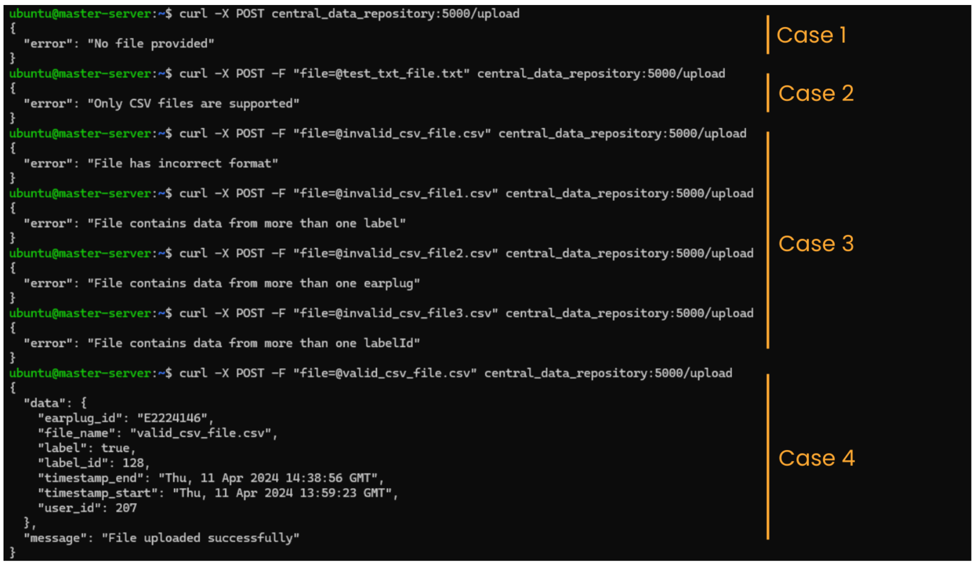
The GET endpoint retrieves a list of files based on provided filters, such as date range, earplug serial number, label type.

- **Case 1 - No Filters Provided Testing Case**:
  - **Description**: Send a request to the GET endpoint without providing any filters.
  - **Desired Response**: The API returns all available files since no filtering criteria are specified.
  - **Actual Response**: The API responds with a list of all available files.
- **Case 2 - Combination of Filters Testing Case**:
  - **Description**: Test the endpoint with various combinations of filters, such as date range, file type, and keywords.
  - **Desired Response**: The API accurately filters files based on the combination of provided filters.
  - **Actual Response**: The API returns a list of files that match the specified combination of filters, demonstrating accurate filtering functionality.

- **Case 3 - Invalid Filters Testing Case**:
  - **Description**: Provide invalid filters (e.g., incorrect date format, unsupported file type).
  - **Desired Response**: The API handles invalid filters and returns an appropriate error response.
  - **Actual Response**: The API detects and rejects invalid filters, returning an error message specifying the issue encountered.

#### Tests Outcomes

Both the GET and POST endpoints were manually tested using the curl command-line tool to validate their functionality with various files and parameters as defined in our test cases. Figure 4.6 presents a screenshot of these manual tests for the POST endpoint, clearly marking each use case with the corresponding curl commands and the API's responses, effectively demonstrating the endpoint's capacity to handle both correct and erroneous inputs.



```
ubuntu@master-server:~$ curl -X POST central_data_repository:5000/upload
{
  "error": "No file provided"
}
ubuntu@master-server:~$ curl -X POST -F "file=@test_txt_file.txt" central_data_repository:5000/upload
{
  "error": "Only CSV files are supported"
}
ubuntu@master-server:~$ curl -X POST -F "file=@invalid_csv_file.csv" central_data_repository:5000/upload
{
  "error": "File has incorrect format"
}
ubuntu@master-server:~$ curl -X POST -F "file=@invalid_csv_file1.csv" central_data_repository:5000/upload
{
  "error": "File contains data from more than one label"
}
ubuntu@master-server:~$ curl -X POST -F "file=@invalid_csv_file2.csv" central_data_repository:5000/upload
{
  "error": "File contains data from more than one earplug"
}
ubuntu@master-server:~$ curl -X POST -F "file=@invalid_csv_file3.csv" central_data_repository:5000/upload
{
  "error": "File contains data from more than one labelId"
}
ubuntu@master-server:~$ curl -X POST -F "file=@valid_csv_file.csv" central_data_repository:5000/upload
{
  "data": {
    "earplug_id": "E2224146",
    "file_name": "valid_csv_file.csv",
    "label": true,
    "label_id": 128,
    "timestamp_end": "Thu, 11 Apr 2024 14:38:56 GMT",
    "timestamp_start": "Thu, 11 Apr 2024 13:59:23 GMT",
    "user_id": 207
  },
  "message": "File uploaded successfully"
}
```

Figure 4.6: Screenshots of the Manual Testing Results for the POST Endpoint

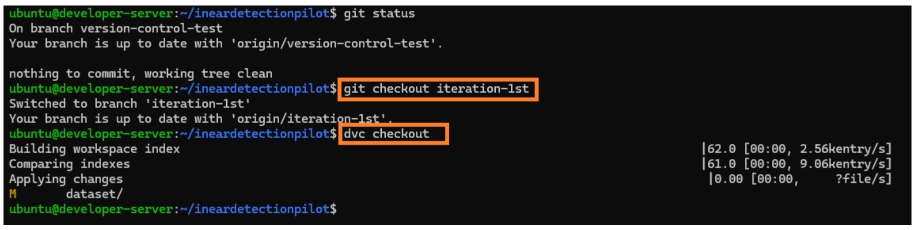
### 4.1.4 Version Control for Data and Project Management

Version control strategy played the crucial role in ensuring the reproducibility and integrity in the pilot project. Both Git and DVC were essential for tracing changes in the project's code and datasets through various iterations. The integration of these tools enabled the synchronization of code versions with corresponding dataset versions, crucial for accurate reproduction of each project phase.

To streamline this process that normally require repetitive use of multiple commands, custom scripts were developed to automate the essential version control operations like checkout and push, enhancing efficiency and reducing potential errors. Following comparison of the manual version control workflow against the script-enhanced workflow, showcases the improvements in version control process management.

### Manual Version Control Process

Default option for version control involved manual addition, committing, and pushing changes to the Git repository. This required careful attention to detail to ensure that all relevant files were staged correctly. Similarly, updates to datasets that were handled manually using DVC commands were susceptible to human error, such as failing to track all necessary files or forgetting to push updated data to the remote storage. Manually ensuring that dataset versions corresponded accurately with code revisions involved meticulous record-keeping and coordination, increasing the complexity and potential for discrepancies.



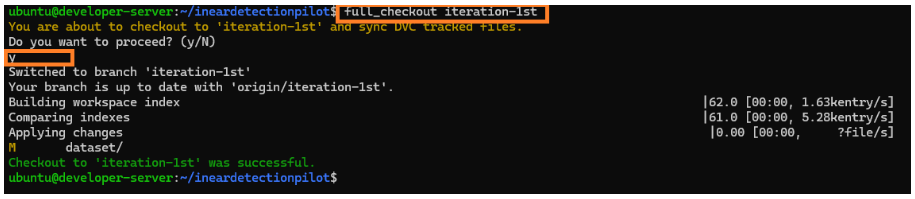
```
ubuntu@developer-server:~/ineardetectionpilot$ git status
On branch version-control-test
Your branch is up to date with 'origin/version-control-test'.

nothing to commit, working tree clean
ubuntu@developer-server:~/ineardetectionpilot$ git checkout iteration-1st
Switched to branch 'iteration-1st'
Your branch is up to date with 'origin/iteration-1st'.
ubuntu@developer-server:~/ineardetectionpilot$ dvc checkout
Building workspace index                                                |62.0 [00:00, 2.56kentry/s]
Comparing indexes                                                       |61.0 [00:00, 9.06kentry/s]
Applying changes                                                         |0.00 [00:00,     ?file/s]
M       dataset/
ubuntu@developer-server:~/ineardetectionpilot$
```

Figure 4.7: Manual Checkout Between Versions

### Scripted Version Control Process

To address the inefficiencies associated with manual version control, custom scripts were introduced to automate the checkout and push processes. When changes were made to code or data, scripts automatically handled Git and DVC commands to ensure all changes were correctly versioned and synchronized. The automation scripts were designed to reduce the number of steps required, streamlining the version control process significantly.

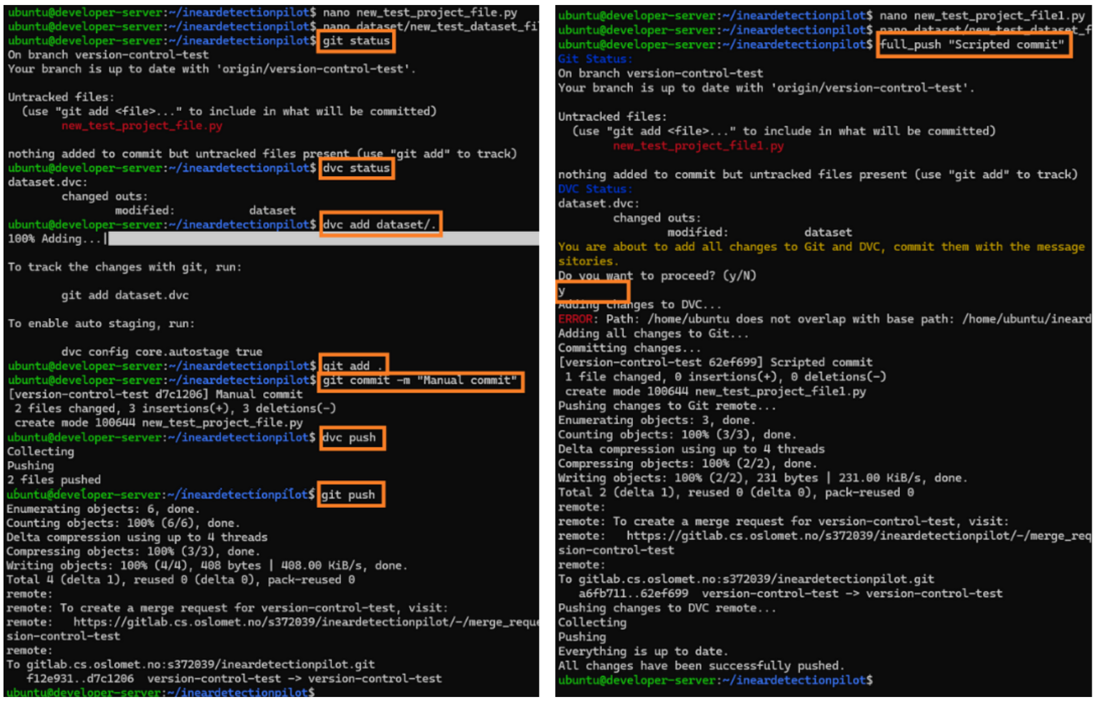


```
ubuntu@developer-server:~/ineardetectionpilot$ full_checkout iteration-1st
You are about to checkout to 'iteration-1st' and sync DVC tracked files.
Do you want to proceed? (y/N)
y
Switched to branch 'iteration-1st'
Your branch is up to date with 'origin/iteration-1st'.
Building workspace index                                                     |62.0 [00:00, 1.63kentry/s]
Comparing indexes                                                            |61.0 [00:00, 5.28kentry/s]
Applying changes                                                              |0.00 [00:00,     ?file/s]
M       dataset/
Checkout to 'iteration-1st' was successful.
ubuntu@developer-server:~/ineardetectionpilot$
```

Figure 4.8: Scripted Checkout Between Versions

### Comparison of Version Control Approaches

The examination of manual versus scripted version control methods illustrates a significant efficiency gap between the two approaches. The manual process, depicted in Figure 4.7, requires step-by-step user interaction for tasks such as checking out branches and updating DVC files, which increases the potential for user error or forgetting the dataset update. In contrast, the scripted approach, shown in Figure 4.8, automates these tasks, reducing the need of checking out the branch and the dataset.



```
ubuntu@developer-server:~/ineardetectionpilot$ nano new_test_project_file.py
ubuntu@developer-server:~/ineardetectionpilot$ nano dataset/new_test_dataset_fi
ubuntu@developer-server:~/ineardetectionpilot$ git status
On branch version-control-test
Your branch is up to date with 'origin/version-control-test'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        new_test_project_file.py

nothing added to commit but untracked files present (use "git add" to track)
ubuntu@developer-server:~/ineardetectionpilot$ dvc status
dataset.dvc:
        changed outs:
                modified:           dataset
ubuntu@developer-server:~/ineardetectionpilot$ dvc add dataset/.
100% Adding...|

To track the changes with git, run:

        git add dataset.dvc

To enable auto staging, run:

        dvc config core.autostage true
ubuntu@developer-server:~/ineardetectionpilot$ git add .
ubuntu@developer-server:~/ineardetectionpilot$ git commit -m "Manual commit"
[version-control-test d7c1206] Manual commit
 2 files changed, 3 insertions(+), 3 deletions(-)
 create mode 100644 new_test_project_file.py
ubuntu@developer-server:~/ineardetectionpilot$ dvc push
Collecting
Pushing
2 files pushed
ubuntu@developer-server:~/ineardetectionpilot$ git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 408 bytes | 408.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for version-control-test, visit:
remote:   https://gitlab.cs.oslomet.no/s372039/ineardetectionpilot/-/merge_reque
sion-control-test
remote:
To gitlab.cs.oslomet.no:s372039/ineardetectionpilot.git
   f12e931..d7c1206  version-control-test -> version-control-test
ubuntu@developer-server:~/ineardetectionpilot$
```

```
ubuntu@developer-server:~/ineardetectionpilot$ nano new_test_project_file1.py
ubuntu@developer-server:~/ineardetectionpilot$ nano dataset/new_test_dataset_f
ubuntu@developer-server:~/ineardetectionpilot$ full_push "Scripted commit"
Git Status:
On branch version-control-test
Your branch is up to date with 'origin/version-control-test'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        new_test_project_file1.py

nothing added to commit but untracked files present (use "git add" to track)
DVC Status:
dataset.dvc:
        changed outs:
                modified:           dataset
You are about to add all changes to Git and DVC, commit them with the message
sitories.
Do you want to proceed? (y/N)
y
Adding changes to DVC...
ERROR: Path: /home/ubuntu does not overlap with base path: /home/ubuntu/ineard
Adding all changes to Git...
Committing changes...
[version-control-test 62ef699] Scripted commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 new_test_project_file1.py
Pushing changes to Git remote...
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 231 bytes | 231.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for version-control-test, visit:
remote:   https://gitlab.cs.oslomet.no/s372039/ineardetectionpilot/-/merge_req
sion-control-test
remote:
To gitlab.cs.oslomet.no:s372039/ineardetectionpilot.git
   a6fb711..62ef699  version-control-test -> version-control-test
Pushing changes to DVC remote...
Collecting
Pushing
Everything is up to date.
All changes have been successfully pushed.
ubuntu@developer-server:~/ineardetectionpilot$
```

Figure 4.9: Comparison of Manual and Scripted Pushing of the Changes

Further comparison, as seen in Figure 4.9, showcases the streamlined effectiveness of scripted operations over manual ones, particularly in pushing changes where the scripted process consolidates several steps into one automated command. The rectangles in this figure highlight the reduced number of steps where user input is required, consolidating several actions into one automated command. This comparison highlights the substantial benefits of automation in managing version control, reducing time and error while improving project consistency and user experience.

## 4.2 Pilot Iterations Analysis

In this section, a detailed examination of the sequential pilot iterations conducted for the development of AI models for in-ear detection within Smart Alert devices is presented. Each iteration is methodically analyzed through the all aspects of the ML workflow.

### 4.2.1 First Iteration

The initial iteration marked the commencement of the project, aiming to lay the groundwork for integrating AI for the in-ear detection problem. This phase focused on the primary steps of gathering data personally and initiating explorations with AI algorithms, setting a foundational benchmark for the project's capabilities.

#### Data Acquisition

Data collection was meticulous, with an emphasis on capturing various scenarios of IN-EAR and OUT-OF-EAR conditions. This careful collection resulted in a robust dataset that was ideal for the initial exploration of an AI-based in-ear detection system. It should be noted, however, that the data in the first iteration came from only one person.

#### Exploratory Data Analysis and Visualisation

The Exploratory Data Analysis and Visualization step was vital in uncovering insights from the initial dataset and setting the stage for applying AI models. Three significant figures were created during this phase:

- **Pie Chart of IN-EAR and OUT-OF-EAR Labels Distribution**: The pie chart in Figure 4.10 illustrates the proportion of IN-EAR to OUT-OF-EAR labels in the dataset, providing a visual representation of label distribution, emphasizing the proportion of data in both categories. Data labeled as IN-EAR forms a larger segment at 57.1% of the total. OUT-OF-EAR condition makes up 42.9% of the data, delineating the dataset's division into nearly balanced states, which is desired for binary classification tasks.

- **Data Distribution Across Serials with IN-EAR and OUT-OF-EAR Counts**: Since the data was collected solely by one person, there is only one group of bars in Figure 4.11 representing one serial of the device. This setup underscores the homogeneity of the dataset. The chart clearly delineates the count of labels for both categories. The visualization directly complements the findings presented in the pie chart.

- **Histograms of dB Values per Frequency Band**: Figure 4.12 depict histograms for dB values captured from multiple channels in every microphone labeled as CDiag1 through CDiag5 for collar microphone, RDiag1 through RDiag5 for right earplug microphone, and LDiag1 through LDiag5 for left earplug mcirophone. Each histogram represents frequency distributions for values recorded from these channels, split into IN-EAR and OUT-OF-EAR conditions.

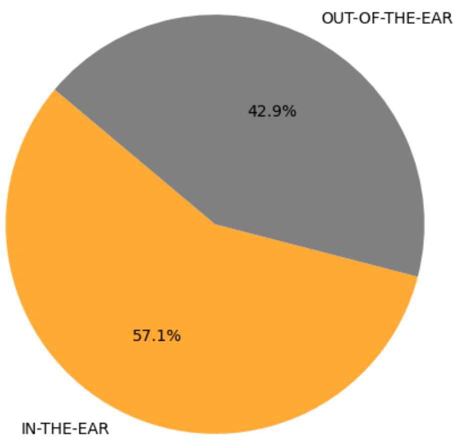


Figure 4.10: Pie Chart of IN-EAR and OUT-OF-EAR Labels Distribution (1st Iteration)

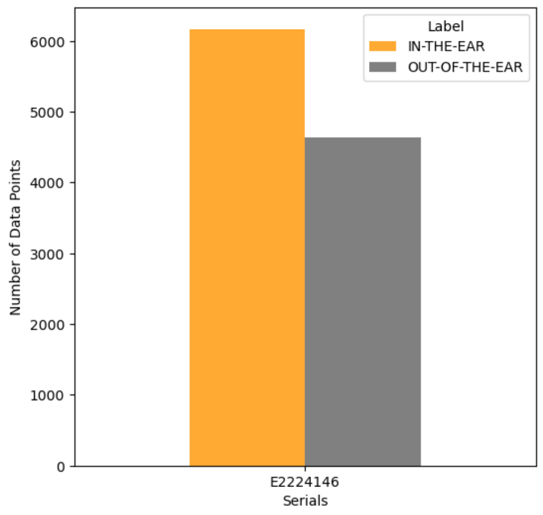


Figure 4.11: Data Distribution Across Serials with IN-EAR and OUT-OF-EAR Counts (1st Iteration)

Consistent color schemes were used in visualizations where orange represents the IN-EAR condition and gray denotes the OUT-OF-EAR condition. This color consistency helps in quickly identifying patterns and anomalies that are crucial for the analysis presented in this study.

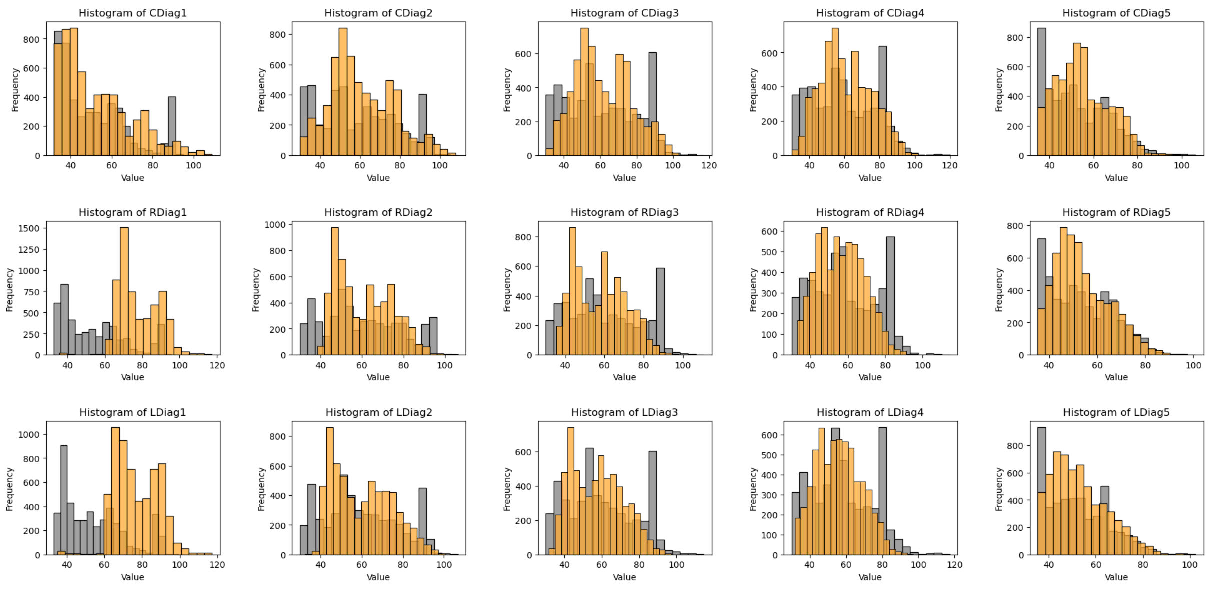


Figure 4.12: Histograms of dB Values per Frequency Band (1st Iteration)

### Data Preparation and Feature Engineering

Data cleaning involved eliminating missing data and narrowing the focus to relevant columns for clearer analysis. Each recording, originally comprising data from the left and right earplug microphones alongside the collar microphone, was reorganized into two records: one with the right earplug and collar microphone data, and another with the left earplug and collar microphone data. This step ensured that each input instance was uniformly formatted for the AI models.

No further preprocessing or feature engineering was employed at this stage, maintaining the integrity of the raw data for a baseline assessment of the model's initial performance.

### Modeling and Training

Algorithms such as k-NN, Linear SVC, DT, RF, Logistic Regression, and MLP were employed in the analysis. All models were run with their default parameters, ensuring consistency and establishing an unaltered baseline for each algorithm's performance. This straightforward approach enabled a direct comparison of the initial performance of all six algorithms without any modifications or optimizations.

**Evaluation and Visualisation**

The evaluation of the models involved two main steps. Firstly, performance metrics such as Accuracy, Precision, Recall, Specificity, and F1 Score were calculated for the testing set, which comprised 20% of the original dataset and was extracted before the training phase. Secondly, these metrics were also calculated on a validation dataset, which consisted of different files, partly sourced from various earplugs and individuals. This approach provided a comprehensive assessment of each model's performance across different datasets and scenarios. The values of these metrics for both sets and all algorithms are summarized in Table 4.1.

| **Model** | **Metric** | **Accuracy** | **Precision** | **Recall** | **Specificity** | **F1 Score** |
|---|---|---|---|---|---|---|
| | | | **Testing Set** | | | |
| k-NN | | 0.98 | 0.98 | 0.99 | 0.98 | 0.99 |
| Linear SVC | | 0.93 | 0.92 | 0.97 | 0.88 | 0.94 |
| RF | | 0.98 | 0.98 | 0.99 | 0.97 | 0.99 |
| MLP | | 0.98 | 0.98 | 0.98 | 0.97 | 0.98 |
| Logistic Regression | | 0.95 | 0.96 | 0.96 | 0.94 | 0.96 |
| DT | | 0.98 | 0.98 | 0.98 | 0.97 | 0.98 |
| | | | **Validation Set** | | | |
| k-NN | | 0.87 | 0.90 | 0.79 | 0.93 | 0.84 |
| Linear SVC | | 0.82 | 0.81 | 0.78 | 0.85 | 0.79 |
| RF | | 0.84 | 0.83 | 0.81 | 0.87 | 0.82 |
| MLP | | 0.85 | 0.97 | 0.69 | 0.98 | 0.81 |
| Logistic Regression | | 0.83 | 0.85 | 0.75 | 0.89 | 0.80 |
| DT | | 0.81 | 0.84 | 0.70 | 0.89 | 0.76 |

Table 4.1: Performance Metrics of Various Models on Testing and New Test Sets

The evaluation of various ML models on testing and validation sets highlights challenges in achieving robust detection of earplug status. All models experienced a decline in performance when applied to the more diverse validation set, suggesting issues with generalizability, particularly due to the training on data from a single individual without preprocessing adjustments like scaling or dimensionality reduction. This setup likely led to overfitting, as indicated by the decreased generalizability of the models on the validation set.

Among the models, the most critical flaw was demonstrated by a decreased recall in some, such as Linear SVC and MLP. These models are particularly vulnerable to the most dangerous type of error: failing to detect when an earplug is OUT-OF-EAR. This scenario falsely assures the user of hearing protection, preventing necessary alerts and potentially leading to unsafe exposure. Despite high precision, the low recall significantly impacts user safety as it increases the risk of undetected earplug absence. In contrast, RF and k-NN showed more balanced metrics, making them potentially more reliable for applications where user safety is paramount.

Given that this analysis represents the first iteration of model exploration, further research and model tuning are essential to enhance their performance and reliability. Future efforts should focus on incorporating more diverse training data, implementing appropriate preprocessing techniques, and refining model parameters to improve recall specifically. This iterative process is crucial to developing a robust system that reliably minimizes false negatives, ensuring that users are always alerted to potential hearing hazards.

#### Iteration Summary

- Initial assessment of the dataset, focusing on the distribution of labels and identifying patterns in the data.
- First attempts at creating models with the limited dataset originating from single person measurements.
- Encountered issues with performance drop after validating the models with completely unseen data, leading to plans for more robust data preprocessing and expanding the dataset.

### 4.2.2 Second Iteration

In the second iteration of the pilot development for AI models focused on in-ear detection, the scope of data inclusion was significantly broadened to enhance the models' generalizability. Unlike the first iteration, which relied on data from a single individual, this phase involved data from multiple users. Additionally, data scaling techniques were implemented to standardize the data, ensuring consistency and comparability across different inputs. Moreover, a method was implemented to systematically display the models' accuracy for each file in the validation set separately, enabling a more detailed analysis of performance across diverse scenarios.

**Data Acquisition**

The datasets employed for this iteration comprised recordings from additional users, providing a richer and more diverse data pool. This expansion was intended to challenge the models with a wider array of data patterns and usage scenarios, enhancing their ability to adapt to more generalized settings.

**Exploratory Data Analysis and Visualisation**

The Exploratory Data Analysis and Visualization phase for the second iteration was crucial for interpreting the expanded dataset and preparing it for subsequent AI model application. Despite the methodology remaining consistent with the first iteration, the incorporation of a larger and more diverse dataset has influenced the outcomes of the visual representations. Two key figures were generated during this phase;

- **Pie Chart of IN-EAR and OUT-OF-EAR Labels Distribution**: Presented in Figure 4.13, this pie chart details the label distribution within the dataset, showing a slight predominance of OUT-OF-EAR labels at 58.2% compared to 41.8% for IN-EAR. This shift from the first iteration's near balance provides new challenges and insights into label disparity, emphasizing the need for careful consideration in model training to avoid bias.
- **Data Distribution Across Serials with IN-EAR and OUT-OF-EAR Counts**: Figure 4.14 illustrates the distribution of labels across different serials of the device, reflecting the variability in data collection sources. This bar chart shows significant differences in label counts between serials, highlighting the heterogeneity introduced by incorporating data from multiple users and devices.

**Data Preparation and Feature Engineering**

The methodology for processing data in the second iteration retained much of the first iteration's framework, with a crucial enhancement in the form of data scaling. This adjustment was specifically introduced to boost model performance, particularly for algorithms that are sensitive to feature scaling. Without such scaling, certain features might disproportionately influence the model due to their range

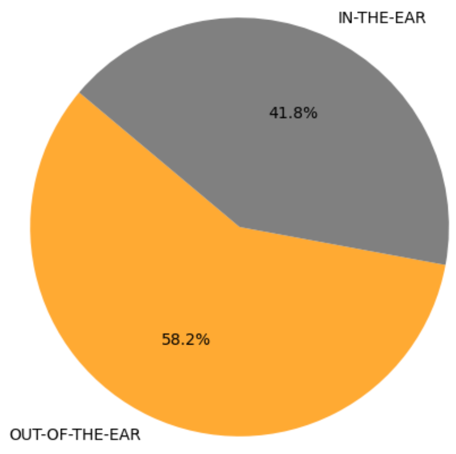


Figure 4.13: Pie Chart of IN-EAR and OUT-OF-EAR Labels Distribution (2nd Iteration)

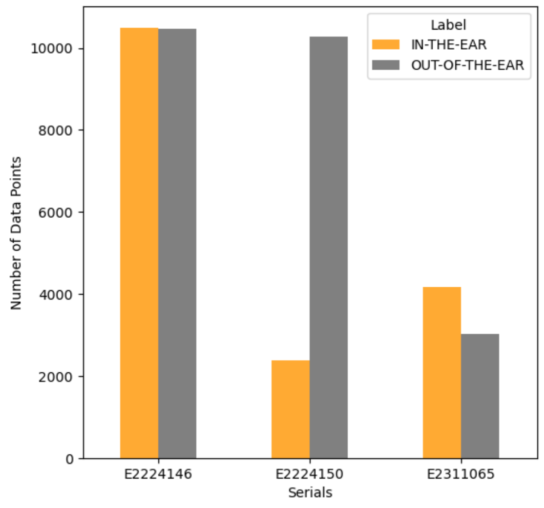


Figure 4.14: Data Distribution Across Serials with IN-EAR and OUT-OF-EAR Counts (2nd Iteration)

and distribution, potentially skewing the results. Standardizing the data ensures that each feature contributes equally to the model's decision process, thus fostering more accurate and consistent predictions across varied datasets. This approach is particularly critical for algorithms like k-NN, SVMs and neural networks, where feature scaling can significantly impact overall performance.

### Modeling and Training

The modeling and training approaches in the second iteration were consistent with the first, with no changes to the models' parameters or architecture. This consistency was maintained to directly assess the effects of the expanded dataset and the new scaling preprocessing step on model performance, highlighting the impact of improved data handling without altering the underlying model structures.

### Evaluation and Visualisation

The evaluation process for the second iteration of AI models closely followed the framework established in the first iteration, utilizing the same performance metrics—Accuracy, Precision, Recall, Specificity, and F1 Score. Metrics were assessed for both the testing set, which comprised 20% of the original dataset, and a validation set, which utilized the same previously unseen files as the first iteration. This consistent approach allows for a straightforward comparison of results across iterations to clearly identify improvements or setbacks resulting from modifications in the data volume and preprocessing steps.

Additionally, this iteration included enhanced visualizations that specifically focused on the accuracy of each model for individual validation files. This new aspect of visualization aimed to uncover any specific challenges the models faced with particular files or conditions, providing deeper insights into model performance and areas needing further refinement.

The summarized results of this evaluation are detailed in Table 4.2. Although the values for testing set showed a decrease in performance compared to the first iteration, this was anticipated due to the increased complexity and variability in the data. The models were now dealing with inputs that were less homogeneous, which typically poses a greater challenge in achieving high accuracy. Analysis of these results reveals that while the performance metrics on the testing set show a general decrease, there is a notable increase in the metrics for the validation set. This pattern suggests that the models, though slightly less effective on the testing set, have improved in their ability to generalize to new, unseen data. For instance, while the precision and recall values are lower on the standardized testing set compared to the first iteration, they have increased on the validation set, indicating enhanced model robustness and adaptability.

Following the encouraging initial results from the model evaluations, it was decided to further verify these outcomes using another method to ensure their reliability. Stratified K-Fold cross-validation was chosen for this purpose because it helps confirm that the models perform consistently across different data segments. This method maintains an equal proportion of each class in every fold, providing a thorough assessment of model stability and generalization across varied subsets of the dataset. This step was crucial in validating the robustness of the models before their practical application.

The Table 4.3 presents performance metrics for the models, comparing results from standard training to those obtained through Stratified K-Fold cross-validation, with each cross-validation result representing the average of five splits. The metrics across both methods do not vary significantly, indicating that the models demonstrate consistent performance and reliable generalization capabilities when exposed to different subsets of the data. This consistency is particularly noteworthy as it suggests that the models are not overfitting to the training set and are likely to perform well on unseen data. The close alignment of these values across standard

| Model | Metric | Accuracy | Precision | Recall | Specificity | F1 Score |
|---|---|---|---|---|---|---|
| | | **Testing Set** | | | | |
| k-NN | | 0.88 | 0.87 | 0.84 | 0.91 | 0.86 |
| Linear SVC | | 0.84 | 0.85 | 0.75 | 0.91 | 0.80 |
| RF | | 0.86 | 0.84 | 0.82 | 0.89 | 0.83 |
| MLP | | 0.91 | 0.88 | 0.92 | 0.91 | 0.90 |
| Logistic Regression | | 0.84 | 0.84 | 0.76 | 0.90 | 0.80 |
| DT | | 0.85 | 0.82 | 0.81 | 0.87 | 0.82 |
| | | **Validation Set** | | | | |
| k-NN | | 0.87 | 0.92 | 0.77 | 0.95 | 0.84 |
| Linear SVC | | 0.82 | 0.92 | 0.66 | 0.95 | 0.77 |
| RF | | 0.88 | 0.94 | 0.78 | 0.96 | 0.86 |
| MLP | | 0.89 | 0.91 | 0.85 | 0.93 | 0.88 |
| Logistic Regression | | 0.83 | 0.91 | 0.68 | 0.95 | 0.78 |
| DT | | 0.85 | 0.89 | 0.76 | 0.92 | 0.82 |

Table 4.2: Performance Metrics of Machine Learning Models on Standardized and Validation Sets (2nd Iteration)

training and cross-validation is encouraging, as it validates the robustness of the training process and the models' ability to handle varied data inputs effectively.

Additionally, this iteration included enhanced visualizations that specifically focused on the accuracy of each model for individual validation files. This new aspect of visualization aimed to uncover any specific challenges the models faced with particular files or conditions, providing deeper insights into model performance and areas needing further refinement. Figure 4.15 illustrates the performance of each AI model in terms of accuracy across individual validation files.

The chart reveals that for most validation files, the models achieve very good accuracies, often exceeding 0.90, indicating strong performance across various scenarios. However, there is a notable exception in the "IN walk-conversation.csv" file, where accuracies significantly drop, with some models like Linear SVC showing accuracies as low as 0.46. This specific file highlights a critical area of concern where models struggle to maintain reliable performance. After consultation with an audiology domain expert, it was determined that during this particular measurement, one earplug was not properly fitted, likely causing the models' confusion. To investigate this hypothesis further, the accuracies for this file were

| Model | Method | Accuracy | Precision | Recall | Specificity | F1 Score |
|---|---|---|---|---|---|---|
| k-NN | ST | 0.88 | 0.87 | 0.84 | 0.91 | 0.86 |
| | CV | 0.89 | 0.88 | 0.85 | 0.92 | 0.87 |
| Linear SVC | ST | 0.84 | 0.85 | 0.75 | 0.91 | 0.80 |
| | CV | 0.84 | 0.85 | 0.75 | 0.90 | 0.80 |
| RF | ST | 0.86 | 0.84 | 0.82 | 0.89 | 0.83 |
| | CV | 0.87 | 0.85 | 0.83 | 0.90 | 0.84 |
| MLP | ST | 0.91 | 0.88 | 0.92 | 0.91 | 0.90 |
| | CV | 0.91 | 0.90 | 0.85 | 0.95 | 0.89 |
| Logistic Regression | ST | 0.84 | 0.84 | 0.76 | 0.90 | 0.80 |
| | CV | 0.84 | 0.84 | 0.77 | 0.90 | 0.80 |
| DT | ST | 0.85 | 0.82 | 0.81 | 0.87 | 0.82 |
| | CV | 0.85 | 0.83 | 0.81 | 0.88 | 0.82 |

Table 4.3: Comparison of Performance Metrics Using Cross-Validation (CV) and Standard Training (ST) (2nd Iteration)

additionally split between the left and right earplug, allowing a more detailed examination of how the fit of each earplug affected model performance.

The chart presented in Figure 4.16 showcases accuracy metrics for AI models used in detecting earplug status, with red indicating the right earplug and blue the left, aligning with audiology standards. Notably, discrepancies in accuracy for the „IN walk-conversation.csv" and „IN tram-ride.csv" files were initially perplexing. Upon consultation with a domain expert and further investigation, it was determined that these were not model inaccuracies. Instead, these results highlighted the models' capability to discern suboptimal earplug fittings—a crucial safety feature. The earplugs, though technically „IN", were not properly fitted, a state that the models accurately identified, demonstrating their effectiveness in detecting not just clear-cut cases but also nuanced situations of earplug use. This insight led to an understanding that the AI's predictions were valid and valuable, showcasing the sophisticated detection abilities of the models.

#### Iteration Summary

- Larger and more diverse dataset, sourced from several people.

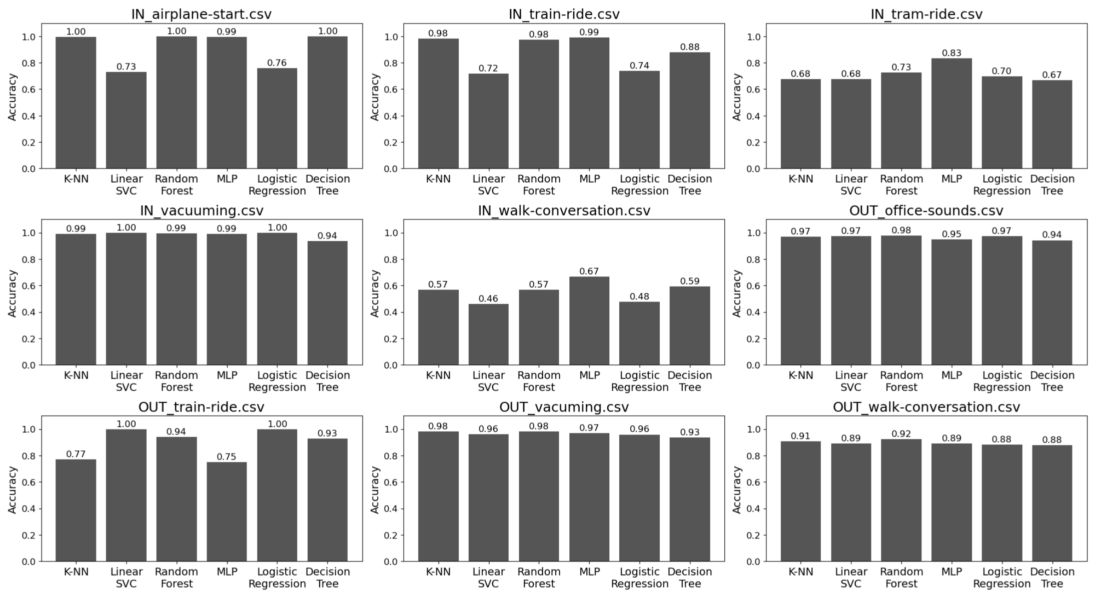


Figure 4.15: Models Accuracy by Validation Files (2nd Iteration)

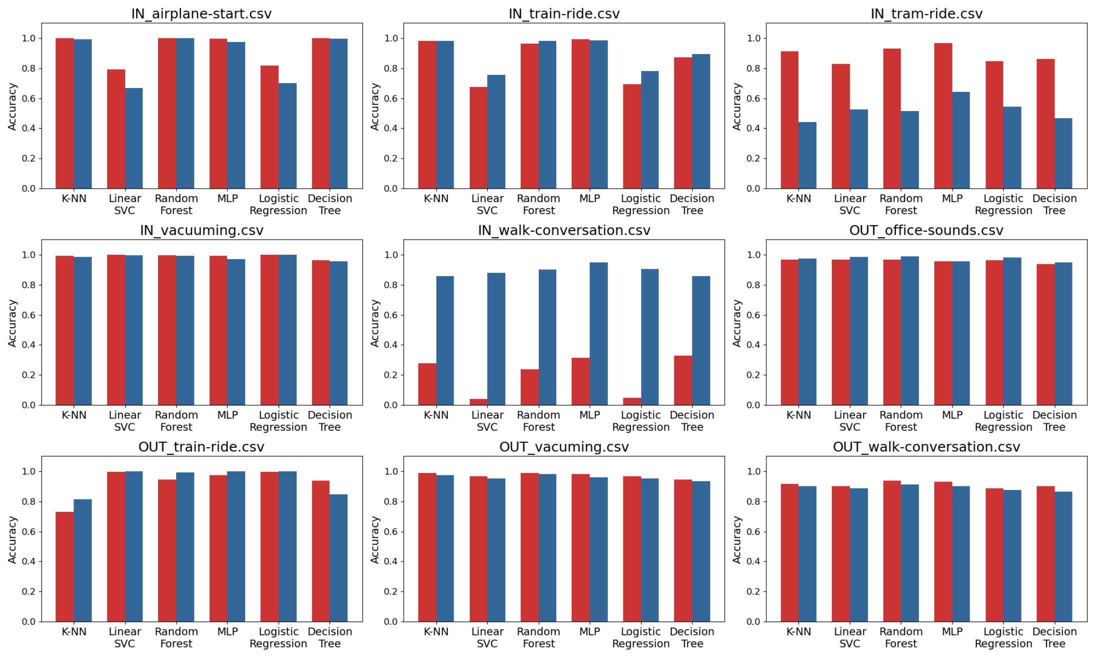


Figure 4.16: Models Accuracy by Validation Files for Left/Right Earplug (2nd Iteration)

- Added scaling of dataset in the Data Preparation step to serve the scale-sensitive algorithms.

- Utilization of cross-validation techniques to ensure that the improved results were stable and not due to random variation or specific to the train-test split used.

### 4.2.3 Third Iteration

In this third iteration of the pilot development, an experimental approach has been taken to enhance the interpretability of the models. Given the strong performance observed in previous iterations, it has been decided to deepen the experimental investigation into the decision-making processes of these models. This focus is crucial for further refining the models, increasing transparency, and building user trust by providing clear insights into the internal workings of the AI systems used in earplug detection.

#### Data Acquisition

In the third iteration of our AI model development, no new data was introduced; however, additional preprocessing and feature engineering techniques were applied to explore further enhancements that could be derived from the existing dataset. This approach was adopted to delve deeper into the data's potential, focusing on extracting more nuanced insights.

#### Exploratory Data Analysis and Visualisation

In the third iteration of the analysis, a correlation matrix was incorporated into the Exploratory Data Analysis and Visualization section to enhance the understanding of relationships among variables when earplugs are IN-EAR and OUT-OF-EAR. This addition aimed to identify key interdependencies between the dB values recorded from earplug and collar microphones across different frequency ranges.

The correlation matrices for IN-EAR data and OUT-OF-EAR data are depicted in Figures 4.17 and 4.18 respectively. Analysis of these matrices reveals that the overall correlations between corresponding Diag and CDiag variables are higher in the OUT-OF-EAR data compared to the IN-EAR data. This observation aligns with the expectation that when earplugs are out, the earplug and collar microphones should detect similar sound levels in the corresponding frequency bands, leading to higher correlations. Such consistency in sound detection across the two microphone types when earplugs are not in use suggests that both devices are equally influenced by the external acoustic environment, reflecting a natural correlation in their measurements.

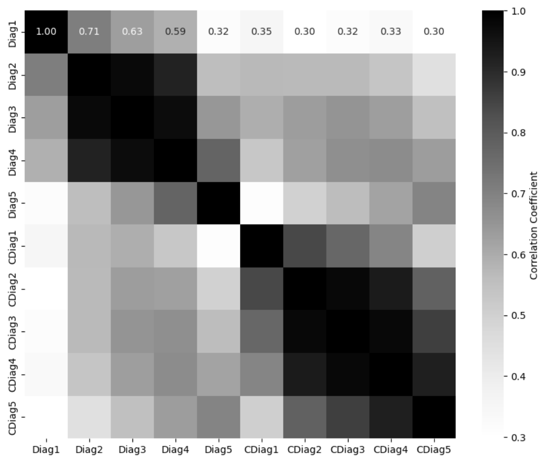


Figure 4.17: Correlation Matrix for IN-EAR Data (3rd Iteration)

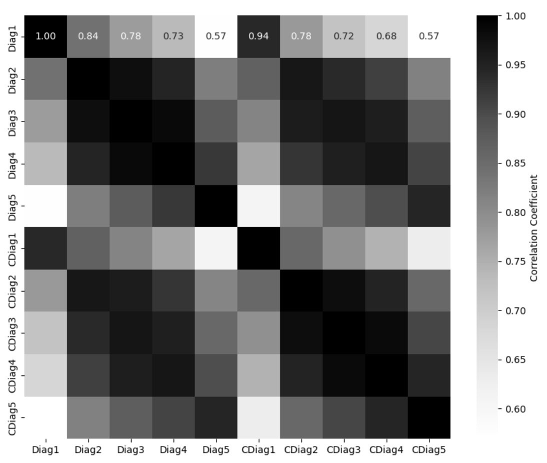


Figure 4.18: Correlation Matrix for OUT-OF-EAR Data (3rd Iteration)

**Data Preparation and Feature Engineering**

In the Data Preparation and Feature Engineering section of this iteration, it was decided to explore the potential benefits of adding differential features for each of the five frequency bands, based on domain knowledge. This initiative involves calculating the difference in decibel levels between the collar and earplug microphones for each specific frequency band. The introduction of these differential features is a strategic move to determine if such enhancements can significantly improve the performance of the in-ear detection models.

Three versions of the dataset were subsequently prepared: one containing only the original dB values from the microphones, a second combining both the original dB values and the differential features, and a third composed solely of the differential features. This approach allowed for a comprehensive evaluation of how each dataset variation impacts the performance of the in-ear detection models.

Figure 4.19 displays histograms of the differential decibel values across five frequency bands (DiffDiag1 to DiffDiag5) between the earplug and collar microphones. The orange bars represent instances when the earplugs are IN-EAR, and the gray bars represent OUT-OF-EAR scenarios. Notably, the distribution of differences when the earplugs are IN tends to have a broader range compared to when the earplugs are OUT. This observation aligns with the expectation that differential values will vary more significantly when earplugs are in place, due to their ability to attenuate external sounds differently depending on the environmental noise and the wearer's activities, such as speaking. In contrast, the values are more centered around zero when the earplugs are OUT-OF-EAR, indicating similar sound levels captured by both microphones, as there is no noise attenuation by the earplugs. This visual representation underscores the variability and the potential of using these differential values for enhancing in-ear detection model accuracy.

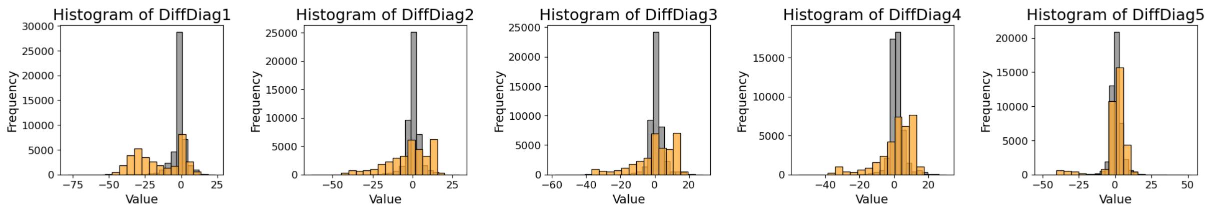


Figure 4.19: Comparative Histograms of Differential dB Values by Earplug Status for Five Frequency Bands (3rd Iteration)

### Modeling and Training

In the current iteration of our analysis, a systematic exploration was conducted to understand the impact of different feature sets on the performance of various ML models. This approach was motivated by the hypothesis that the choice of features and their scale could influence model outcomes.

To conduct this investigation, three distinct datasets that were prepared in previous steps were utilized for training and testing the models:

- **Original Dataset**: This dataset, used in previous iteration, consists of the baseline features without any modifications. It serves as a control group to gauge the effects of additional or altered features in the other dataset versions.
- **Extended Dataset**: This version includes the original features along with differential values calculated from the dataset. The differential values are intended to capture changes between pair of columns representing values for the same frequency range, potentially offering additional insights.
- **DiffOnly Dataset**: Contrasting the extended dataset, this dataset exclusively comprises differential values, excluding the original features. This allows for an assessment of whether the dynamics captured by the differential values alone can suffice for effective model training and prediction.

Additionally, it was decided to expand this experiment with an exploration of the scaling effect. In the previous iteration, all features were scaled, but interest grew regarding its actual impact, as not every algorithm requires scaling. Moreover, the original features are already within a quite similar range, suggesting that scaling might not be necessary for certain models. This nuanced approach allows for a clearer understanding of how each algorithm performs under varying preprocessing conditions, particularly for those like DTs and RFs that are less sensitive to feature scaling compared to algorithms like k-NN and SVC.

### Evaluation and Visualisation

Table 4.4, organizes performance metrics for various ML models across different data conditions, utilizing a concise notation to communicate the scaling of data. Model abbreviations used are k-NN for K-Nearest Neighbors, SVC for Support Vector Classification, RF for Random Forest, MLP for Multi-Layer Perceptron, LR for Logistic Regression, and DT for Decision Tree. Data types include "Original", "Extended", and "DiffOnly". The check mark (✓) indicates scaled data, while the cross mark (✗) denotes unscaled data, and the diamond mark (✧) signifies that scaling did not impact the results significantly, based on rounding metrics to two decimal places.

The Extended dataset did not consistently enhance model performance, indicating that the addition of extra features or data points might not always be beneficial, possibly due to introducing redundant information in this case. On the other hand, the DiffOnly dataset generally led to slightly poorer performance metrics across models, suggesting that using only differential features could remove important information necessary for achieving higher accuracy and other metrics.

Scaling generally did not have a significant impact on most models. Notably, it did not affect the k-NN model, which is often sensitive to unscaled data. It may be due to the fact that the features in the dataset already have similar ranges. Similarly, DTs and RF models, which are generally not sensitive to different feature range, showed negligible differences in the performance metrics values between scaled and unscaled data. In contrast, the MLP demonstrated a improvement with scaled data, highlighting its sensitivity to input standardization, which is typical for neural networks that benefit from uniform scaling for effective weight updates during training.

### Iteration Summary

- Including the addition of differential values to explore their potential role in improving algorithms' performance.

| Model | Data Type | Scaled | Accuracy | Precision | Recall | Specificity | F1 Score |
|---|---|---|---|---|---|---|---|
| k-NN | **Original** | ✧ | 0.88 | 0.87 | 0.84 | 0.91 | 0.86 |
| | **Extended** | ✧ | 0.88 | 0.87 | 0.85 | 0.91 | 0.86 |
| | **DiffOnly** | ✧ | 0.87 | 0.87 | 0.81 | 0.92 | 0.84 |
| SVC | **Original** | ✧ | 0.85 | 0.90 | 0.73 | 0.94 | 0.81 |
| | **Extended** | ✓ | 0.85 | 0.90 | 0.73 | 0.94 | 0.81 |
| | | ✗ | 0.85 | 0.89 | 0.74 | 0.94 | 0.81 |
| | **DiffOnly** | ✓ | 0.81 | 0.90 | 0.61 | 0.95 | 0.73 |
| | | ✗ | 0.81 | 0.90 | 0.62 | 0.95 | 0.73 |
| RF | **Original** | ✧ | 0.86 | 0.84 | 0.82 | 0.89 | 0.83 |
| | **Extended** | ✧ | 0.86 | 0.84 | 0.82 | 0.89 | 0.83 |
| | **DiffOnly** | ✧ | 0.84 | 0.82 | 0.80 | 0.88 | 0.81 |
| MLP | **Original** | ✓ | 0.92 | 0.97 | 0.83 | 0.98 | 0.89 |
| | | ✗ | 0.90 | 0.88 | 0.88 | 0.92 | 0.88 |
| | **Extended** | ✓ | 0.91 | 0.92 | 0.87 | 0.94 | 0.89 |
| | | ✗ | 0.91 | 0.93 | 0.84 | 0.96 | 0.89 |
| | **DiffOnly** | ✓ | 0.90 | 0.95 | 0.79 | 0.97 | 0.86 |
| | | ✗ | 0.89 | 0.95 | 0.79 | 0.97 | 0.86 |
| LR | **Original** | ✓ | 0.84 | 0.84 | 0.76 | 0.90 | 0.80 |
| | | ✗ | 0.83 | 0.86 | 0.72 | 0.92 | 0.78 |
| | **Extended** | ✓ | 0.84 | 0.84 | 0.76 | 0.90 | 0.80 |
| | | ✗ | 0.82 | 0.83 | 0.73 | 0.90 | 0.78 |
| | **DiffOnly** | ✧ | 0.82 | 0.88 | 0.64 | 0.94 | 0.74 |
| DT | **Original** | ✓ | 0.85 | 0.82 | 0.81 | 0.87 | 0.82 |
| | | ✗ | 0.85 | 0.82 | 0.81 | 0.88 | 0.82 |
| | **Extended** | ✧ | 0.85 | 0.83 | 0.81 | 0.88 | 0.82 |
| | **DiffOnly** | ✧ | 0.82 | 0.78 | 0.78 | 0.85 | 0.78 |

Table 4.4: Detailed Performance Metrics for Various Machine Learning Models Across Different Data Conditions

- Comprehensive testing of all models across all three datasets (Original, Extended, DiffOnly) with both scaled and unscaled data to assess the full impact of feature selection and scaling.

### 4.2.4 Fourth Iteration

The focus of the final iteration was on fine-tuning the algorithms through a comprehensive grid search and enhancing model interpretability. To facilitate this, only algorithms capable of reporting feature importance were employed. Notably, k-NN and MLP were excluded from this phase despite their strong performance in earlier iterations, due to their lack of inherent interpretability in terms of feature importance.

#### Data Acquisition

Data from the previous iteration was reused, maintaining consistency in the dataset to ensure that the potential improvements in model performance could be attributed to algorithmic refinements and not to variations in data.

#### Exploratory Data Analysis and Visualisation

This stage was unchanged from previous iterations, focusing on understanding the underlying patterns and distributions within the data.

#### Data Preparation and Feature Engineering

Optional scaling was applied based on findings from the previous iteration. DTs and RF operated on unscaled data to leverage their inherent handling of feature scales, while Logistic Regression and Linear SVC utilized scaled data. Only original features were used, excluding derived differential features explored in previous iteration to simplify the models and focus on primary data attributes.

#### Modeling and Training

Each model underwent a rigorous grid search to identify the optimal parameters, focusing solely on maximizing accuracy across three cross-validation splits. The best-performing model configurations were then used to compute detailed performance metrics.

### Evaluation and Visualisation

In the final iteration, a detailed evaluation and visualization of model performance and feature importance were conducted to better understand the predictive capabilities of each model and the significance of various features in the prediction process.

For DT and RF models, feature importance was directly extracted using built-in methods provided by these models. The values of feature importance for these two algorithms are presented on Figures 4.20 and 4.21 respectively. These methods quantify the contribution of each feature to the predictive accuracy of the model. On the other hand, for Logistic Regression and LinearSVC models, feature importance was inferred indirectly through the examination of their coefficients. While these coefficients are not directly interpretable as feature importance scores, they provide insights into the influence of each feature on the model's decision boundary. Coefficients values for Logistic Regression model are presented on Figure 4.22 and for Linear SVC model on Figure 4.23.

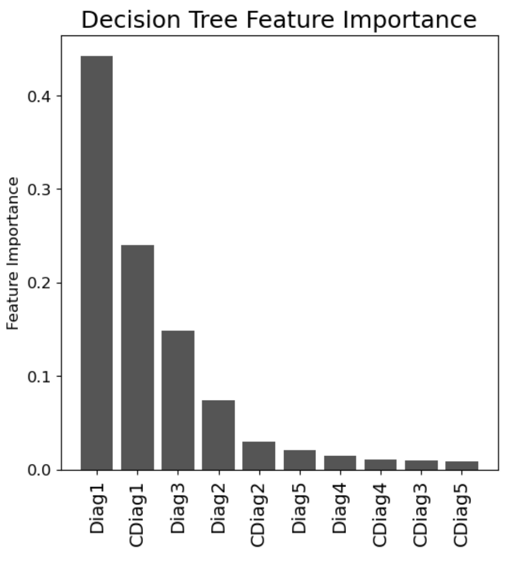


Figure 4.20: Decision Tree Feature Importance (4th Iteration)

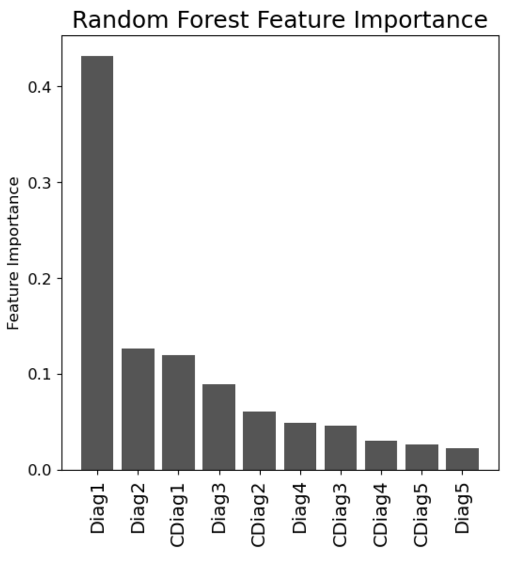


Figure 4.21: Random Forest Feature Importance (4th Iteration)

Two features - Diag1 and CDiag1 - consistently emerge as highly significant across all models, underscoring their pivotal roles in the classification process.

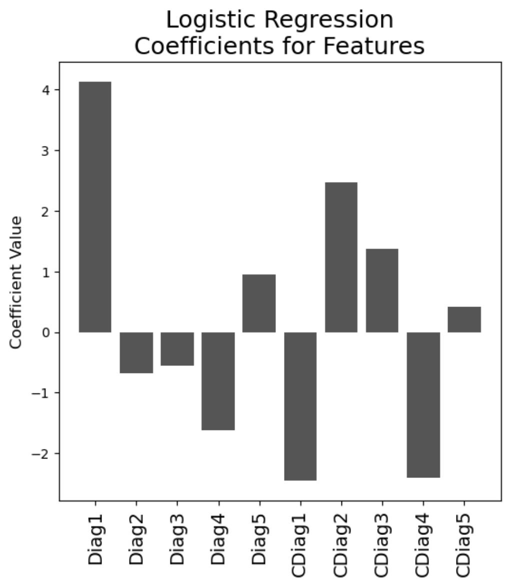


Figure 4.22: Logistic Regression Coefficients for Features (4th Iteration)

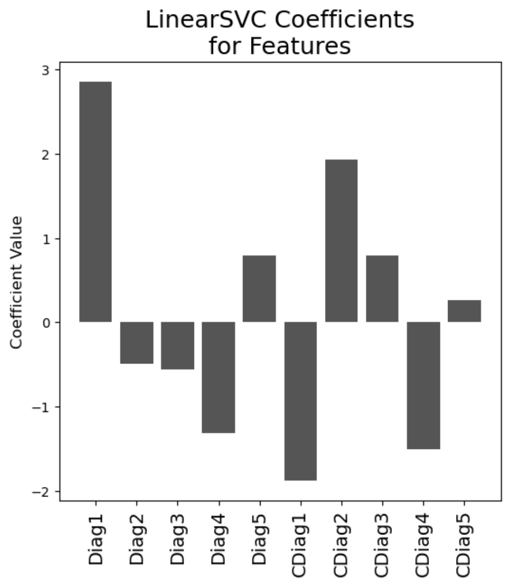


Figure 4.23: LinearSVC Coefficients for Features (4th Iteration)

Conversely, other features show varying degrees of importance across different models. This variation can be largely attributed to the strong correlation among Diag2, Diag3, and Diag4 features, and similarly among CDiag2, CDiag3, and CDiag4 which was recognized in the third iteration and is visible on Figures 4.17 and 4.18. Such correlations can influence the model's learning process, as correlated features may contribute redundant information. Consequently, some models might prioritize one feature over others within these correlated groups to optimize performance.

In terms of performance on a validation set, the algorithms, once tuned, demonstrated similar metrics values to those observed in earlier iteration. However, some issues were identified within the validation set, particularly with files exhibiting poor earplug fit for one ear. This underscores the necessity of further validation using a new validation dataset. Conducting such validation could provide a more robust assessment of the models' performance and generalizability, ensuring that the algorithms effectively handle varied real-world scenarios.

**Iteration Summary**

- Models were optimized for accuracy by rigorous grid search, with a focus on models that could evaluate feature importance.
- Key features like Diag1 and CDiag1 consistently showed high importance

across models.

- Future iterations could explore using a subset of features to simplify models and enhance performance.

## 4.3 Summary

This chapter has evaluated the created system and its components and provided a detailed analysis of the pilot iterations and their outcomes. It showcased the performance of the ML algorithms used for in-ear detection, including metrics such as accuracy and precision, presented through various graphs and tables for different cases. It tested the impact of creating additional columns, scaling data, and tuning the algorithms to optimize performance. Building on these insights, the next chapter will transition into a discussion of the broader implications of these findings, addressing the practical applications, challenges encountered during implementation, and potential strategies for optimizing the system.

# Chapter 5

# Discussion

In the previous chapter, the results of this research were presented, demonstrating the effectiveness and challenges of our approaches. This chapter aims to revisit and reflect upon the thesis' objectives, key components, and the methodologies that were used to address them. Each component will be thoroughly examined not only to assess the success and impact of these elements, but also to identify and discuss any limitations encountered. This comprehensive review seeks to contextualize our findings within the broader scope of the field, offering insights into both achievements and areas for improvement.

## 5.1 Assessment of Research Goals

This thesis was guided by two main objectives aimed at enhancing the integration of AI technologies into the field of in-ear detection for Smart Alert hearing protection devices:

- **Design and Implement a System**: To develop and implement a robust system that facilitates experimentation with AI technologies specifically for in-ear detection.
- **Evaluate ML Classification Algorithms**: To evaluate the performance and limitations of popular ML classification algorithms in accurately determining the in-ear placement of Smart Alert hearing protection devices.

Both objectives were instrumental in guiding the functional implementation of the AI system. The first objective, while broad, established a comprehensive framework for the design and implementation that allowed for effective experimentation with various AI technologies.

The second objective focused on the flexibility and adjustability of algorithms during the pilot development. Specific metrics such as accuracy, precision, recall, and F1-score were employed to evaluate the performance of these algorithms. This allowed for a more focused analysis on the capabilities and limitations of the algorithms, providing actionable insights into their performance. However, while these metrics provided a solid foundation for evaluation, the broad application scope and variability in environmental conditions presented challenges in assessing the full range of algorithmic performance.

## 5.2 Data Acqusition

A fundamental aspect of this research involved setting up a process to gather labeled data, which are essential for implementing supervised learning classification algorithms. This process was successfully established, resulting in a fully functional platform for data labeling. With the assistance of the Minuendo employees, it was possible to collect data that allowed to start experimenting with ML algorithms and even draw preliminary insights.

Despite the successful setup of the data acquisition system, there were notable limitations in the volume and diversity of data collected. The data gathered needs to be sufficiently extensive to guarantee the reliability of the results across varied scenarios. For more robust and generalizable findings, a larger and more diverse dataset would be necessary. The current manual process relies heavily on internal resources of the small company and does not allow for the extensive data collection.

To overcome these limitations and enhance data collection, transitioning the solution to a production mode could be considered. This would enable the collection of a much larger dataset, improving the diversity and representativeness of the data. However, such an expansion raises questions about user engagement and the motivation for data labeling. Introducing elements of gamification could be one strategy to encourage user participation and engagement. By making the data collection process more interactive and rewarding, it may increase user involvement and thereby enrich the dataset.

Additionally, exploring the integration of other sensors could offer alternative or supplementary data collection methods that bypass some of the current limitations associated with relying solely on microphone data for in-ear detection. However, this approach could fundamentally alter the problem statement, shifting the focus from improving microphone-based detection to developing multi-sensor detection systems.

## 5.3 System for Iterative Pilot Development

The system designed in this research provides a functional infrastructure for the initial stages of integrating AI, particularly when exploring the potential of ML algorithms. Built on a cloud-based platform, the system incorporates modern DevOps practices and Data Version Control to ensure reproducibility and systematic tracking of experiments, which is vital for maintaining the integrity of experimental processes.

Automation features prominently in the system through the use of IaC and CI/CD pipeline. These tools automate much of the deployment and management process, making the system straightforward to set up and maintain. This level of automation minimizes human error and increases the efficiency of operational workflows.

However, the system does face limitations, primarily because it is designed to facilitate experimentation rather than function as a complete MLOps system. It lacks a model deployment stage, which is critical for operationalizing ML in production environments. The system is also not built to scale seamlessly for larger, more complex projects, and it requires further development to address security more comprehensively and to add robust monitoring and logging capabilities.

These limitations highlight the system's current role as a platform for development and testing which was rather than for full-scale production use. Future enhancements could focus on expanding its capabilities to include model deployment and improving scalability and security features. Such improvements would transform the system into a comprehensive MLOps solution capable of supporting ML applications across various sectors.

## 5.4 Utilization of ML Algorithms for In-ear Detection

Although relatively straightforward, the experiments conducted with ML algorithms for in-ear detection have demonstrated promising potential. These initial trials, using simple classification models, suggest that the application of ML could enhance in-ear detection capabilities. However, a significant limitation was the relatively small and non-diverse dataset used, insufficient for fully validating the current hypotheses or refining the predictive accuracy of the developed models.

During the experiments, a notable observation emerged while analyzing the accuracy of model predictions across different validation files. Two specific files showed significantly poorer results. Further investigation, including consultation with an expert, revealed that these discrepancies were due to poor earplug fit in one of the ears during the measurements. Despite being labeled as IN, the inadequate fit of the earplugs likely compromised protection, posing a classification challenge. It turned out that when models predicted an OUT-OF-EAR label for these instances, they were not incorrect. This finding suggests that in-ear detection may not be a straightforward binary classification task, and that alternative methods assessing the quality of fit might be more appropriate.

Additionally, the pilot analysis uncovered that certain features influence model decisions more than others. This insight, if verified with a larger and more diverse dataset, suggests that focusing on these key features could streamline the modeling process. For the company, this could lead into experimenting with reducing the number of frequency bands analyzed, which would enhance system efficiency and reduce computational demands on devices. Such adjustments not only optimize performance but also align with strategic objectives to improve the overall functionality and cost-efficiency of the technology.

The initial findings from these simple experiments lay a promising foundation for further investigation. They not only emphasize the importance of expanding the dataset but also challenge existing assumptions about the nature of in-ear detection. As the project progresses, exploring new ideas could lead to more sophisticated models and a deeper understanding of the practical aspects of earplug fit and its implications for hearing protection.

## 5.5 Summary

This chapter has assessed the research objectives, delving into the challenges encountered in data gathering and evaluating the system and its limitations.
It has provided a detailed analysis of the pilot findings within the context of hearing protection, discussing how the results contribute to the field and what improvements can be made. The discussion also reflects on the practical applications and potential enhancements to better integrate ML for in-ear detection. The next chapter will conclude the thesis by summarizing the key findings, outlining the main contributions, and suggesting directions for future research.

# Chapter 6

# Summary and Conclusions

This chapter encapsulates the key outcomes and insights derived from this study alongside its contributions. It also outlines potential future directions to extend the research further.

## 6.1 Summary of Work

This thesis focused on creating a system to facilitate research and experimentation with AI technologies for in-ear detection within the Smart Alert hearing protection device. In partnership with Minuendo, a company dedicated to improving hearing safety, the project explored the potential of utilizing popular ML algorithms to bolster the functionality of hearing protection. It provided a practical framework for applying theoretical models and integrating AI technology in an industrial context.

The thesis details a comprehensive approach to developing a cloud-based infrastructure that supports the exploration and iterative refinement of ML algorithms. This setup not only facilitated the pilot testing of various AI models but also ensured the reproducibility of results, which is critical in scientific research. Recognizing the importance of a robust dataset for effectively training and evaluating AI algorithms, the project also included the development of a system for collecting and labelling data.

The pilot implementation tested multiple algorithms to identify which ones might be most effective for detecting proper earplug fit. The results from these initial tests provided insights into how AI might be used to enhance the functionality of hearing protection devices, although further testing and refinement are necessary.

Overall, this thesis lays the groundwork for further research into the application of AI in industrial safety equipment, emphasizing practical and incremental improvements rather than large-scale transformations. By integrating advanced practices in system automation and cloud infrastructure management, the project establishes a robust framework for continuous development and testing in an experimental setting.

## 6.2 Main Contributions

This thesis aimed to streamline the exploration of AI algorithms' potential to enhance hearing protection devices. The project used a comprehensive approach to integrating AI for in-ear detection, culminating in several key contributions that underscore the potential of the practical utility of AI in occupational safety and set a solid foundation for future innovations in the domain. The main contributions of this thesis are as follows:

**Development of a System for AI Experimentation** A significant achievement of this work is creating a comprehensive system that facilitates iterative testing and experimentation with AI algorithms for in-ear detection. This system is built on agile principles, allowing for the swift exploration and refinement of ML models in a structured yet flexible manner.

**Establishment of a Data Collection and Labeling Process** Recognizing the critical importance of high-quality data for AI model training and evaluation, this thesis details the creation of a data collection and labelling process. This process ensures the availability of accurate and well-annotated data sets, indispensable for developing reliable AI models. The methodology developed here supports the current project's needs and offers a blueprint for future potential research.

**Exploration of ML Algorithms for In-ear Detection** Another essential contribution is the initial evaluation and analysis of various ML algorithms. This investigation provides initial insights into the algorithms' performance and suitability for in-ear detection task. The findings from this evaluation play a crucial role in directing future efforts towards optimizing, refining, and implementing the most promising algorithms for the task.

## 6.3 Future Work

Building upon the foundations laid by this thesis, the path forward for enhancing hearing protection devices with AI is ripe with potential. A natural progression of this work would involve exploring a broader spectrum of AI algorithms. It could be interesting to explore algorithms that can analyze data in a temporal context, acknowledging the inherent continuity and time-related dependencies in earplug usage patterns. For instance, considering the physical impossibility of inserting or removing the device within a second, algorithms that account for time series data could offer more nuanced insights and improved detection accuracy.

Another exciting direction for future research involves rethinking the binary nature of in-ear detection. Rather than a simple IN-EAR or OUT-OF-EAR status, there could be significant benefits to developing models that assess the quality of the earplug fit. This nuanced approach could lead to more effective hearing protection by identifying when earplugs are not providing optimal protection, even if they are in place. Additionally, there's potential to enhance system efficiency by exploring ways to reduce the number of frequency bands analyzed. Reducing the computational load could allow more sophisticated algorithms to run on-device, enhancing the real-time capabilities of hearing protection devices.

Furthermore, the continuation of data-gathering efforts is of utmost importance. Expanding the dataset with broader usage scenarios, environmental conditions, and user behaviors will be pivotal in refining AI models. A more diverse and comprehensive dataset will empower the development of algorithms that are robust across various real-world conditions, thereby bolstering the reliability and effectiveness of AI-enhanced hearing protection devices.

Regarding system infrastructure, there is an opportunity to further develop the cloud-based capabilities introduced in this thesis. Enhancing the cloud infrastructure to better support AI model development and deployment could involve more sophisticated automation and optimization strategies. For example, implementing advanced MLOps practices could help streamline model deployment, updates, and management, improving the system's efficiency and scalability.

A significant milestone in the evolution of hearing protection technology will be the implementation of AI algorithms to function on the fly within the device units themselves. This significant shift would replace the static, rule-based systems with dynamic learning models capable of real-time analysis. However, achieving this transition demands not only technical advancements but also a concerted effort to build greater trust in the developed AI models. Establishing this trust requires rigorous validation of the models' reliability and accuracy in various scenarios.

# Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean and Matthieu Devin. *TensorFlow: Large-scale machine learning on heterogeneous systems*. Generic. 2015.

[2] Sridhar Alla and Suman Kalyan Adari. *Beginning MLOps with MLFlow: Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure*. 1st Edition. Berkeley, CA: Berkeley, CA: Apress L. P, 2020. ISBN: 1484265483 9781484265482 9781484265499 1484265491. DOI: 10.1007/978-1-4842-6549-9.

[3] Mohammad Hassan Almaspoor, Ali Safaei, Afshin Salajegheh and Behrouz Minaei-Bidgoli. 'Support Vector Machines in Big Data Classification: A Systematic Literature Review'. In: (2021).

[4] Rebecka C. Ångström, Michael Björn, Linus Dahlander, Magnus Mähring and Martin W. Wallin. 'Getting AI Implementation Right: Insights from a Global Survey'. In: *California Management Review* 66.1 (2023), pp. 5–22. ISSN: 0008-1256.

[5] Ken Arnold, James Gosling and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005. ISBN: 0321349806.

[6] Minuendo AS. *Smart Alert*. Accessed: 29.03.2024. 2024. URL: https://www.minuendo.com/smart-alert.

[7] Les Atlas, Jerome Connor, Dong Park, Mohamed El-Sharkawi, Robert Marks, Alan Lippman, Ronald Cole and Yeshwant Muthusamy. 'A performance comparison of trained multilayer perceptrons and trained classification trees'. In: IEEE, pp. 915–920.

[8] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt and Ron Jeffries. *The agile manifesto*. Generic. 2001.

[9] Ayoub Bensakhria. 'End-to-End MLOPS Pipeline for Object Detection using TensorFlow 2 and Flask'. In: ().

[10] Mathieu Bérubé, Tanya Giannelia and Gregory Vial. 'Barriers to the Implementation of AI in Organizations: Findings from a Delphi Study'. In: (2021). ISSN: 0998133140.

[11] Sumon Biswas, Mohammad Wardat and Hridesh Rajan. 'The art and practice of data science pipelines'. In.

[12] Ivette Cejas, Jennifer Coto, Chrisanda Sanchez, Meredith Holcomb and Nicole E. Lorenzo. 'Prevalence of depression and anxiety in adolescents with hearing loss'. In: *Otology & neurotology* 42.4 (2021), e470–e475. ISSN: 1531-7129.

[13] Kou-Huang Chen, Shih-Bin Su and Kow-Tong Chen. 'An overview of occupational noise-induced hearing loss among workers: epidemiology, pathogenesis, and preventive measures'. In: *Environmental health and preventive medicine* 25.1 (2020), p. 65. ISSN: 1342-078X.

[14] Alexander Chern, Alexandria L. Irace and Justin S. Golub. 'The laterality of age-related hearing loss and depression'. In: *Otology & Neurotology* 43.6 (2022), pp. 625–631. ISSN: 1531-7129.

[15] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[16] Mandepudi Nobel Chowdary, Bussa Sankeerth, Chennupati Kumar Chowdary and Manu Gupta. 'Accelerating the Machine Learning Model Deployment using MLOps'. In: vol. 2327. IOP Publishing, p. 012027. ISBN: 1742-6596.

[17] Angelo Corallo, Anna Maria Crespino, Vito Del Vecchio, Massimiliano Gervasi, Mariangela Lazoi and Manuela Marra. 'Evaluating maturity level of big data management and analytics in industrial companies'. In: *Technological Forecasting and Social Change* 196 (2023), p. 122826. ISSN: 0040-1625.

[18] DataCamp. *Preprocessing in Data Science (Part 2): Centering, Scaling and Logistic Regression*. Accessed: 05.05.2024. May 2016. URL: https://www.datacamp.com/tutorial/preprocessing-in-data-science-part-2-centering-scaling-and-logistic-regression.

[19] Peter Denning, Douglas E. Comer, David Gries, Michael C. Mulder, Allen B. Tucker, A. Joe Turner and Paul R. Young. 'Computing as a discipline: preliminary report of the ACM task force on the core of computer science'. In: pp. 41–41.

[20] Christof Ebert, Gorka Gallardo, Josune Hernantes and Nicolas Serrano. 'DevOps'. In: *IEEE software* 33.3 (2016), pp. 94–100. ISSN: 0740-7459. DOI: 10.1109/MS.2016.68.

[21] Pramod Gupta and Naresh K. Sehgal. *Introduction to machine learning in the cloud with python: Concepts and practices*. Springer Nature, 2021. ISBN: 3030712702.

[22] Johannes Hangl, Simon Krause and Viktoria Joy Behrens. 'Drivers, barriers and social considerations for AI adoption in SCM'. In: *Technology in Society* 74 (2023), p. 102299. ISSN: 0160-791X.

[23] Charles R. Harris, K. Jarrod Millman, Stéfan J. Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg and Nathaniel J. Smith. 'Array programming with NumPy'. In: *Nature* 585.7825 (2020), pp. 357–362. ISSN: 0028-0836.

[24] Jeff Heaton, Ian Goodfellow, Yoshua Bengio and Aaron Courville. 'Deep learning: Genetic programming and evolvable machines'. In: *Genetic programming and evolvable machines* 19.1 (2018), pp. 305–307. ISSN: 1389-2576. DOI: 10.1007/s10710-017-9314-z.

[25] Jonny Holmström. 'From AI to digital transformation: The AI readiness framework'. In: *Business Horizons* 65.3 (2022), pp. 329–339. ISSN: 0007-6813.

[26] John D. Hunter. 'Matplotlib: A 2D graphics environment'. In: *Computing in science & engineering* 9.03 (2007), pp. 90–95. ISSN: 1521-9615.

[27] Plotly Technologies Inc. *Collaborative data science*. 2015. URL: https://plot.ly.

[28] Sayali D. Jadhav and H. P. Channe. 'Comparative study of K-NN, naive Bayes and decision tree classification techniques'. In: *International Journal of Science and Research (IJSR)* 5.1 (2016), pp. 1842–1845.

[29] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani and Jonathan Taylor. *An introduction to statistical learning: With applications in python*. Springer Nature, 2023. ISBN: 3031387473.

[30] Zohaib Jan, Farhad Ahamed, Wolfgang Mayer, Niki Patel, Georg Grossmann, Markus Stumptner and Ana Kuusk. 'Artificial intelligence for industry 4.0: Systematic review of applications, challenges, and opportunities'. In: *Expert Systems with Applications* 216 (2023), p. 119456. ISSN: 0957-4174.

[31] Dona M. P. Jayakody, Osvaldo P. Almeida, Craig P. Speelman, Rebecca J. Bennett, Thomas C. Moyle, Jessica M. Yiannos and Peter L. Friedland. 'Association between speech and high-frequency hearing loss and depression, anxiety and stress in older adults'. In: *Maturitas* 110 (2018), pp. 86–91. ISSN: 0378-5122.

[32] JetBrains. *PyCharm Documentation*. https://www.jetbrains.com/pycharm/documentation/. Accessed: 05.05.2024. n.d.

[33] Meenu Mary John, Helena Holmström Olsson and Jan Bosch. 'Towards mlops: A framework and maturity model'. In: IEEE, pp. 1–8. ISBN: 1665427051.

[34] Ben Johnson and Anjana S. Chandran. 'Comparison between Python, Java and R progrmming language in machine learning'. In: *Int. Res. J. Modernization Eng. Technol. Sci* 3.6 (2021), pp. 1–6.

[35] Mayank Kejriwal. 'AI in Practice and Implementation: Issues and Costs'. In: *Artificial Intelligence for Industries of the Future: Beyond Facebook, Amazon, Microsoft and Google*. Springer, 2022, pp. 25–45.

[36] Steffen Kinkel, Marco Baumgartner and Enrica Cherubini. 'Prerequisites for the adoption of AI technologies in manufacturing–Evidence from a worldwide sample of manufacturing companies'. In: *Technovation* 110 (2022), p. 102375. ISSN: 0166-4972.

[37] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla and Carol Willing. 'Jupyter Notebooks – a publishing format for reproducible computational workflows'. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS Press. 2016, pp. 87–90.

[38] Henrik Kortum, Jonas Rebstadt, Tula Böschen, Pascal Meier and Oliver Thomas. 'Towards the operationalization of trustworthy AI: integrating the EU assessment list into a procedure model for the development and operation of AI-systems'. In: (2022). ISSN: 3885797208.

[39] Oliver Kramer. 'K-Nearest Neighbors'. In: *Dimensionality Reduction with Unsupervised Nearest Neighbors*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 13–23. ISBN: 978-3-642-38652-7. DOI: 10.1007/978-3-642-38652-7_2. URL: https://doi.org/10.1007/978-3-642-38652-7_2.

[40] Dominik Kreuzberger, Niklas Kühl and Sebastian Hirschl. 'Machine learning operations (mlops): Overview, definition, and architecture'. In: *IEEE access* (2023). ISSN: 2169-3536.

[41] Trevor LaViale. 'Deep Dive on KNN: Understanding and Implementing the K-Nearest Neighbor Algorithm'. In: *Tech Blog* (2023). Accessed: 27.04.2024.

[42] Noha Lim, Hongshik Ahn, Hojin Moon and James J. Chen. 'Classification of high-dimensional data with ensemble of logistic regression models'. In: *Journal of Biopharmaceutical Statistics* 20.1 (2009), pp. 160–171. ISSN: 1054-3406.

[43] Chin-Mei Liu and Charles Tzu-Chi Lee. 'Association of hearing loss with dementia'. In: *JAMA network open* 2.7 (2019), e198112–e198112.

[44] John A. Loonam and Joe McDonagh. 'Exploring top management support for the introduction of enterprise information systems: a literature review'. In: *Irish Journal of Management* 26.1 (2005), p. 163. ISSN: 1649-248X.

[45] Gilles Louppe. 'Understanding random forests: From theory to practice'. In: *arXiv preprint arXiv:1407.7502* (2014).

[46] Ben Lovejoy. *AirPods 3 improvements include better in-ear detection*. Accessed: 12.04.2024. 2021. URL: https://9to5mac.com/2021/10/20/airpods-3-improvements-in-ear-detection/.

[47] Sasu Mäkinen. 'Designing an open-source cloud-native MLOps pipeline'. In: *University of Helsinki* (2021).

[48] Sasu Mäkinen, Henrik Skogström, Eero Laaksonen and Tommi Mikkonen. 'Who needs MLOps: What data scientists seek to accomplish and how can MLOps help?' In: IEEE, pp. 109–112. ISBN: 1665444703.

[49] Wes McKinney. 'Data structures for statistical computing in Python'. In: vol. 445, pp. 51–56.

[50] Bacigál Michal. 'Design and Implementation of Machine Learning Operations'. In: (2024).

[51] Microsoft. *Visual Studio Code Documentation*. https://code.visualstudio.com/docs. Accessed: 05.05.2024. n.d.

[52] Md Nahiduzzaman, Md Julker Nayeem, Md Toukir Ahmed and Md Shahid Uz Zaman. 'Prediction of heart disease using multi-layer perceptron neural network and support vector machine'. In: IEEE, pp. 1–6. ISBN: 1728160405.

[53] A. Navia-Vázquez and E. Parrado-Hernández. 'Support vector machine interpretation'. In: *Neurocomputing (Amsterdam)* 69.13 (2006), pp. 1754–1759. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2005.12.118.

[54] World Health Organization et al. *World report on hearing*. World Health Organization, 2021.

[55] Eneko Osaba, Esther Villar, Jesus L. Lobo, Ibai Laña and Andries Engelbrecht. *Artificial Intelligence: Latest Advances, New Paradigms and Novel Applications*. BoD–Books on Demand, 2021. ISBN: 183962387X.

[56] Deven Panchal, Isilay Baran, Dan Musgrove and David Lu. 'MLOps: Automatic, Zero-touch and Reusable Machine Learning Training and Serving Pipelines'. In: IEEE, pp. 175–181. ISBN: 9798350319040.

[57] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein and Luca Antiga. 'Pytorch: An imperative style, high-performance deep learning library'. In: *Advances in neural information processing systems* 32 (2019).

[58] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss and Vincent Dubourg. 'Scikit-learn: Machine learning in Python'. In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830. ISSN: 1532-4435.

[59] Mary Poppendieck. 'Lean software development'. In: IEEE, pp. 165–166. ISBN: 0769528929.

[60] scikit-learn: machine learning in Python. *scikit-learn Documentation (n.d.). Accuracy score*. Accessed: 29.03.2024. URL: https://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score.

[61] scikit-learn: machine learning in Python. *scikit-learn Documentation (n.d.). Confusion matrix*. Accessed: 29.03.2024. URL: https://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix.

[62] scikit-learn: machine learning in Python. *scikit-learn Documentation (n.d.). Decision Trees. Missing Values Support.* Accessed: 05.05.2024. URL: https://scikit-learn.org/stable/modules/tree.html#tree-missing-value-support.

[63] scikit-learn: machine learning in Python. *scikit-learn Documentation (n.d.). Decision Trees. Tips on practical use.* Accessed: 05.05.2024. URL: https://scikit-learn.org/stable/modules/tree.html#tips-on-practical-use.

[64] scikit-learn: machine learning in Python. *scikit-learn Documentation (n.d.). Ensembles: Gradient boosting, random forests, bagging, voting, stacking*. Accessed: 29.03.2024. URL: https://scikit-learn.org/stable/modules/ensemble.html#forest.

[65] scikit-learn: machine learning in Python. *scikit-learn Documentation (n.d.). Ensembles: Gradient boosting, random forests, bagging, voting, stacking. Random forests and other randomized tree ensembles.* Accessed: 05.05.2024. URL: https://scikit-learn.org/stable/modules/ensemble.html#random-forests-and-other-randomized-tree-ensembles.

[66] scikit-learn: machine learning in Python. *scikit-learn Documentation (n.d.). Nearest Neighbors*. Accessed: 29.03.2024. URL: https://scikit-learn.org/stable/modules/neighbors.html#classification.

[67] scikit-learn: machine learning in Python. *scikit-learn Documentation (n.d.). Neural network models (supervised)*. Accessed: 29.03.2024. URL: https://scikit-learn.org/stable/modules/neural_networks_supervised.html.

[68] scikit-learn: machine learning in Python. *scikit-learn Documentation (n.d.). Support Vector Machines*. Accessed: 29.03.2024. URL: https://scikit-learn.org/stable/modules/svm.html#svm-classification.

[69] R. R Core Team. 'R: A language and environment for statistical computing'. In: (2013).

[70] Gilberto Recupito, Fabiano Pecorelli, Gemma Catolino, Sergio Moreschini, Dario Di Nucci, Fabio Palomba and Damian A. Tamburri. 'A multivocal literature review of mlops tools and features'. In: IEEE, pp. 84–91. ISBN: 1665461527.

[71] Guido Rossum. *Python reference manual*. Generic. 1995.

[72] Roopashri Shetty, M. Geetha, Dinesh U. Acharya and G. Shyamala. 'Data Preprocessing and Finding Optimal Value of K for KNN Model'. In: Springer, pp. 1–9.

[73] Georgios Symeonidis, Evangelos Nerantzis, Apostolos Kazakis and George A. Papakostas. 'MLOps-definitions, tools and challenges'. In: IEEE, pp. 0453–0460. ISBN: 1665483032.

[74] Amir Taherizadeh and Catherine Beaudry. 'An emergent grounded theory of AI-driven digital transformation: Canadian SMEs' perspectives'. In: *Industry and Innovation* 30.9 (2023), pp. 1244–1273. ISSN: 1366-2716.

[75] Matteo Testi, Matteo Ballabio, Emanuele Frontoni, Giulio Iannello, Sara Moccia, Paolo Soda and Gennaro Vessio. 'Mlops: A taxonomy and a methodology'. In: *IEEE Access* 10 (2022), pp. 63606–63618. ISSN: 2169-3536.

[76] Rhett S. Thomson, Priscilla Auduong, Alexander T. Miller and Richard K. Gurgel. 'Hearing loss as a risk factor for dementia: a systematic review'. In: *Laryngoscope investigative otolaryngology* 2.2 (2017), pp. 69–79. ISSN: 2378-8038.

[77] John Wilder Tukey. *Exploratory data analysis*. Vol. 2. Springer, 1977.

[78] Mirela Cătălina Türkeș, Ionica Oncioiu, Hassan Danial Aslam, Andreea Marin-Pantelescu, Dan Ioan Topor and Sorinel Căpușneanu. 'Drivers and barriers in using industry 4.0: a perspective of SMEs in Romania'. In: *Processes* 7.3 (2019), p. 153. ISSN: 2227-9717.

[79] Janviere Umurerwa and Maja Lesjak. *AI Implementation and Usage: A qualitative study of managerial challenges in implementation and use of AI solutions from the researchers' perspective*. Generic. 2021.

[80] Andrés Felipe Varón Maya. 'The state of MLOps'. In: (2021).

[81] Michael L. Waskom. 'Seaborn: statistical data visualization'. In: *Journal of Open Source Software* 6.60 (2021), p. 3021. ISSN: 2475-9066.

[82] Samar Wazir, Gautam Siddharth Kashyap and Parag Saxena. 'MLOps: A review'. In: *arXiv preprint arXiv:2308.10908* (2023).

[83] Samantha Werens and Jörg von Garrel. 'Implementation of artificial intelligence at the workplace, considering the work ability of employees'. In: *TATuP-Zeitschrift für Technikfolgenabschätzung in Theorie und Praxis* 32.2 (2023), pp. 43–49. ISSN: 2567-8833.

[84] Ian H. Witten and Eibe Frank. 'Data mining: practical machine learning tools and techniques with Java implementations'. In: *Acm Sigmod Record* 31.1 (2002), pp. 76–77. ISSN: 0163-5808.

[85] Geqi Yan, Wanying Zhao, Chaoyuan Wang, Zhengxiang Shi, Hao Li, Zhenwei Yu, Hongchao Jiao and Hai Lin. 'A comparative study of machine learning models for respiration rate prediction in dairy cows: Exploring algorithms, feature engineering, and model interpretation'. In: *Biosystems Engineering* 239 (2024), pp. 207–230. ISSN: 1537-5110.

# Appendix A

# Source Code

The master thesis repository, accessible via https://github.com/s372039/master-thesis, contains all the source code and configuration files used throughout the project. This repository is structured into three distinct directories:

#### System Management

Located in the system_management directory, this section includes Terraform and Puppet files used for configuring and managing the master server infrastructure.

#### Central Data Repository Application

The central_data_repository_application directory houses the application code for the central data repository. It includes a Flask API server setup and is containerized using Docker. This directory also contains Dockerfiles, docker-compose configurations, and CI/CD pipeline definitions.

#### In-Ear Detection Repository

The in-ear_detection_repository houses the Jupyter notebook (pilot_development.ipynb) that details the machine learning experiments conducted for in-ear detection. This directory also utilizes DVC to manage and version large datasets. Additional configuration files support the use of Git and DVC for version control and reproducibility.