TUNISIAN REPUBLIC
Ministry of Higher Education and Scientific Research
University of Carthage
National Institute of Applied Sciences and Technology

End of Year Project

submitted as part of the
End of Year Project Report
Specialty: **Software Engineering**

AI-powered Mobile Application for Breathing Sound Analysis and Disease Detection

Prepared by
Ahmed SILINI
Ghassen CHERIF
Mohamed Aziz BALTI

Defended on the **4th of May 2025**
in front of a jury composed of:

Dr. Sana HAMDI	**Dr. Wided MILED**
Supervisor	Examiner

Academic year: **2024 / 2025**

Dedication

This project is dedicated with deep gratitude and respect to those who have stood beside us throughout this journey.

To our families — your unwavering love, support, and encouragement have been the pillars upon which we've built not only this work, but our growth and perseverance. Your sacrifices, patience, and belief in our potential gave us strength when we needed it most.

To our supervisor, ***Dr. Sana HAMDI*** *— thank you for your insightful guidance, constructive feedback, and constant encouragement. Your trust in our abilities pushed us to do better and take ownership of our work. We are truly grateful for your mentorship and your commitment to helping us succeed.*

To our professors and the faculty at ***INSAT****, thank you for shaping our knowledge, skills, and mindset over the past years. Your dedication to education and innovation has inspired us to pursue excellence in everything we do.*

Finally, to each other — for the collaboration, resilience, and shared passion that carried this project forward. This work is a testament to teamwork and shared vision.

With heartfelt appreciation,
Ahmed SILINI
Ghassen CHERIF
Mohamed Aziz BALTI

Contents

List of Acronyms

AI	Artificial Intelligence
COPD	Chronic Obstructive Pulmonary Disease
COVID-19	Coronavirus Disease 2019
CLAP	Contrastive Learning Audio Processing
RAG	Retrieval-Augmented Generation
LLM	Large Language Model
CRISP-DM	Cross-Industry Standard Process for Data Mining
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
DL	Deep Learning
LlaMa	Large Language Model Meta AI
BERT	Bidirectional Encoder Representations from Transformers
BGE	BERT Generated Embeddings
MRR	Mean Reciprocal Rank
MAP	Mean Average Precision
UCD	User-Centered Design
BaaS	Backend-as-a-Service
WHO	World Health Organization
CDC	Centers for Disease Control and Prevention
API	Application Programming Interface
REST	Representational State Transfer
UI	User Interface
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
iOS	iPhone Operating System
VS Code	Visual Studio Code

List of Figures

List of Tables

General Introduction

The rapid evolution of artificial intelligence (AI) and mobile technologies has brought transformative changes across multiple domains, including healthcare. These advancements are now enabling the design of intelligent systems that support early disease detection, remote monitoring, and accessible diagnostics. One area of growing relevance is respiratory health, particularly in light of the increased attention to respiratory diseases following the COVID-19 pandemic and the continued burden of chronic conditions such as asthma, chronic obstructive pulmonary disease (COPD), and sleep apnea.

Despite the prevalence of such conditions, access to timely and affordable respiratory screening remains limited in many parts of the world. Specialized medical equipment and trained professionals are often required to conduct auscultation and interpret respiratory anomalies. This reality poses a significant barrier to early detection and long-term monitoring—two factors that are critical to improving patient outcomes. The proliferation of smartphones, however, opens new opportunities for building accessible, AI-powered health tools that can reach broader populations.

This end-of-year project seeks to address this challenge by designing and implementing a mobile application capable of analyzing breathing sounds and providing AI-assisted respiratory diagnostics. The system enables users to record audio samples using a smartphone microphone, which are then processed using machine learning models trained to classify respiratory anomalies such as wheezing, crackles, and rhonchi. Based on the classification results, the system offers a preliminary diagnostic suggestion. To enhance usability and trust, a conversational AI assistant is also integrated—allowing users to ask questions, receive explanations, and obtain medically grounded feedback.

The core objective of the project is to provide a portable and intelligent solution for respiratory disease detection, one that combines ease of use, technical robustness, and medical relevance. Achieving this goal involves contributions across multiple areas of computer science, including mobile development, signal processing, deep learning, backend architecture, and natural language processing.

The mobile application is built using a cross-platform framework, ensuring compatibility with both Android and iOS devices. It includes modules for secure user authentication,

audio recording, and diagnosis history tracking. The backend infrastructure, powered by a backend-as-a-service platform, supports encrypted data storage, API integration, and serverless functions for model inference. On the AI side, two main components are developed: (1) a contrastive learning model (CLAP) for respiratory sound classification, and (2) a lightweight retrieval-augmented generation (RAG) chatbot based on a quantized language model. These components are trained on publicly available datasets and fine-tuned to deliver fast, accurate, and explainable results to end users.

Beyond implementation, this report provides a structured exploration of the project lifecycle, from the initial definition of objectives to system evaluation. Chapter 1 introduces the general context, the underlying problem, and the proposed solution. Chapter 2 outlines the foundational concepts that support the system, including respiratory anomalies, deep learning principles, and mobile development strategies. Chapter 3 presents a review of related work in respiratory diagnostics and medical chatbots. Chapters 4 and 5 detail the application of the CRISP-DM methodology to both the classification model and the AI assistant. Chapter 6 focuses on implementation, highlighting system architecture, mobile development, backend integration, and the tools used.

Through this interdisciplinary work, we aim not only to demonstrate the feasibility of smartphone-based respiratory diagnostics, but also to contribute to the broader field of AI for public health. By combining modern machine learning techniques with accessible mobile interfaces, the project aspires to bring preliminary respiratory screening closer to users—anytime, anywhere.

Chapter 1

General Context

Introduction

Respiratory diseases, such as asthma, chronic obstructive pulmonary disease (COPD), and bronchitis, remain among the most common and impactful health challenges globally. Early detection and effective diagnosis are essential for reducing complications and improving patient outcomes. Traditionally, respiratory assessment relies heavily on auscultation—the process of listening to lung sounds through a stethoscope. While widely practiced, this technique depends greatly on clinical expertise and is inherently subjective.

As digital health tools become more accessible, especially through smartphones, there is growing potential to transform the way respiratory conditions are detected and monitored. In particular, the use of Artificial Intelligence (AI) enables automated analysis of lung sounds and medical data, providing faster and potentially more accurate support for both patients and clinicians.

This project introduces a mobile application that empowers users to monitor their respiratory health at home. The app allows users to record breathing sounds using their phone's microphone and upload personal medical documents (e.g., prescriptions, symptom reports, or discharge summaries). Through secure and privacy-preserving AI techniques, these inputs are analyzed to detect respiratory anomalies and provide informed assistance.

1.1. Project Background and Objectives

1.1.1. Project Context

The idea for this system is rooted in the limitations of traditional diagnostic tools and the underutilization of available clinical information. Public datasets, such as the ICBHI 2017 Respiratory Sound Database, have demonstrated the feasibility of using machine learning for automated respiratory sound classification. However, most existing approaches focus solely on audio data, ignoring valuable contextual clues that could be derived from a patient's medical history or symptoms.

This project builds upon these foundations by embracing a multimodal approach. It combines the power of deep audio models with textual understanding to interpret both lung sounds and patient-specific metadata. The mobile app becomes a bridge between clinical-grade respiratory screening and user-friendly, accessible diagnostics.

1.1.2. Problem Statement

Current solutions for respiratory anomaly detection face several limitations:

- **Single Modality Focus:** Many existing tools rely exclusively on sound data, overlooking important contextual information.
- **Subjectivity in Diagnosis:** Manual auscultation is prone to human error and variability between clinicians.
- **Lack of Interpretability:** Users often receive a diagnosis or prediction without any explanation or follow-up guidance.
- **Limited Accessibility:** Clinical tools are often not available outside hospitals or specialized environments.

This project addresses the following key question:

"How can we build an AI-powered mobile application that combines user-recorded lung sounds with clinical records to detect respiratory anomalies and provide intelligent, interactive feedback?"

1.1.3. Proposed Solution

We propose an integrated, user-friendly mobile application that performs end-to-end respiratory analysis using multimodal AI. The system operates as follows:

- **User Interaction:** The user records their breathing sounds through the app and optionally uploads medical documentation in PDF format.
- **Multimodal Embedding:** A pretrained CLAP (Contrastive Language-Audio Pretraining) model is used to embed both the audio signal and extracted text from the medical documents into a shared latent space, capturing a holistic representation of the patient's condition.
- **Respiratory Anomaly Detection:** Using these embeddings, the model classifies the type of lung sound (e.g., *normal*, *wheeze*, *crackle*, or *both*).
- **Clinical Reasoning:** The app then sends the output to a Large Language Model (LLM), which, supported by a Retrieval-Augmented Generation (RAG) system, infers possible underlying conditions and provides medically-informed responses.
- **Conversational Assistant:** The LLM acts as a virtual assistant that explains results, suggests next steps, and answers follow-up questions, helping users make sense of their health status.
- **Privacy and Security:** All user data is handled with strong encryption and processed locally or on secure servers in compliance with healthcare data regulations.

By combining sound interpretation, contextual analysis, and conversational intelligence in a single mobile interface, this solution aims to democratize respiratory diagnostics. It supports users with accessible insights while maintaining clinical relevance, and ultimately enhances both early detection and user understanding in a safe, secure manner.

1.2. Project Methodology

1.2.1. Methodology: CRISP-DM Methodology

During our project, we chose to adopt the *Cross-Industry Standard Process for Data Mining (CRISP-DM)* methodology. CRISP-DM is a well-known framework for data mining and ML projects. It offers detailed steps that cover the entirety of the project flow, from its business understanding to its deployment. The key steps defined by this framework that we went by are illustrated in Figure 1.2.1 and are as follows:

- **Business Understanding:** This includes defining the business needs and understanding the issues we are addressing with our project. This is also where we define our business goals and requirements.
- **Data Understanding:** This step includes the exploration and analysis of our available data, highlighting its different characteristics, identifying any potential biases it may have, and understanding our acquired data and the preprocessing steps we need to get it.
- **Data Preparation:** This is the step where we apply different preprocessing techniques to our acquired data. This might include preparing the data for modeling, ensuring it has suitable formats, etc.
- **Modeling:** This consists of selecting appropriate models for our task, training and fine-tuning them using the prepared data, and experimenting with different architectures, hyperparameters, and optimization techniques to enhance model performance.
- **Evaluation:** In this step, we evaluate the performance of the models using appropriate metrics for the specific tasks, and we assess their effectiveness in those tasks.
- **Deployment:** This step is dedicated to studying real-life scenarios where our chosen model might be needed, potentially leading to its deployment to make it available to users.

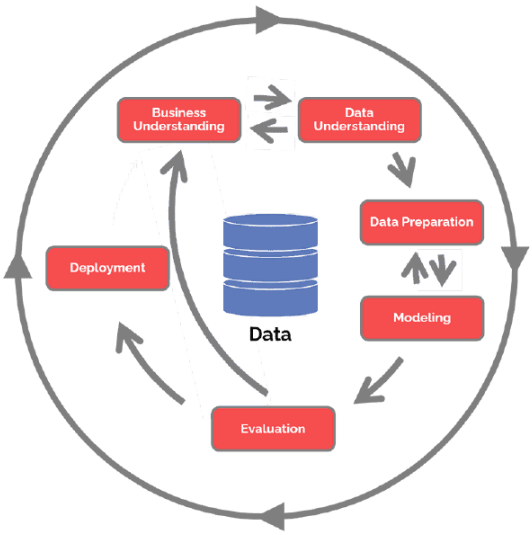

Figure 1.2.1: The different steps of the CRISP-DM methodology.

1.2.2. Project Planning (Gantt Chart)

Figure 1.2.2 illustrates the Gantt diagram detailing our project timeline from April 24th, to Mai, 18th.

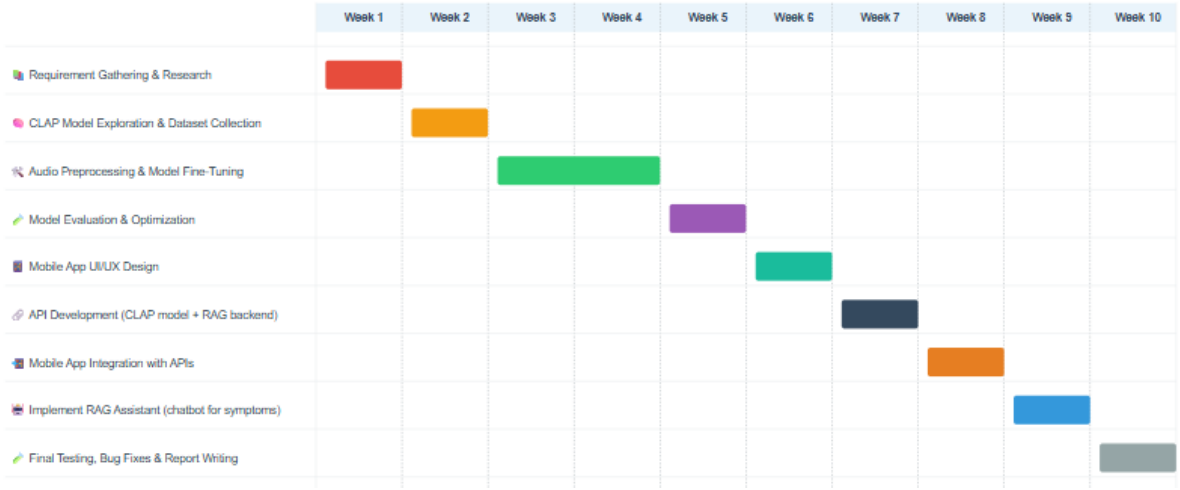

Figure 1.2.2: Project timeline represented with a Gantt chart.

Chapter 2

Business Understanding and Key Concepts

Introduction

Respiratory diseases are among the leading causes of morbidity and mortality worldwide. Conditions such as asthma, pneumonia, bronchitis, and chronic obstructive pulmonary disease (COPD) often present with distinct acoustic signatures in lung sounds. Early and accurate detection of these anomalies can significantly improve clinical outcomes, reduce hospital admissions, and lower healthcare costs. Traditional auscultation with a stethoscope, though effective, is highly subjective and dependent on clinical expertise. This project leverages deep learning and multimodal processing to automate the classification of respiratory sounds using both audio and clinical metadata, aiming to support clinicians with an objective and reproducible decision-support tool.

2.1. Project Domain – Respiratory Anomalies

2.1.1 Respiratory Anomalies and Their Significance

Respiratory anomalies are abnormal sounds heard during lung auscultation, which are indicative of underlying pulmonary conditions. Common anomalies include:

- **Wheezing:** A high-pitched, musical sound typically associated with narrowed airways and conditions such as asthma or COPD.
- **Crackles (Rales):** Discontinuous popping sounds heard during inhalation, often indicating fluid in the alveoli as seen in pneumonia or pulmonary fibrosis.

Importance of Early Detection in Healthcare

Early diagnosis of respiratory diseases through anomaly detection can:

- Enable timely medical intervention and prevent disease progression.
- Reduce the need for costly imaging or invasive diagnostic procedures.
- Facilitate remote diagnosis in telemedicine and resource-limited settings.
- Assist in large-scale screening, especially during pandemics or respiratory outbreaks.

This project's integration of both audio recordings and patient metadata (extracted from clinical PDF reports) aims to emulate real-world diagnostic scenarios where doctors rely on both auscultatory findings and patient history for diagnosis.

2.2. Deep Learning Concepts

Natural Language Processing

Natural Language Processing (NLP) is a field of artificial intelligence that develops algorithms to enable machines to process, interpret, and generate human-like text. NLP empowers computers to perform language-related tasks traditionally requiring human intelligence, including Language translation, Sentiment analysis, Text classification, Information extraction

A key advancement in NLP has been the development of *word embeddings* - numerical representations that capture semantic meaning and contextual relationships between words. These embeddings form the foundation for modern language understanding systems.

Word Embeddings

Word embeddings are vector representations of words. The challenge they aim to address is the ability to represent similar words—i.e., words that often appear in similar contexts—with similar vectors, where the vector directions also carry semantic meaning.

While word vectors had been previously used in information retrieval, one of their most pivotal applications was their integration into neural networks. The need for word embeddings became evident when researchers began feeding words into Neural Networks (NNs).

Several encoding strategies were considered:

- **Character-based representation:** Training a character-level model consumes a large portion of the network's capacity to simply learn valid word forms, potentially limiting its ability to learn higher-level features such as context or semantics.
- **Dictionary-based representation:**
  - *One-hot encoding:* Each word is represented as a vector of length N (the vocabulary size) with all entries set to 0 except for a 1 in the position corresponding to that word. This approach does not encode similarity and leads to sparse and high-dimensional data, which is inefficient.
  - *Index-based encoding:* Assigning a unique integer to each word implies false assumptions—e.g., that adjacent indices mean semantic closeness.

These limitations highlighted the need for richer numerical representations that reflect both word meaning and context. Additionally, words can have multiple meanings in different contexts, motivating dynamic or multi-context embeddings.

Popular Embedding Models

- **Word2Vec:** This model learns embeddings by predicting context words given a target word (Skip-Gram) or predicting a word from its context (Continuous Bag-of-Words, or CBOW). Word2Vec optimizes training speed and scalability through techniques like *Negative Sampling*. It excels at capturing local context and performs well on tasks like word prediction.
- **GloVe (Global Vectors):** GloVe is an unsupervised learning algorithm that constructs a co-occurrence matrix capturing the frequency with which word pairs appear together in a corpus. It then factorizes this matrix to produce dense word vectors. GloVe captures both local and global context but may struggle with tasks involving polysemous words (words with multiple meanings).

Vector Analogies

Word embeddings allow for interesting arithmetic operations that reveal semantic relationships. A well-known example illustrates this with gender analogies:

$$\texttt{vector("king")} - \texttt{vector("man")} + \texttt{vector("woman")} \approx \texttt{vector("queen")}$$

This vector analogy demonstrates that the learned embedding space encodes meaningful relationships such as gender differences, enabling analogical reasoning.

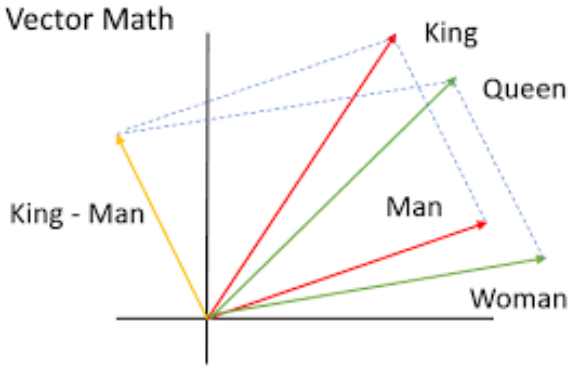

Figure 2.2.1: The classical *king - man + woman = queen* example of word embeddings.

Word embeddings are widely used in NLP tasks such as sentiment analysis, machine translation, and question answering. Interestingly, research has shown that source code exhibits similar co-occurrence patterns to natural language. As a result, vector representations of tokens in programming languages have also been explored using similar embedding techniques.

Audio Embeddings

Audio embeddings are numerical representations of audio samples that capture essential characteristics and features of the sound. These embeddings are widely used in applications such as audio classification, music recommendation, and speech recognition. Essentially, an audio embedding transforms complex audio data into a fixed-size vector, making it easier for machine learning models to analyze and compare different audio samples. This compact vector encodes information such as pitch, rhythm, timbre, and other auditory features.

The process of generating audio embeddings typically involves several steps:

- **Pre-processing:** The raw audio signal is first pre-processed. Common steps include noise reduction, normalization, and segmentation into smaller chunks or frames.
- **Feature extraction:** From the pre-processed audio, features are extracted using techniques such as:
  - *Short-Time Fourier Transform (STFT)*
  - *Mel-Frequency Cepstral Coefficients (MFCC)*
  - *Mel-spectrograms*

  These methods help capture the frequency content of the audio over time, allowing models to encode meaningful auditory information.
- **Embedding generation:** Machine learning models such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) are applied to the extracted features to produce dense vector embeddings.

For instance, CNNs may learn to recognize different instruments or environmental sounds, while RNNs can capture temporal patterns in speech or music. The resulting embedding is a fixed-size vector that summarizes the most relevant properties of the audio sample.

These audio embeddings are then used for downstream tasks such as clustering, retrieval, or classification, enabling systems to interpret and manipulate audio data effectively in various contexts.

Large Language Models

A language model is a statistical representation of a human language, capable of estimating the likelihood of word sequences based on the text data it was trained on.

In their most advanced form, *Large Language Models* (LLMs) are a fusion of feedforward neural networks and transformers. LLMs are characterized by three major features:

- **Scale:** LLMs are trained on extremely large datasets and have a vast number of parameters, often exceeding one billion.
- **General-purpose capabilities:** These models are capable of performing a wide range of tasks, including language generation, summarization, translation, question answering, and even solving mathematical problems.
- **Fine-tuning potential:** Although LLMs are trained on general data, they can be fine-tuned for specific tasks using much smaller, task-specific datasets.

LLMs leverage Deep Learning (DL) techniques—particularly the transformer architecture—to capture the complex dependencies and hierarchical structures present in natural language. Notable examples include *GPT-3* (Generative Pre-trained Transformer 4) and *Llama 3.2*.

The emergence of transformers has played a critical role in the development of LLMs, enabling them to understand and generate human-like text with high fluency and contextual awareness. Transformers are now the standard architecture underlying most modern LLMs.

Transformers

Transformers are a Neural Network architecture that have revolutionized Natural Language Processing (NLP). Neural Networks are capable of analyzing vast amounts of data, including images, videos, text, and audio. Each data type is best suited for specific neural architectures; for instance, Convolutional Neural Networks (CNNs) are optimized for visual data.

CNNs simulate how the human brain processes visual information and are particularly powerful for tasks like object detection, facial recognition, and handwritten text recognition. However, for language-related tasks, CNNs were not the most effective.

To address this, Recurrent Neural Networks (RNNs) were developed for handling sequential data, as shown in Figure 2.2.2. RNNs could capture temporal dependencies and were widely used for language modeling tasks. However, they struggled with long-term dependencies due to issues like the vanishing and exploding gradient problems. This made them difficult to train and limited their ability to remember information across long sequences.

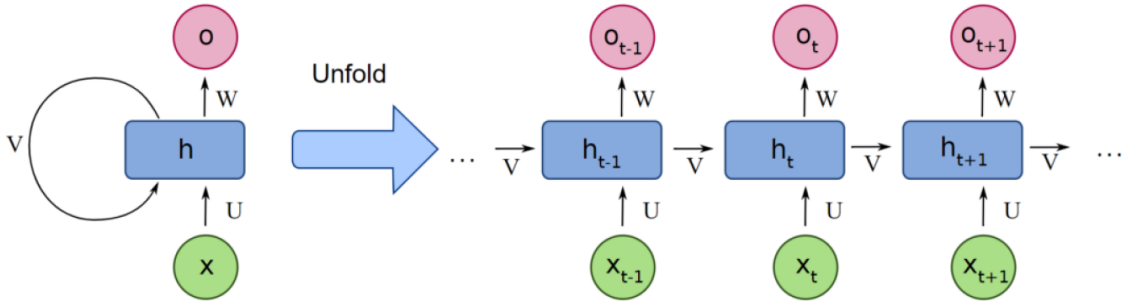

Figure 2.2.2: RNN Architecture

In 2017, the introduction of Transformers by researchers from Google and the University of Toronto marked a significant breakthrough. Initially designed for translation tasks, the Transformer architecture has since become the standard for modern NLP models.

Unlike RNNs, Transformers can be easily parallelized, which allows for the training of extremely large models. The architecture of the Transformer is illustrated in Figure 2.2.3.

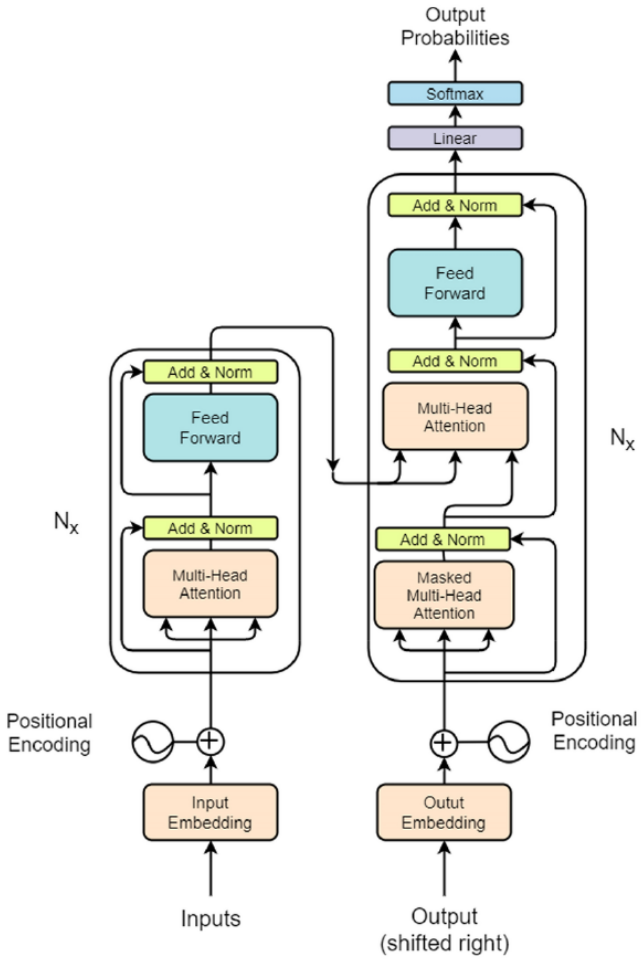

Figure 2.2.3: Transformer Diagram from the original paper "Attention is All You Need" [1]

Transformers are based on three key concepts:

- **Positional Encodings:** Since Transformers do not process input sequences in order like RNNs, positional encodings are added to each word to preserve the sequence information. Originally, sinusoidal functions were used to generate these encodings, and the model learns to interpret them during training.
- **Attention:** Introduced in 2015, the attention mechanism assigns a contextual weight to each word in the input, allowing the model to focus on the most relevant parts of a sentence. This is not based solely on individual words, but on the entire context.
- **Self-Attention:** This is a special form of attention where the model considers the same input sequence to compute the relationships between its elements. Self-

attention helps capture dependencies across the whole sequence simultaneously, ensuring that important information is retained throughout the process.

Not all Transformer models are the same. There are many architectural variations designed for specific tasks, such as BERT for masked language modeling, GPT for autoregressive generation, and T5 for text-to-text tasks. Despite these differences, they all share the same foundational principles of attention and positional encoding.

Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) [3] is a hybrid architecture that combines the strengths of retrieval-based and generative models. Instead of relying solely on pre-trained knowledge, RAG enhances a generative language model by incorporating external documents retrieved from a large corpus in real time.

The architecture consists of two main components:

- **Retriever:** Fetches relevant documents or passages based on the input query, typically using dense or sparse retrieval methods.
- **Generator:** A language model that conditions its output on both the input query and the retrieved documents to generate accurate and informative responses.

RAG is particularly useful in knowledge-intensive tasks such as question answering and summarization, where grounding the output in factual information is crucial.

CLAP (Contrastive Language-Audio Pretraining)

CLAP is a neural network model designed to learn joint representations of audio and natural language. By employing contrastive learning techniques, CLAP aligns audio signals with their corresponding textual descriptions in a shared embedding space. This approach enables the model to perform tasks such as zero-shot audio classification, retrieval, and captioning without the need for task-specific training.

The architecture of CLAP consists of two main components:

- **Audio Encoder:** Processes audio inputs, typically using a SWIN Transformer, to extract meaningful features from log-Mel spectrograms.

- **Text Encoder:** Utilizes models like RoBERTa to convert textual descriptions into feature representations.

Both audio and text features are projected into a common latent space, allowing the model to measure similarity between audio clips and textual inputs effectively. This design facilitates various applications, including audio-text retrieval and zero-shot classification, by leveraging the semantic alignment between modalities.

2.3. Mobile Development Concepts

Mobile development refers to the design and creation of software applications intended to operate on mobile devices such as smartphones and tablets. In the context of this project, mobile development serves as the foundation for the user-facing component of the system, enabling users to interact with diagnostic features, access results, and consult the AI assistant. This section introduces the key mobile development concepts relevant to the architecture used in this work.

Cross-Platform Development

Cross-platform development is a software development approach that allows a single application codebase to be deployed on multiple operating systems, most notably Android and iOS. This reduces redundancy and development effort while maintaining consistency in functionality and design. It is especially beneficial in healthcare applications, where uniform user experience and rapid iteration are critical.

Modular Mobile Architecture

A modular architecture divides the application into independent functional units, such as user interface components, business logic, and data services. This separation of concerns improves scalability, testability, and long-term maintainability. It also supports incremental feature integration—particularly important in complex systems like diagnostic platforms.

User-Centered Design in Health Applications

Mobile health applications must prioritize accessibility, clarity, and ease of use. User-centered design (UCD) involves tailoring the interface to user needs, using intuitive navigation, medically neutral colors, and readable typography. These design principles are

especially important in applications intended for patients, where cognitive load and clarity directly impact usability.

Data Privacy and Security Principles

Mobile applications that handle health-related data must comply with strict privacy and security guidelines. Core concepts include local data encryption, secure authentication flows, permission-based access to device hardware (e.g., microphone), and encrypted communication channels. These safeguards help ensure that personal health information is protected throughout the diagnostic process.

Real-Time and Asynchronous Interaction

Many mobile applications rely on real-time feedback and asynchronous operations—such as sending diagnostic data to a server and waiting for results. Concepts such as event-driven programming, state management, and asynchronous networking are fundamental to building responsive, interactive experiences without blocking the user interface.

Summary

This section introduced the theoretical foundations of mobile development as they apply to healthcare systems. Concepts such as cross-platform development, modular architecture, security, and user-centered design provide the groundwork for implementing reliable and accessible diagnostic tools on mobile platforms. These principles informed the design decisions detailed later in the implementation chapter.

Conclusion

This chapter introduced the key domains that underpin the design of our AI-powered respiratory diagnosis system. These domains—medical anomalies, deep learning, and mobile development—serve as the conceptual foundation for the project's implementation.

First, we explored the nature of respiratory anomalies and their clinical significance, highlighting sounds such as wheezing, crackles, and stridor. These sounds provide early indicators of diseases like asthma, pneumonia, and COPD. Understanding their acoustic characteristics is essential for developing diagnostic algorithms that can detect them automatically.

Second, we reviewed core deep learning concepts, with a focus on audio classification using models such as CLAP, as well as the use of large language models (LLMs) for natural language understanding. These AI tools enable the system to both analyze user-recorded audio and provide conversational explanations and guidance through the integrated assistant.

Finally, we presented key mobile development concepts relevant to our implementation. Cross-platform development, modular architecture, responsive design, and data privacy principles ensure that the system remains accessible, secure, and user-friendly across devices. These concepts support the delivery of accurate, real-time diagnostics directly through a mobile interface.

Together, these foundations—clinical, algorithmic, and technical—set the stage for the practical implementation of our system, detailed in the chapters that follow.

Chapter 3

State-of-the-art and related work

Introduction

Respiratory sound classification is a key component in the development of AI-assisted tools for pulmonary disease screening. This section reviews existing methodologies used for analyzing lung sounds, outlines performance on a common benchmark dataset, and identifies key limitations in current approaches. We conclude by presenting our contribution, which addresses these gaps through a multimodal, deployment-ready framework.

3.1. Overview of Respiratory Sound Classification

Research in respiratory disease detection using lung auscultation audio has expanded rapidly, with significant emphasis on early, non-invasive diagnosis. Classical approaches have relied on signal processing techniques like MFCCs and wavelets combined with shallow classifiers (e.g., SVMs, Random Forests). More recent studies employ deep learning — particularly convolutional and recurrent architectures — trained directly on spectrograms or raw waveforms.

3.2. The ICBHI Dataset and Prior Work

The ICBHI 2017 Respiratory Sound Database is a benchmark dataset containing over 5.5 hours of annotated lung sounds from 126 patients, labeled for the presence of crackles, wheezes, both, or normal [**?**]. Several studies have evaluated models on this dataset with a focus on binary or multi-class classification.

3.3. Comparative Performance on ICBHI Dataset

Model	**Input**	**Classes**	**F1 / Acc.**	**Notes**
Perna et al. (2019) [5]	MFCC + CNN	3	85.3% (Acc.)	Spectrogram-based CNN model
Demir et al. (2021) [6]	ResNet + MFCC	3	88.7% (F1)	Ensemble of CNNs trained on MFCC features
Ibrahim et al. (2020) [7]	BiLSTM	3	84.5% (F1)	Temporal sequence modeling using recurrent layers
Aygun et al. (2022) [8]	EfficientNet + SpecAugment	3	90.2% (Acc.)	CNNs with spectrogram augmentation techniques

Table 3.1: Comparison of respiratory sound classification models on the ICBHI dataset.

Note: Differences in dataset splits, class definitions, and preprocessing methods make direct comparison approximate.

3.4. Limitations in Existing Approaches

Despite high reported accuracies, most existing studies suffer from:

- **Modality Limitation:** Only use audio data; no use of patient context.
- **Device Heterogeneity:** Include both stethoscope and microphone recordings, reducing consistency.
- **Lack of Deployment Focus:** No exploration of real-time, user-facing APIs or interfaces.

3.5. Our Contribution and Innovation

Our work advances the field with:

- **Microphone-only Data:** More relevant to smartphone-based real-world recordings.
- **Textual Metadata Integration:** Clinical descriptions extracted from PDFs enhance context.
- **Multimodal CLAP Embedding:** Fuses audio and text for richer, patient-aware predictions.
- **End-to-End Deployment:** A FastAPI server enables immediate, interactive use for real scenarios.

Conclusion

This section has explored the state of respiratory sound classification, highlighting both the strengths and blind spots of recent methods. While performance metrics on the ICBHI dataset are promising, most models overlook practical constraints such as recording device variation and lack of contextual patient data. Our multimodal, deployment-ready system directly addresses these issues, positioning it for real-world use in accessible respiratory diagnostics.

Chapter 4

Respiratory Sound Classification

Introduction

This section outlines the complete data mining workflow used to develop the respiratory sound classification system, following the CRISP-DM (Cross-Industry Standard Process for Data Mining) methodology. Each sub-section corresponds to a phase in the CRISP-DM cycle—from business understanding and data acquisition to modeling, evaluation, and deployment. Particular emphasis is placed on handling multimodal data, where audio recordings of lung sounds are complemented by metadata extracted from clinical PDF reports. The use of CLAP (Contrastive Language-Audio Pretraining) enables effective fusion of these modalities for downstream classification. This structured approach ensures the development is methodical, clinically relevant, and technically robust.

4.1. Business Understanding

4.1.1. Business Objectives

This project aims to classify respiratory conditions by leveraging multimodal data — specifically, lung auscultation audio and patient metadata (e.g., symptoms, notes). To accomplish this, we employ the CLAP [10] (Contrastive Language-Audio Pretraining) architecture to embed both modalities into a shared latent space. These joint embeddings are then passed to a lightweight classifier to predict respiratory sound categories such as *normal*, *wheeze*, or *crackle*.

This approach supports healthcare professionals by:

- Enhancing diagnostic accuracy using combined audio-text context.
- Reducing time and subjectivity in manual auscultation interpretation.
- Providing AI-driven assistance in remote or resource-limited healthcare settings.

Stakeholders include:

- **Clinicians:** Benefit from real-time AI-supported diagnostics.
- **Patients:** Gain earlier and more accurate diagnoses.
- **Medical Device Manufacturers:** Can integrate the classifier into stethoscope hardware.
- **Regulatory Authorities:** Ensure compliance and certification for clinical deployment.
- **Research and Development Teams:** Responsible for system design and performance.

4.1.2. Project Goals

- Use CLAP to generate multimodal embeddings from respiratory audio and patient text descriptions.
- Train a high-performing classifier on these embeddings to predict diagnostic categories.
- Ensure clinical interpretability and real-time deployment readiness.

4.1.3. Constraints and Risks

- **Data Privacy:** Must comply with HIPAA/GDPR when processing patient metadata.
- **Multimodal Alignment:** Embeddings from CLAP must capture relevant diagnostic features from both audio and text.
- **Generalization:** Risk of underfitting due to class imbalances between the labels "wheeze" , "crackles" , "normal" , "both".
- **Clinical Interpretability:** Must justify predictions for clinician trust and regulatory approval.

Summary

This project proposes a novel application of CLAP for multimodal fusion in respiratory sound classification. By embedding both patient audio and textual information into a shared space and using a downstream classifier, the system enables accurate, fast, and explainable predictions. Success depends on a robust technical pipeline, regulatory alignment, and demonstrated clinical utility.

4.2. Data Understanding

4.2.1. Data Sources

The primary dataset used for training the model is the **ICBHI 2017 Respiratory Sound Database**, which includes:

- **Audio recordings** of lung sounds collected from 126 patients using an electronic stethoscope.
- **Metadata files**, such as demographic information and clinical diagnoses.
- **Segmentation annotations** indicating labeled intervals of respiratory events (inspiration, expiration).

4.2.2. Loaded Data

Three key components were extracted:

1. **Diagnosis Labels:** A CSV file (`patient_diagnosis.csv`) mapping patient IDs to their respiratory condition (e.g., asthma, COPD, pneumonia).
2. **Demographic Metadata:** A text file containing age, sex, BMI, and child-specific attributes like height and weight.
3. **Audio Files:** Multiple WAV files per patient, capturing respiratory cycles in varying environments and conditions.

4.2.3. Exploratory Analysis

The following steps were conducted to better understand the data:

- Inspected class distribution to identify imbalance (e.g., overrepresentation of "healthy" vs. "COPD").
- Visualized audio signal characteristics such as waveform length, sampling rate consistency, and signal-to-noise ratio.
- Merged demographic and diagnosis data by patient ID to facilitate multimodal embedding.
- Generated descriptive sentences per patient from structured metadata, such as: *"Patient 101 is a 65-year-old male. The adult has a BMI of 27.3 kg/m².*"

4.2.4. Challenges Identified

- **Missing Data:** Some entries lack age, BMI, or diagnosis labels, requiring filtering or imputation.
- **Class Imbalance:** Certain respiratory conditions are underrepresented, potentially biasing the classifier.
- **Variable Audio Lengths:** Recording durations vary significantly, requiring padding or trimming during preprocessing.
- **Data Heterogeneity:** Audio captured in uncontrolled environments introduces noise and inconsistency.

Summary

The dataset provides a rich combination of audio and clinical metadata, which is essential for multimodal learning. By using CLAP to embed both the auscultation sounds and textual descriptions, the project exploits this diversity to improve classification performance. A thorough understanding of the dataset's structure and limitations guided preprocessing and model design choices.

4.3. Data Preparation

Overview

To enable the use of the CLAP (Contrastive Language-Audio Pretraining) model, both audio signals and textual descriptions were preprocessed and aligned into a format suitable for joint embedding. Data preparation involved several key stages: cleaning, transformation, feature generation, and multimodal input construction.

4.3.1. Audio Preprocessing

- **Loading:** WAV audio files were loaded using `torchaudio` with consistent settings across all samples.
- **Resampling:** All audio signals were resampled to 48 kHz to match the input requirements of the CLAP model.
- **Normalization:** Amplitude normalization was applied to reduce variability in recording loudness.
- **Device Filtering:** Only audio recorded using a microphone device was retained; recordings captured via stethoscope or user-recorded samples were excluded to ensure consistency and realistic ambient conditions.

4.3.2. Text Preprocessing

- **Demographic Metadata Integration:** Patient age, sex, and physical metrics were combined into a coherent textual description.
- **Description Generation:** A custom function generated natural language sentences, e.g., *"Patient 103 is a 7-year-old female child weighing 22 kg and measuring*

120 cm in height."

- **Tokenization:** Descriptions were tokenized using the CLAP processor to match the format of its language encoder.
- **Missing Value Handling:** Patients missing critical metadata fields were assigned defaults.

4.3.3. Multimodal Pair Construction

Each input sample consisted of a synchronized multimodal pair:

1. A preprocessed 5-second audio tensor representing a single breathing cycle.
2. A text description specific to the patient from whom the audio was collected.

These were fed into the CLAP model to generate aligned audio-text embeddings in a shared latent space.

4.3.4. Label Encoding

- Respiratory condition labels (e.g., crackles, wheezes, normal) were derived from expert annotations per breathing cycle.
- Labels were encoded using `LabelEncoder` from `scikit-learn` to obtain numerical class indices.
- Ambiguous or unlabeled cycles were excluded to avoid introducing noise in the training process.

4.3.5. Dataset Splitting

- An official dataset split provided by the dataset curators was used to separate training and test sets.
- This ensured reproducibility and comparability with prior work.
- Patient-level disjointness was maintained: no subject appeared in both training and test sets, preventing data leakage.

Summary

The preprocessing pipeline standardized and aligned audio and text inputs for CLAP-based embedding. Filtering by recording device, adherence to the official train/test split, and careful label and segmentation handling ensured high-quality inputs for downstream classification.

4.4. Modeling

Overview

The core modeling strategy involves using the CLAP (Contrastive Language-Audio Pre-training) model to transform heterogeneous input data — audio signals and textual patient metadata — into a unified latent representation. These embeddings are then passed through a custom multi-layer neural network for classification.

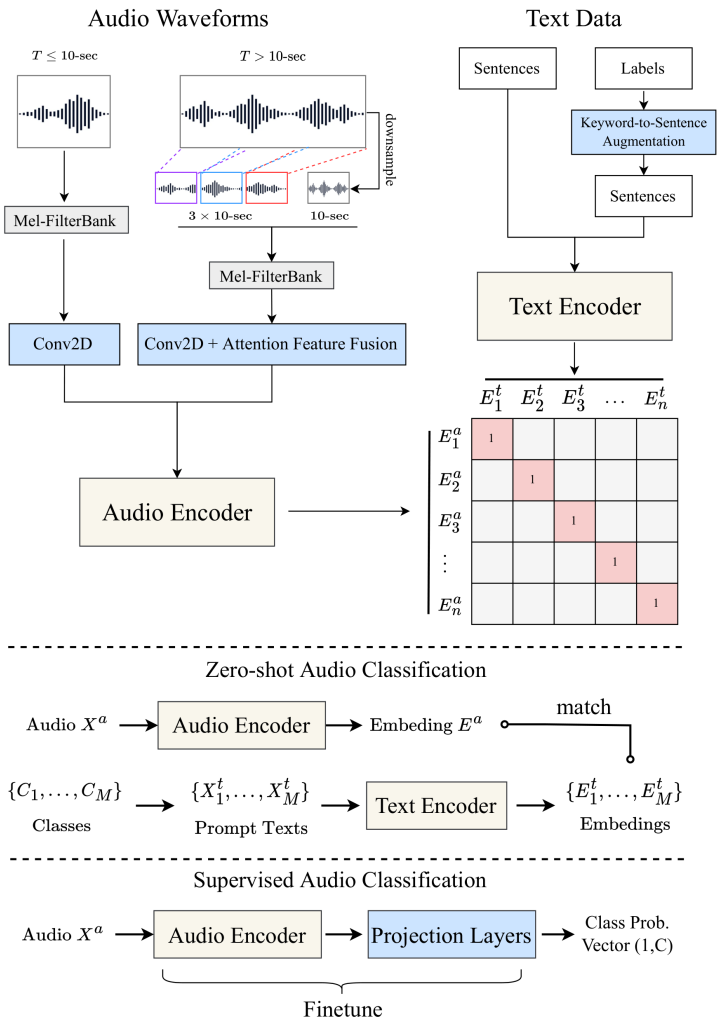

Figure 4.4.1: CLAP model architecture used for embedding audio and text.

4.4.1. CLAP: Multimodal Embedding Model

CLAP is a transformer-based model trained to align audio and text modalities in a shared embedding space. It consists of two separate encoders:

- **Audio Encoder:** Processes raw waveforms into feature embeddings using convolutional layers followed by transformers.
- **Text Encoder:** A transformer-based language model (similar to BERT) that embeds textual patient metadata.

The output of each encoder is a fixed-length vector, and CLAP is pretrained using contrastive learning to pull semantically aligned audio-text pairs closer in the embedding space.

Embedding Generation for Classification For this task:

- Each breathing cycle was segmented from the audio and paired with patient text data.
- These inputs were passed through the CLAP model to generate audio and text embeddings.
- The embeddings were concatenated into a single feature vector.
- This fused embedding was used as input to a downstream neural classifier.

4.4.2. Classifier Architecture

The classifier is a deep feedforward neural network with batch normalization, dropout regularization, and ReLU activations:

- **Input:** Fused embedding vector of dimension d (audio + text).
- **Layer 1:** Linear(d, 512) $\rightarrow$ BatchNorm $\rightarrow$ ReLU $\rightarrow$ Dropout(0.5)
- **Layer 2:** Linear(512, 256) $\rightarrow$ BatchNorm $\rightarrow$ ReLU
- **Layer 3:** Linear(256, 64) $\rightarrow$ BatchNorm $\rightarrow$ ReLU
- **Output:** Linear(64, 3) $\rightarrow$ Softmax over respiratory classes (normal, crackles, wheezes)

4.4.3. Training Procedure

- **Loss Function:** Cross-entropy loss for multi-class classification.
- **Optimizer:** AdamW with initial learning rate of 1×10^{-3} and weight decay of 1×10^{-3}.
- **Learning Rate Scheduler:** CosineAnnealingLR with $T_{\max} = 10$.
- **Epochs:** Trained for 100 epochs with early stopping based on validation loss.
- **Batch Size:** Mini-batches of 32 samples for efficient GPU usage.
- **Fine-tuning Strategy:** CLAP weights were frozen during classifier training to preserve pretrained semantics and reduce training time.

4.4.4. Model Variants and Experiments

Different modeling choices were explored to optimize performance:

- **Fusion Strategy:** Simple concatenation of audio and text embeddings.
- **Embedding Size:** Tested with different encoder output dimensions.
- **Depth and Width of Classifier:** Adjusting hidden layer sizes and number of layers.
- **Freezing vs. Fine-tuning CLAP:** Partial fine-tuning yielded marginal improvements but increased training time.

Summary

The modeling pipeline effectively combines audio and textual modalities through CLAP embeddings. A deep, regularized classifier leverages these representations for accurate respiratory sound classification. This architecture strikes a balance between expressiveness and efficiency, supporting real-time inference and deployment on modest hardware.

4.5. Evaluation

Overview

The model was evaluated on a held-out test set of 1,732 breathing cycles. Each sample included a fused CLAP embedding of audio and patient metadata. The classification targets were `normal`, `crackles`, and `wheezes`.

4.5.1. Performance Metrics

- **Accuracy:** 57.10%
- **Macro F1-score:** 0.53
- **Weighted F1-score:** 0.57
- **Macro Precision:** 0.55
- **Macro Recall:** 0.52

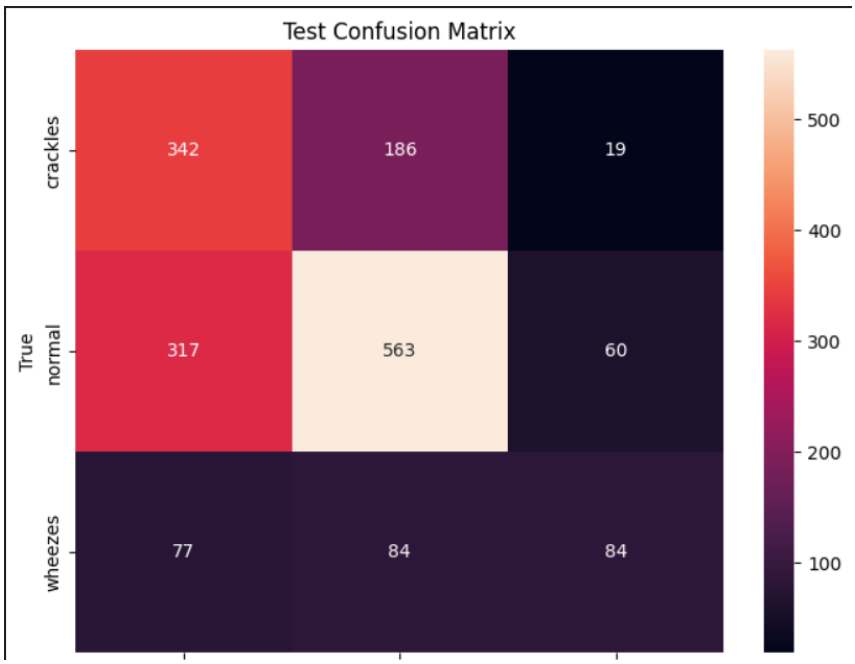

Figure 4.5.1: Test confusion matrix for respiratory sound classification.

Class	**Precision**	**Recall (Sensitivity)**	**F1-score**	**Support**
Crackles	0.46	0.63	0.53	547
Normal	0.68	0.60	0.64	940
Wheezes	0.52	0.34	0.41	245

Table 4.1: Class-wise classification report.

4.5.2. Sensitivity and Specificity

- **Sensitivity (Recall):**
  - Crackles: 0.625
  - Normal: 0.599
  - Wheezes: 0.343
  - **Average:** 0.392
- **Specificity:**
  - Crackles: 0.668
  - Normal: 0.659
  - Wheezes: 0.947
  - **Average:** 0.568

4.5.3. Observations

- The model performs best on the `normal` class, with an F1-score of 0.64.
- `Wheezes` are the hardest to classify (F1-score: 0.41), due to fewer examples and overlapping acoustic patterns with crackles.
- Specificity is highest for the `wheezes` class, indicating few false positives.

Summary

The CLAP-based classifier shows promising overall accuracy and solid recall for crackles and normal sounds. Its lower sensitivity for wheezes suggests room for improvement through better balancing, augmentation, or fine-tuning the audio encoder. Still, the model maintains interpretability and scalability, aligning with the clinical goals of the system.

4.6. Deployment (Model Serving)

4.6.1. Overview

To enable real-time inference, the trained multimodal CLAP-based classifier was deployed using a FastAPI web server. This server exposes a RESTful endpoint allowing users to

upload a WAV audio file along with a patient information file in PDF format. The server extracts text from the PDF and combines it with the audio input to produce a diagnostic prediction.

4.6.2. File Structure

The deployment folder contains the following core components:

- `main.py` – FastAPI application containing preprocessing, inference, and routing logic.
- `model.pt` – Trained PyTorch classifier model weights.
- `label_encoder.pkl` – Serialized `LabelEncoder` used to decode class indices into labels.
- `requirements.txt` – Specifies required Python libraries including `torch`, `transformers`, `PyMuPDF`, and `fastapi`.
- `Dockerfile` – Enables containerization of the app for consistent and scalable deployment.

4.6.3. API Endpoints

Two endpoints are exposed:

- `GET /` – Basic health check returning a simple greeting.
- `POST /predict` – Accepts:
  - A WAV audio file.
  - A PDF file containing patient information (e.g., age, sex, symptoms).

  and returns a JSON response containing the predicted diagnostic label.

4.6.4. Processing Pipeline

Upon receiving a request:

1. The audio file is validated, loaded, resampled to 48 kHz, converted to mono, and trimmed to a maximum of 8 seconds.

2. The PDF file is parsed using `PyMuPDF` (`fitz`) to extract text-based metadata (e.g., "Patient is Male 36 , with BMI...").

3. Audio and extracted text are processed using the CLAP processor to generate embeddings.

4. Embeddings are passed to the trained classifier to generate class probabilities.

5. The most probable class is decoded using `LabelEncoder` and returned.

4.6.5. Robustness and Error Handling

The API includes robust validation and fallback strategies:

- Rejects non-WAV or non-PDF inputs.
- Handles corrupted or overly long audio clips.
- Gracefully processes empty or unreadable PDFs.
- Returns descriptive error messages and HTTP status codes for all edge cases.

Conclusion

This section outlined the full CRISP-DM pipeline used to develop a multimodal respiratory sound classifier. By combining lung audio with patient metadata extracted from PDFs, the system leveraged CLAP embeddings for accurate, context-aware predictions. The final model was deployed using FastAPI, enabling real-time and scalable inference suitable for clinical settings.

Chapter 5

LLM-Based AI Assistant (RAG)

Introduction

This chapter explores the development of an intelligent assistant designed to support users with questions related to respiratory health. By combining language understanding with access to trusted medical information, the assistant aims to deliver clear, helpful, and relevant responses in real time. Our focus is on creating a system that is not only powerful but also accessible and responsible.

In the following sections, we walk through the key stages of building this assistant—from understanding the problem and preparing the data, to designing the system, evaluating its performance, and deploying it for real-world use.

5.1. Business Understanding

The AI assistant, built using a Large Language Model (LLM) enhanced with Retrieval-Augmented Generation (RAG), is designed to provide users with accurate, contextual information about respiratory symptoms and diseases through natural language interaction. This tool complements the audio classification system by enabling users to ask questions and receive clear explanations and guidance related to their respiratory health.

Project Goals

- **Deliver Clear Medical Information:** Provide understandable explanations about respiratory anomalies such as wheezing and crackles detected by the system.
- **Support Decision-Making:** Assist both patients and doctors in interpreting symptoms and accessing relevant medical knowledge to make better-informed decisions.
- **Enhance Accessibility:** Offer users immediate and easy access to reliable respiratory health information without the need for expert presence.
- **Encourage User Engagement:** Foster continuous interaction by allowing users to ask follow-up questions and explore related health topics.
- **Integrate Seamlessly:** Work alongside the audio classification module to create a comprehensive and user-friendly respiratory health platform.

Challenges and Constraints

- **Medical Accuracy:** Ensure the assistant provides reliable and clinically sound responses to avoid misinformation.
- **Responsiveness:** Maintain fast and natural conversational interactions to improve user experience.
- **Scope Limitation:** Focus the assistant strictly on respiratory health information, not on clinical diagnosis or treatment.

5.2. Data Understanding

To build an effective Retrieval-Augmented Generation (RAG) system, a high-quality and diverse knowledge base is essential. This phase focuses on understanding the types and

characteristics of the data used to populate the assistant's knowledge base.

Data Collection Strategy

To ensure the reliability and medical accuracy of the information, we performed targeted data scraping from trusted, verified medical sources:

- **World Health Organization (WHO):** For global health guidelines and comprehensive information on respiratory conditions.
- **National Institutes of Health (NIH):** For research-based medical literature and symptom explanations.
- **Centers for Disease Control and Prevention (CDC):** For public health data, prevention guidelines, and symptom overviews.

In addition to these sources, we conducted extensive research to gather:

- **Frequently Asked Questions (FAQs):** Common patient questions about respiratory diseases like asthma, bronchitis, pneumonia, COPD, etc.
- **Patient Education Content:** Short and accessible explanations to ensure the assistant can respond to both technical and non-technical queries.

Data Types

- **Unstructured Text:** Medical articles, clinical summaries, FAQs.
- **Semi-Structured Text:** Bullet points, Q&A-style content from FAQs and health portals.

Data Coverage and Goals

- Ensure comprehensive coverage of the most common respiratory diseases, including symptoms, causes, treatment options, and prevention.
- Include both clinical terms (e.g., "dyspnea", "rales") and layperson vocabulary (e.g., "shortness of breath", "crackles when breathing") to accommodate diverse users.
- Capture a wide range of user intents: symptom explanation, general disease information, treatment advice (non-prescriptive), etc.

5.3. Data Preparation

Once the raw data was collected from trusted sources, it had to be cleaned, standardized, and transformed into a format suitable for both retrieval and generation. This phase shown in the Figure 5.3.1 plays a crucial role in ensuring the quality and efficiency of the RAG pipeline.

5.3.1. Cleaning and Filtering

The raw textual data extracted from sources like WHO and CDC websites often contained a mix of useful content and noise. A cleaning pipeline was applied to ensure only relevant and high-quality information was retained.

- **Structural Cleaning:** HTML content was parsed and converted into plain text while preserving logical sections (e.g., paragraphs, headers). This ensured the data remained well-structured for further processing.
- **Removal of Irrelevant Elements:** Common non-informative parts such as headers, footers, ads, navigation menus, and update timestamps were removed using pattern-matching and HTML parsing.
- **Redundancy Elimination:** Duplicate or repetitive information—like repeated titles or boilerplate disclaimers—was identified and removed. When similar content appeared multiple times, the most complete or contextually rich version was kept.

This process significantly improved the clarity and usefulness of the final dataset used in the RAG pipeline.

5.3.2. Chunking

Once the data was cleaned and filtered, the next crucial step involved splitting the text into coherent and manageable segments, a process known as chunking. This step is essential for the retrieval phase of the RAG (Retrieval-Augmented Generation) system because it enables the model to work with smaller, more focused passages instead of entire articles.

We applied a fixed-length token chunking strategy, where the textual content was segmented into chunks of 800 tokens each. To maintain contextual continuity across chunks and ensure that important information was not cut off at the edges, we used a sliding

window approach with an overlap of 80 tokens between consecutive chunks. This overlapping approach allowed us to preserve the flow of ideas, especially when key information spanned across chunk boundaries.

Choosing 800 tokens as the chunk size provided a balance between depth of context and computational efficiency. Each chunk was large enough to capture meaningful information, such as a complete explanation of a symptom or a treatment recommendation, while still being short enough to work efficiently within the input limits of the language model.

These context-rich chunks formed the basic units for the embedding and retrieval process that powers the RAG system.

5.3.3. Metadata Augmentation

In addition to chunking the textual content, each chunk was enriched with metadata to enhance retrieval accuracy and provide additional context during query processing. The metadata included the source website (such as WHO, NIH, or CDC) and the document title from which the chunk was extracted.

This metadata helps the RAG system to better understand the origin and relevance of each chunk, allowing it to prioritize responses from trusted and authoritative sources. Moreover, having this contextual information supports clearer, more informative answers and makes it easier to trace back responses to their original documents for verification or updates.

5.3.4. Embedding Generation

Following chunking, each text chunk was transformed into a dense vector representation using the `Bge-m3` embedding model. This model encodes the semantic meaning of text, enabling similarity comparisons between user queries and document chunks in a shared vector space.

Generating embeddings is a crucial step in preparing the data for retrieval, as it allows the system to identify and rank the most relevant information based on meaning rather than simple keyword overlap.

5.3.5. Indexing in Vector Store

After generating embeddings for all text chunks, they were indexed using ChromaDB, an efficient and open-source vector database. ChromaDB enables fast and accurate approx-

imate nearest neighbor (ANN) searches based on vector similarity.

This indexing step plays a crucial role in the retrieval process, allowing the system to quickly identify and return the most relevant chunks in response to a user query. By organizing the vectorized data into an optimized structure, ChromaDB ensures low-latency performance during inference.

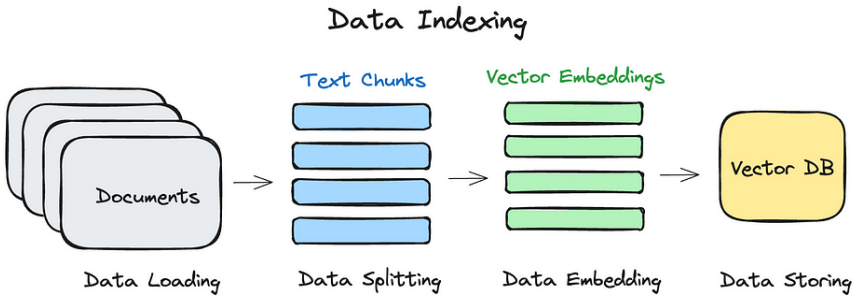

Figure 5.3.1: Indexing process

5.4. Modeling

5.4.1. RAG Architecture Overview

Retrieval-Augmented Generation (RAG) is a hybrid architecture that combines dense document retrieval with generative language models to produce grounded and context-aware answers, as illustrated in Figure 5.4.1, which shows the complete pipeline from user query to final response.

Core Components

- **Retriever:** Retrieves relevant documents from a knowledge base using semantic similarity.
- **Generator:** Generates the final answer using the user's query and the retrieved context.

Workflow

1. **Query Encoding:** The user's question is first embedded into a dense vector using the `Bge-m3` [2] embedding model, which captures the semantic meaning of the input.

2. **Document Retrieval:** The query vector is used to search the `ChromaDB` vector store, where document chunks have also been embedded and indexed. The system retrieves the top-k most relevant chunks based on cosine similarity.

3. **Context Injection:** The retrieved chunks are concatenated and formatted as part of the prompt. This contextual information is added before the query to give the language model background knowledge it can reference during generation.

4. **Response Generation:** The prompt, consisting of both the retrieved content and the original query, is passed to the LLM (`Llama3-2B-Instruct`) [4], which generates a coherent and grounded response. The generated answer is influenced by the evidence provided, increasing factual accuracy.

Advantages

- **Improved Factual Accuracy:** Since answers are based on up-to-date, verified documents, the model is less likely to hallucinate or fabricate information.
- **Dynamic Knowledge Updates:** New content can be added to the vector store without needing to retrain the model, allowing continuous updates and scalability.
- **Transparent Reasoning:** Retrieved chunks can be shown alongside responses, allowing users (including patients and doctors) to trace the origin of the information and validate it.
- **Domain Adaptability:** RAG can be easily tailored to specific domains like medical information, making it highly suitable for our use case involving respiratory health.

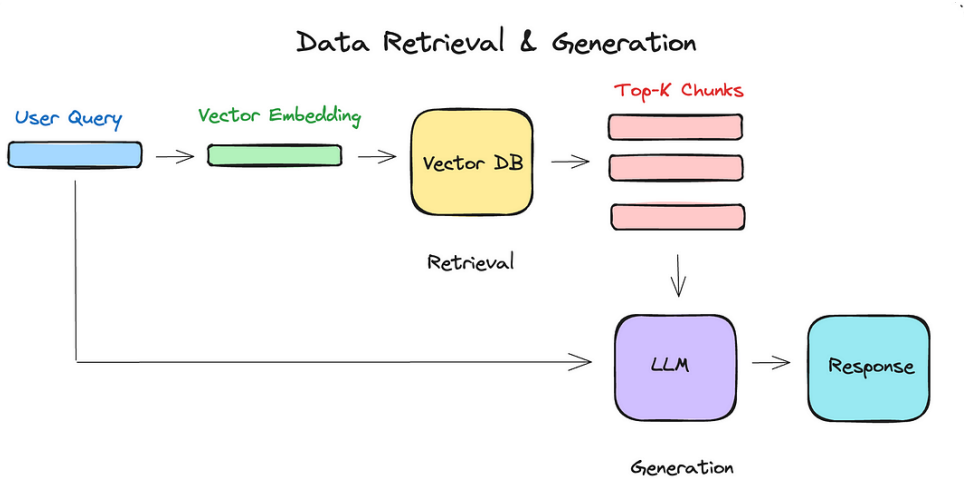

Figure 5.4.1: RAG Pipeline

5.4.2. Retriever Component

The retriever is a critical part of the RAG pipeline responsible for identifying and returning the most semantically relevant document chunks in response to user queries.

We implemented a **dense retriever**, which relies on representing both queries and documents as dense vectors in a shared embedding space. Unlike traditional sparse retrieval methods (e.g., TF-IDF or BM25) that depend on exact keyword matching, dense retrieval captures semantic similarity by measuring the distance between vector representations. This allows retrieval of relevant information even when the exact query terms are not present in the documents.

Dense Retriever Mechanism

Both the query q and each document chunk d_i are embedded into dense vectors:

$$\mathbf{v}_q = \text{Embed}(q), \quad \mathbf{v}_{d_i} = \text{Embed}(d_i)$$

where $\text{Embed}(\cdot)$ is the embedding function implemented by the `BGE-M3` model.

The similarity score between the query and each document chunk is computed using cosine similarity:

$$\text{sim}(\mathbf{v}_q, \mathbf{v}_{d_i}) = \frac{\mathbf{v}_q \cdot \mathbf{v}_{d_i}}{\|\mathbf{v}_q\| \|\mathbf{v}_{d_i}\|}$$

The retriever then ranks all candidate documents based on their similarity scores and selects the top-k documents with the highest scores.

Embedding Model: BGE-M3

To generate these embeddings, we used the `BGE-M3` embedding model developed by BAAI. This model was selected due to its strong capabilities:

- **Multi-Functionality:** Supports dense retrieval, sparse retrieval, and multi-vector retrieval, making it versatile for different search scenarios.
- **Multi-Linguality:** Trained on over 100 languages, enabling support for diverse user inputs.

- **Multi-Granularity:** Handles inputs ranging from short queries to long documents (up to 8192 tokens), which suits embedding medical literature and user queries effectively.

These characteristics allow `BGE-M3` to generate rich, meaningful embeddings that improve retrieval quality.

Top-K Selection

The retriever selects the top 3 most relevant document chunks for each query based on cosine similarity between embeddings. This choice was made after analyzing the collected queries and frequently asked questions (FAQ). We found that no more than three documents were generally necessary to provide sufficient context for answering user queries accurately. Setting top-k = 3 balances providing enough relevant information while avoiding excessive or noisy context that could confuse the language model. This approach ensures the generator receives focused and contextually relevant information, improving the overall response quality.

5.4.3. Generator Component

Language Model Description

We used in the generator component the Llama 3.2 1B Instruct model with Q6-K quantization. This model contains approximately 1 billion parameters, offering a balance between computational efficiency and language understanding capabilities. It is specifically fine-tuned on instruction-following datasets, which enables it to interpret and respond effectively to user queries with clear, context-aware, and relevant answers. The model benefits from Q6-K quantization, a technique that reduces the precision of model weights to 6 bits, allowing for significant reductions in memory usage and faster inference times without substantially sacrificing performance. This makes the model well-suited for deployment in resource-constrained environments, such as mobile applications. The Llama 3.2 1b Instruct model's combination of compact size, instruction fine-tuning, and efficient quantization makes it ideal for our AI assistant, enabling fast and reliable generation of medically accurate responses in real-time.

Instruct-Tuning

Objective: The fine-tuning process aimed to adapt the Llama 3.2 model for instruction-following tasks, enhancing its ability to generate accurate and context-aware responses

based on user queries.

Fine-Tuning Methodology: Meta employed a combination of Supervised Fine-Tuning (SFT) and Reinforcement Learning with Human Feedback (RLHF) to align the model's outputs with human preferences for helpfulness and safety.

- **Supervised Fine-Tuning (SFT):** In this phase, the model was trained on a dataset comprising instruction-response pairs. This supervised approach helped the model learn to follow instructions and generate appropriate responses.
- **Reinforcement Learning with Human Feedback (RLHF):** Following SFT, RLHF was utilized to further refine the model's behavior. Human feedback was incorporated to guide the model towards producing outputs that align with desired characteristics, such as factual accuracy and helpfulness.

Training Data: The fine-tuning process utilized a diverse mix of publicly available online data, encompassing various languages and domains. This multilingual dataset enabled the model to perform effectively across different languages and tasks .

Outcome: The fine-tuned Llama 3.2 Instruct models demonstrated improved performance on common industry benchmarks, outperforming many existing open-source and closed chat models in tasks requiring instruction-following capabilities .

Quantization

Quantization is a widely used technique in deep learning to reduce the computational and memory costs of neural networks by representing weights and activations with lower precision numerical formats. Instead of using 32-bit floating-point numbers (FP32), quantization typically converts these values to 16-bit (FP16), 8-bit, or even lower-bit integer representations such as 6-bit or 4-bit integers. In our project, we employed `Q6_K` quantization, which specifically compresses model weights to 6-bit precision, allowing a significant reduction in model size and inference latency while maintaining strong performance.

Techniques in Quantization: There are several methods for quantization, including:

- **Post-Training Quantization (PTQ):** This technique applies quantization after the model has been fully trained. It is fast and does not require retraining but can sometimes lead to accuracy degradation if not carefully tuned.

- **Quantization-Aware Training (QAT):** In this method, the model is trained or fine-tuned with quantization in mind. The training simulates quantized weights and activations, helping the model adapt to the lower precision and thus preserving accuracy better.

Our use of `Q6_K` falls into post-training quantization optimized for language models, focusing on balancing compression and accuracy without retraining the entire model.

How Quantization Works: The core idea is to map floating-point weights $w \in R$ to discrete quantized values $q \in Z$ using a scale s and zero-point z, typically by:

$$q = \text{round}\left(\frac{w}{s}\right) + z$$

where s scales the real values into the quantization range and z shifts the zero point. In 6-bit quantization, the values are mapped to integers in the range $[0, 63]$.

For inference, the quantized weights are dequantized back to floating-point by:

$$\hat{w} = s \times (q - z)$$

The challenge is to choose s and z such that the quantized weights closely approximate the original weights and minimize error propagation through the network.

Benefits of Q6_K Quantization:

- **Memory Reduction:** Reducing the weight precision from 32-bit to 6-bit reduces model size by over 5 times.
- **Inference Speedup:** Integer arithmetic operations on 6-bit numbers are faster and more energy-efficient on compatible hardware.
- **Compatibility:** `Q6_K` quantization is designed specifically for transformer architectures, preserving their performance better than naïve quantization approaches.
- **Minimal Accuracy Loss:** Carefully calibrated quantization parameters help maintain the semantic understanding and generation quality of the language model.

Challenges

- **Quantization Error:** Low-bit precision introduces quantization noise which can degrade model output if not carefully managed.
- **Hardware Support:** Efficient execution of 6-bit operations requires specialized hardware or optimized libraries, which might not be universally available.
- **Calibration:** Choosing appropriate scale and zero-point parameters requires calibration on representative data to minimize performance loss.

Overall, `Q6_K` quantization provides an effective trade-off between resource efficiency and model performance, making it highly suitable for deploying `Llama 3.2 1B Instruct` in our AI assistant, ensuring fast, reliable, and resource-conscious inference.

Prompt Engineering

Prompt engineering is the process of designing and structuring the input provided to a language model to effectively guide its behavior and output. It plays a critical role in ensuring the model understands the task objective, responds appropriately, and uses the provided context efficiently.

In our RAG-based assistant, prompt engineering was essential to help the language model generate relevant, fact-based, and context-aware answers.

System Prompt Design We defined a system prompt to instruct the model on its role and expected tone. This was used to prime the model at the beginning of each interaction.

```
You are a knowledgeable and compassionate medical assistant chatbot
specialized in respiratory diseases. Your goal is to help users understand
symptoms, conditions, prevention, and treatments related to the respiratory
system. Always provide clear, concise, and medically accurate information
based on trusted sources. Avoid repetition, stay focused on the specific
question, and include practical advice when relevant.
```

This guided the model to behave as a reliable assistant, aligned with the medical nature of the application.

Contextual Prompt Construction When a user submits a query, the top-3 most relevant chunks from the knowledge base are retrieved and injected into the prompt following this structure:

```
Question: [User's original question]
Context:
1. [First retrieved knowledge chunk]
2. [Second retrieved knowledge chunk]
3. [Third retrieved knowledge chunk]

Instruction: Use only the provided context to answer.
If information is incomplete, respond: "Based on my medical resources,
I would suggest consulting a healthcare professional about this."
Never speculate beyond the given context.
```

The prompt explicitly instructs the LLM to: (1) treat the context as primary source but verify completeness, (2) acknowledge limitations when context is insufficient by recommending professional consultation, and (3) strictly avoid fabrication. This design ensures trustworthy responses while minimizing hallucinations and maintaining clinical responsibility - particularly crucial for respiratory health queries where accuracy impacts patient outcomes.

This design ensures that responses are fact-grounded, transparent, and aligned with medical best practices.

5.5. Evaluation

5.5.1. Retriever Evaluation

The retriever component plays a crucial role in the Retrieval-Augmented Generation (RAG) pipeline. It is responsible for selecting the most relevant pieces of information (chunks) from a large corpus in response to a user query. To evaluate its effectiveness, we used several well-established metrics from the information retrieval field.

Evaluation Metrics

- **Mean Reciprocal Rank (MRR):** The MRR evaluates how highly the first relevant document appears in the ranked list of retrieved documents. For each query, the reciprocal rank is calculated as:

$$\text{Reciprocal Rank} = \frac{1}{\text{rank of the first relevant document}}$$

  Then, the MRR is the average of these reciprocal ranks across all queries:

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$$

A higher MRR (closer to 1) indicates that the system often retrieves the correct document in the top position, which is critical for user-facing applications.

- **Mean Average Precision (MAP):** MAP considers all relevant documents, not just the first one. For each query, it calculates the average precision across the retrieved results and then averages over all queries. It is defined as:

$$\text{MAP} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \left(\frac{1}{m_i} \sum_{k=1}^{n} P(k) \cdot rel(k) \right)$$

Where:

  - $P(k)$ is the precision at cut-off k
  - $rel(k)$ is a binary indicator of relevance at position k
  - m_i is the total number of relevant documents for query i

This metric rewards systems that rank all relevant documents highly.

- **Recall@K (e.g., Recall@3):** Recall@K measures how many relevant documents are retrieved among the top K results. It is computed as:

$$\text{Recall@K} = \frac{\text{\# of relevant documents in top K}}{\text{Total \# of relevant documents}}$$

In this project, we used Recall@3 since the retriever was configured to return Top-K = 3 documents. This value was chosen based on manual analysis of a dataset of FAQs and real user queries, where it was observed that most answers could be accurately supported using at most three context documents.

Global MRR Benchmark Evaluation

The dense retriever leveraging the `BGE-M3` embedding model was evaluated on a multilingual MRR benchmark dataset. The results demonstrate that `BGE-M3` consistently outperformed other embedding models across multiple languages, showcasing its strong generalization and retrieval capabilities.

The MRR scores achieved by `BGE-M3` on different languages are as follows:

- English: 0.68

- French: 0.60
- Hungarian (HU): 0.64
- Czech (CS): 0.65

These scores highlight the model's robust ability to retrieve relevant documents effectively in various linguistic contexts, which is crucial for building a reliable AI assistant. The comparative performance is illustrated in Figure 5.5.1, which clearly shows `BGE-M3` leading other tested models on this benchmark.

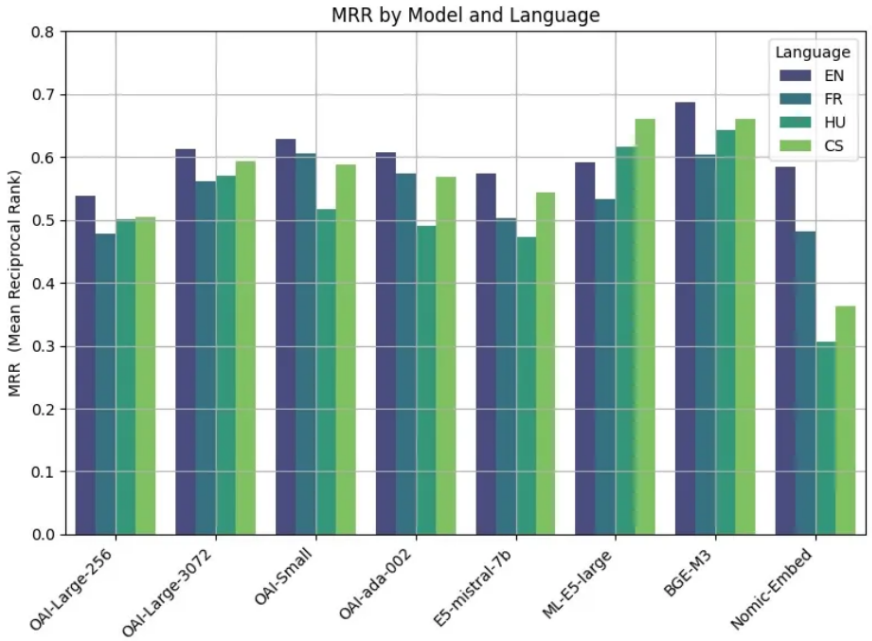

Figure 5.5.1: MRR Benchmark results

Domain Evaluation

For evaluating the retriever's performance in the context of respiratory diseases, we created a dedicated evaluation dataset. This dataset includes approximately 50 manually selected questions related to common respiratory conditions, symptoms, and treatments. The questions were gathered from verified sources such as medical forums, FAQs, and healthcare websites to reflect real-world user inquiries.
We conducted a detailed manual analysis to determine which documents are relevant for answering each question. This involved reviewing the retrieved documents and annotating those essential for fully understanding the queries. This process also helped us identify the optimal number of relevant documents per question, informing retrieval parameter choices such as the top-k value.

This manual curation provides a high-quality ground truth for evaluation, allowing precise assessment of retriever performance in our specific domain.

The retriever achieved **exceptional performance** on our respiratory disease dataset:

- **Mean Reciprocal Rank (MRR): 0.72**
- **Mean Average Precision (MAP): 0.68**
- **Recall@3: 0.75**

These scores demonstrate the retriever's effectiveness in ranking relevant documents higher, which is critical for accurate and context-aware AI assistant responses. The strong performance reflects the suitability of our embedding model and retrieval strategy tuned to the domain.
Nonetheless, some questions involving rare or highly specialized information showed slightly lower precision, highlighting areas for potential improvement. Overall, the results confirm that our retriever performs well in the respiratory disease domain and supports the AI assistant's effectiveness.

5.5.2. End-to-End System Evaluation

To assess the end-to-end effectiveness of our Retrieval-Augmented Generation (RAG) system, we applied the *LLM-as-a-Judge* technique, where a powerful large language model evaluates generated answers using a structured rubric. Specifically, we employed `GPT-4` as an automatic evaluator to review responses generated by our system for a carefully curated set of 50 domain-specific questions.
As depicted in Figure 5.5.2, the evaluation pipeline feeds the user question, the RAG-generated response, and—where applicable—the baseline response into the judge model. GPT-4.0 then rates the answers on six essential dimensions, each on a scale from 0 to 10:

- **Groundedness:** Alignment with retrieved context
- **Correctness:** Factual and domain accuracy
- **Coherence:** Clarity, structure, and logical flow
- **Relevance:** Direct response to query
- **Novelty:** Added value from synthesis
- **Overall Score:** General utility and effectiveness

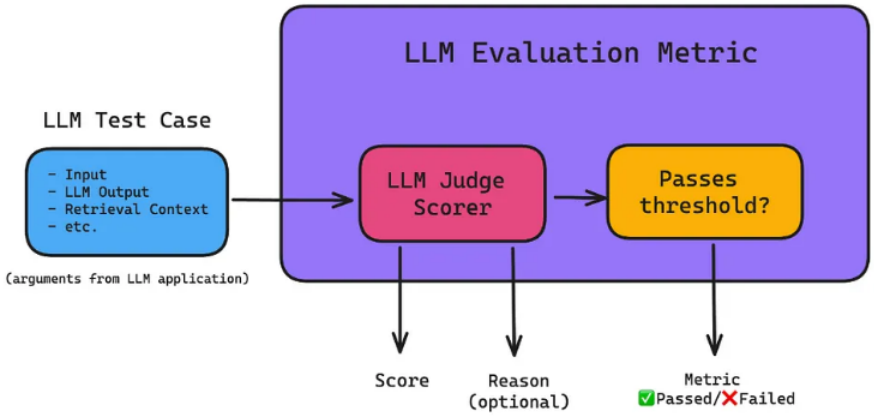

Figure 5.5.2: LLM Evaluation process

Metric	RAG System	Baseline
Groundedness	8.7	5.2
Correctness	8.5	5.8
Coherence	9.1	8.7
Relevance	8.9	6.3
Novelty	7.8	6.9
Overall Score	8.6	6.4

Table 5.1: Performance comparison: RAG vs. Baseline (50 questions)

To quantify the contribution of retrieval, we compared our RAG system with a baseline model that used the same Llama 3.2 1B Instruct LLM but without any contextual documents. The average scores across the 50-question evaluation set are shown in Table 5.1.

As observed in Table 1, the RAG system consistently outperformed the baseline across all metrics. The most substantial gains were seen in groundedness, correctness, and relevance, confirming that access to relevant, retrieved context significantly enhances the LLM's ability to generate reliable and precise answers. While coherence remained relatively high in both cases, the RAG version improved the interpretability and factual grounding of responses, making it better suited for medical assistance applications.

5.6. Deployment

To make the system accessible and efficient in a real-world setting, we designed a lightweight and modular deployment architecture. The deployment stack ensures that both the LLM and embedding-based retrieval components are available through a simple API interface.

Ollama for Model Serving: We used Ollama to serve both the Llama 3.2 1B In-

struct model and the BGE-M3 embedding model locally. Ollama provides an efficient and GPU-friendly environment to run quantized models with minimal overhead, which helped achieve low-latency responses during generation and retrieval.

FastAPI for Backend Interface: A FastAPI server was built to act as a communication layer between the front end and the core RAG system. This backend handles user queries, performs retrieval using the embedding model, generates responses via the LLM, and returns structured answers to the front end.

Docker for Containerization: To ensure portability and ease of deployment, the entire backend system (including FastAPI and Ollama runtime) was packaged into a Docker container. This allows consistent deployment across environments and simplifies maintenance and scaling.

This deployment setup strikes a balance between efficiency and simplicity, making the AI assistant suitable for both development and production testing scenarios.

Conclusion

This chapter outlined the development of our AI assistant powered by a Retrieval-Augmented Generation (RAG) system. From concept to deployment, the focus was on creating a helpful, reliable, and multilingual tool to assist users with respiratory health questions. The results demonstrate the potential of combining retrieval with generation to provide informative and grounded answers, while always emphasizing the importance of professional medical advice.

Chapter 6

Implementation

Introduction

This chapter presents the implementation phase of our project: an AI-powered mobile application designed to analyze breathing sounds and assist in detecting respiratory diseases.

The implementation phase translates theoretical concepts into a working system. It focuses on the tools, technologies, and methodologies used to build the application and ensure its reliability and performance.

The core objective is to develop a mobile app capable of recording respiratory audio, uploading it securely, and using AI-based models to provide preliminary diagnostic feedback. The app also allows users to upload medical reports and offers a simple, intuitive interface with real-time insights.

To achieve this, we adopted a modular architecture that separates mobile development, backend services, and AI inference. The mobile app was developed using Flutter, chosen for its cross-platform capabilities and development speed. Appwrite was selected as the backend platform for its support of authentication, storage, and serverless functions.

This chapter begins with an overview of the system architecture and its components. It then details the implementation of key modules, including audio capture and encryption, data transmission, backend integration, and AI model deployment. Each section highlights the technical decisions made and the reasoning behind them.

6.1. System Architecture

The system architecture of our AI-powered mobile application is designed to ensure modularity, scalability, and secure data handling. It is composed of three main components: the mobile application, the backend services, and the AI inference engine. Each component plays a distinct role while communicating seamlessly with the others.

The architecture follows a modular pattern to facilitate independent development and testing of each module. This design choice improves maintainability and simplifies the integration of additional features in the future.

The overall system structure is illustrated in Figure 6.1.1, which outlines the interactions between the core components of the application.

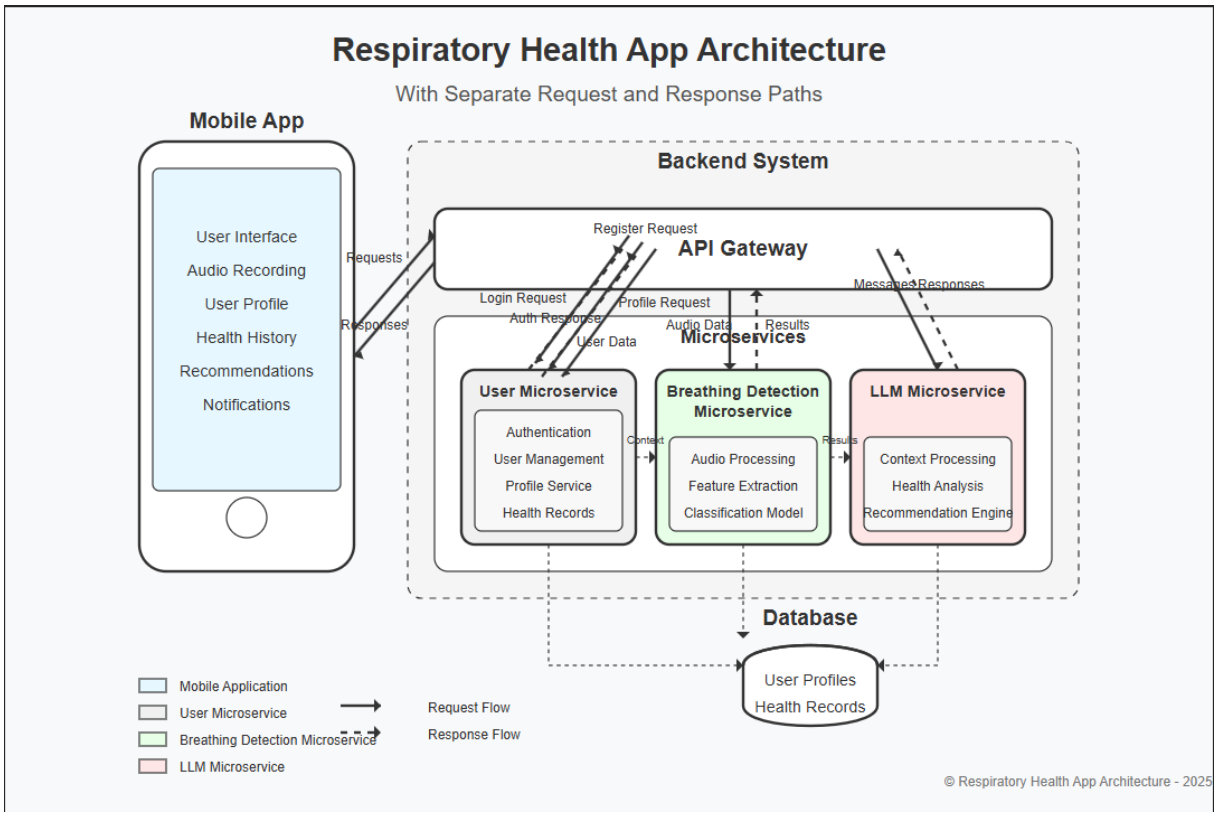

Figure 6.1.1: System Architecture of the AI-Powered Mobile Application

Mobile Application

The mobile application, developed using Flutter, provides the user interface and handles client-side functionalities. These include audio recording, local encryption, file selection, and UI interactions. It also manages authentication and communication with the backend.

Backend Services

The backend is powered by Appwrite, a self-hosted backend-as-a-service platform. It handles authentication, file storage (medical reports and audio), and serverless functions. Appwrite ensures secure and scalable data management while offering APIs that are easy to integrate with Flutter.

AI Inference Engine

The AI inference module is responsible for analyzing respiratory audio data and generating preliminary diagnostic feedback. Once an audio file is uploaded and validated, it is sent to a dedicated inference server or function that runs a pre-trained deep learning model to detect anomalies in the breathing pattern.

This modular architecture enables efficient development, enhances security, and supports the core functionality of respiratory disease detection.

6.2. Mobile Application

The mobile application serves as the central user interface of our AI-powered respiratory analysis system. Built with Flutter, it offers a seamless and cross-platform experience. This section presents a walkthrough of the app's core features, showcasing its design, functionality, and user flow.

6.2.1. Visual Identity and Design Language

The application embraces a medically inspired color palette referencing the official World Health Organization (WHO) guidelines, aiming for clarity, trust, and accessibility. The logo symbolizes the lungs through a stylized graphic, while the central dot represents a sensor tasked with detecting respiratory anomalies.

The primary branding elements, including both filled and outline logo variants, are shown in Figure 6.2.1.

Figure 6.2.1: Application Logos: Filled and Outline

6.2.2. Authentication: Login and Signup

The user journey begins with secure authentication. A simple interface for logging in or signing up is provided, as seen in Figure 6.2.2.

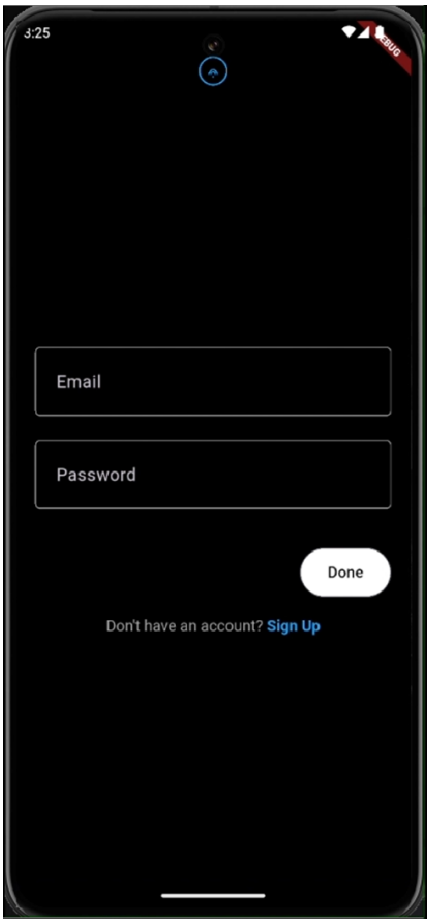

Figure 6.2.2: Login Screen

6.2.3. User Profile Management

Once authenticated, users are directed to a profile screen where they can view and manage their personal information. This interface is illustrated in Figure 6.2.3.

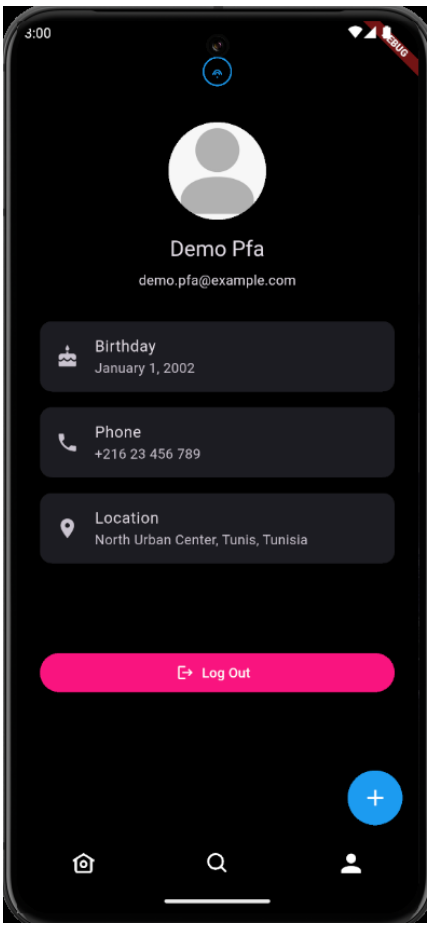

Figure 6.2.3: Profile Screen

6.2.4. Home Screen – No Diagnoses Yet

The initial home screen is presented when the user has not yet submitted any diagnoses. This clean interface is displayed in Figure 6.2.4.

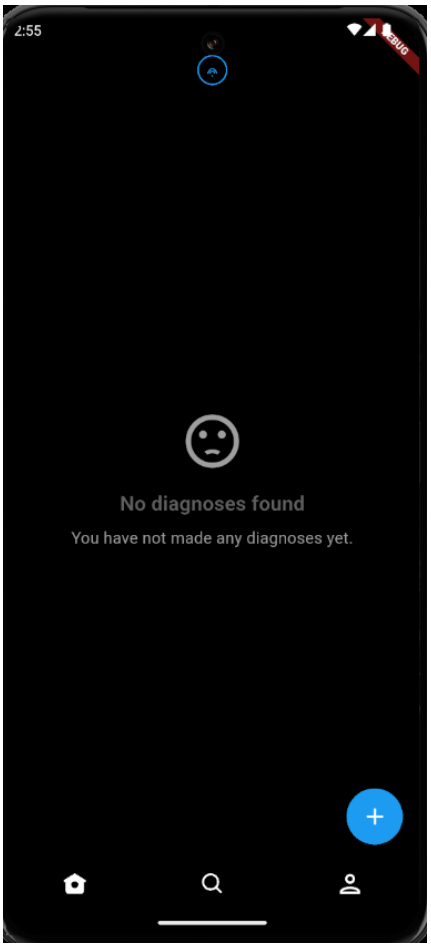

Figure 6.2.4: Home Screen Without Diagnoses

6.2.5. Search Screen – No History

A search feature is available for reviewing past diagnoses. In the absence of history, the interface appears as shown in Figure 6.2.5.

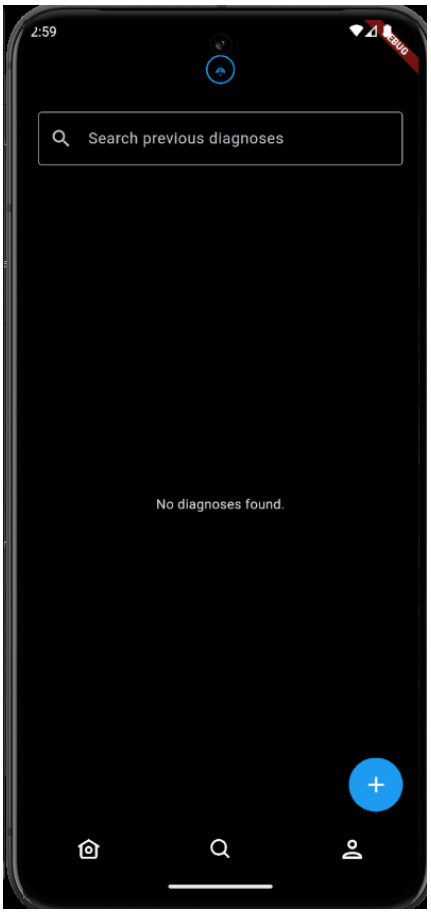

Figure 6.2.5: Search Screen Without History

6.2.6. Creating a New Diagnosis

To start a diagnosis, users can record audio, upload medical files, and input symptoms using the interface shown in Figure 6.2.6.

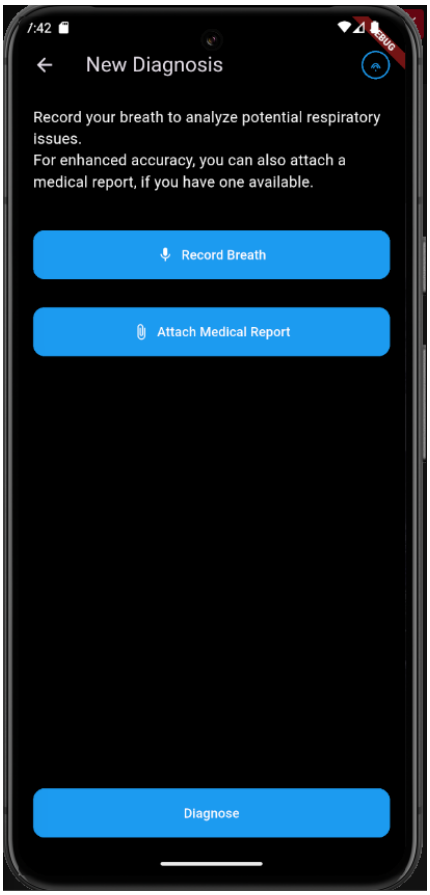

Figure 6.2.6: Diagnosis Creation Interface

6.2.7. Home Screen – With Previous Diagnoses

After a diagnosis is performed, the home screen updates to reflect the user's diagnostic history, as demonstrated in Figure 6.2.7.

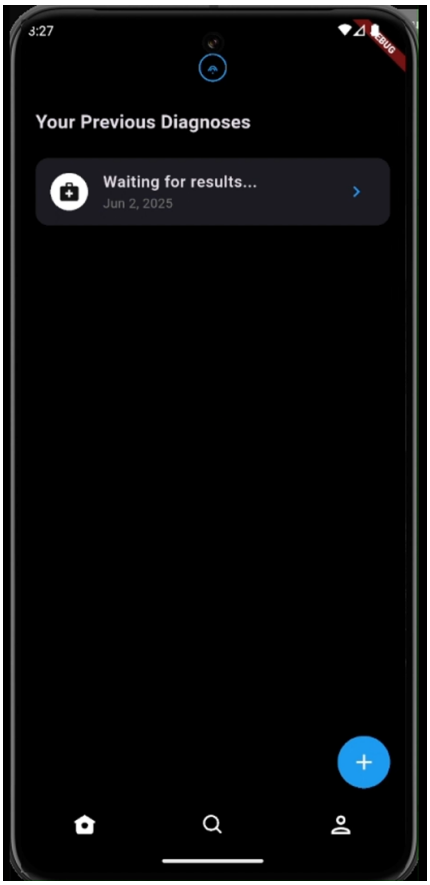

Figure 6.2.7: Home Screen Displaying Previous Diagnoses

6.2.8. Diagnosis Details – Waiting for Results

Once a new diagnosis is submitted, the system informs the user that the analysis is in progress. This intermediate state is depicted in Figure 6.2.8.

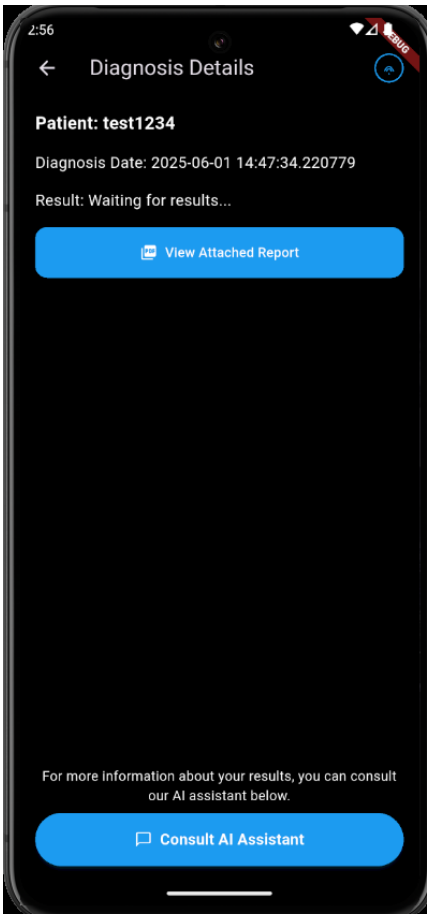

Figure 6.2.8: Diagnosis Details – Awaiting AI Analysis

6.2.9. Diagnosis Details – Results Ready

After analysis is complete, users are presented with detailed diagnostic feedback, as shown in Figure 6.2.9.

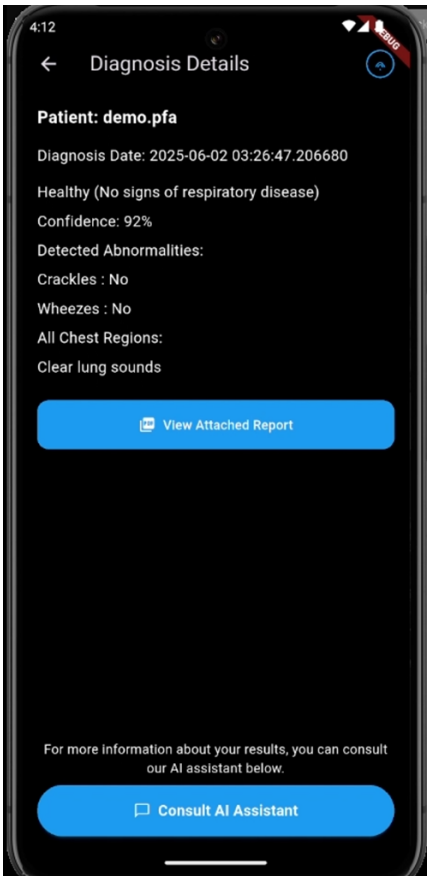

Figure 6.2.9: Diagnosis Details – Results Ready

6.2.10. AI Assistant Onboarding

When consulting the AI assistant for the first time, users are guided through an onboarding screen that outlines its purpose, as illustrated in Figure 6.2.10.

Figure 6.2.10: AI Assistant Onboarding Screen

6.2.11. AI Assistant – Start Screen

Following onboarding, users arrive at a starting interface that invites medical queries, as depicted in Figure 6.2.11.

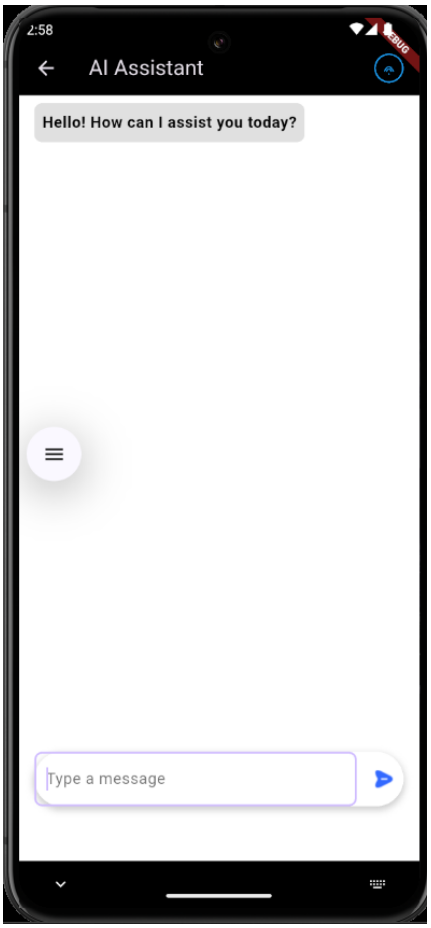

Figure 6.2.11: AI Assistant – Starting Interface

6.2.12. AI Assistant – Chat Interface

The assistant responds in a conversational style, as shown in Figure 6.2.12, providing interactive, medically grounded guidance.

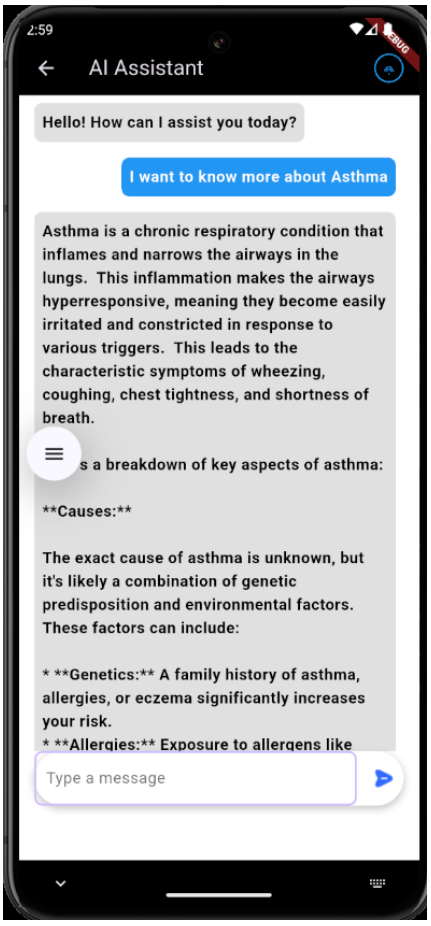

Figure 6.2.12: AI Assistant Chat Interaction

6.2.13. Summary

The mobile application was designed to deliver a smooth, medically reliable, and user-centric experience. Each screen plays a specific role in guiding the user through authentication, diagnosis creation, result review, and AI consultation. Leveraging Flutter, the app ensures cross-platform consistency and accessibility. The clear design, inspired by WHO standards, and the integration of visual cues like the lung-symbol logo reinforce its identity as a trusted respiratory health tool.

6.3. Backend and API Integration

The backend infrastructure is a critical component of our AI-powered mobile application, enabling secure data handling, user management, and seamless communication between the mobile client and AI services. Built around Appwrite—a self-hosted backend-as-a-service (BaaS) platform—the system benefits from a robust, modular, and scalable architecture.

6.3.1. Appwrite as Backend-as-a-Service (BaaS)

Appwrite serves as the core backend platform, providing essential services such as authentication, database storage, file handling, serverless computing, and real-time updates. It supports structured data for user profiles and health records, encrypted storage for audio and medical files, and custom functions for executing AI-based health analysis.

The architecture shown in Figure 6.3.1 depicts how the Flutter-based mobile frontend communicates with Appwrite services over secure HTTPS, enabling a seamless flow of authentication, data storage, and diagnostic processing. Each backend service operates independently while being orchestrated via the Appwrite API Gateway, which manages routing and authentication for all client requests.

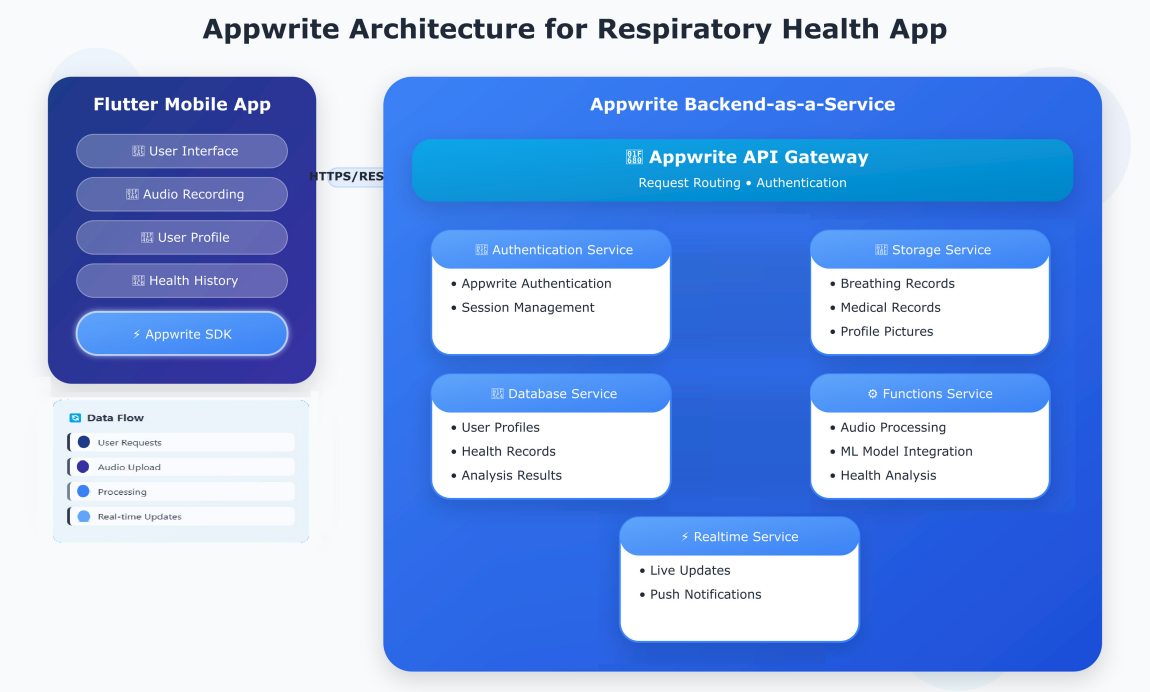

Figure 6.3.1: Appwrite-Based Backend Architecture

6.3.2. Authentication and Session Management

User authentication is handled through Appwrite's secure email-password mechanism. Upon successful login or registration, the client receives a session token, which is used to authorize all subsequent requests. This approach ensures privacy, data isolation, and secure access to personalized health information.

6.3.3. Diagnosis Submission and File Handling

When a user initiates a new diagnosis, the application uploads the audio recording and any supplementary files to Appwrite's storage service. These assets are then linked to a new diagnosis document in the Appwrite database, which contains relevant metadata such as the timestamp, reported symptoms, and user ID.

6.3.4. Triggering AI Inference

Following submission, an Appwrite function is triggered asynchronously. This serverless function retrieves the uploaded audio file, forwards it to the AI inference engine through a secure REST API, and receives a structured JSON response. The results—comprising predicted conditions, confidence scores, and supplementary data—are saved back to the corresponding document in the database.

6.3.5. Result Synchronization with the Mobile App

To ensure responsiveness, the mobile app routinely polls the backend for updates to diagnosis records. When inference results are available, the interface updates to reflect the final diagnosis. Optionally, Appwrite's real-time services can push changes to the client directly, allowing live updates and improved interactivity in future versions.

6.3.6. Summary

The backend architecture effectively bridges the user-facing application with the AI-driven analysis pipeline. Appwrite provides a secure, modular environment for managing identity, health data, and inference workflows. This integration enables scalable diagnostics while maintaining user privacy and system extensibility.

6.4. Tools and Technologies Used

The development of our AI-powered mobile diagnostic system was supported by a carefully selected suite of tools, frameworks, and platforms that together ensured efficiency, scalability, and security across the entire stack. This section introduces the core technologies employed throughout the project.

6.4.1. Flutter for Cross-Platform Development

Flutter was chosen as the core framework for building the mobile application, enabling a unified codebase for both Android and iOS platforms. Its declarative UI toolkit and reactive architecture allowed for efficient development and a smooth user experience.

The Flutter logo is shown in Figure 6.4.1.

Figure 6.4.1: Flutter Logo

6.4.2. Dart Programming Language

Dart, the language behind Flutter, was used for implementing both application logic and UI elements. Its type safety and support for asynchronous operations made it well-suited for responsive mobile development.

The Dart logo is displayed in Figure 6.4.2.

Figure 6.4.2: Dart Logo

6.4.3. Appwrite for Backend-as-a-Service

Appwrite was selected as the backend-as-a-service solution, offering user authentication, database storage, file handling, and serverless functions. Its modular design simplified backend operations and enhanced security.

The Appwrite logo is presented in Figure 6.4.3.

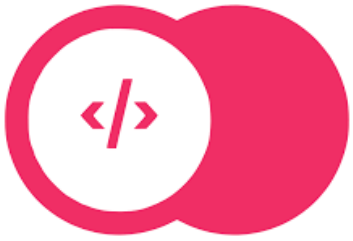

Figure 6.4.3: Appwrite Logo

6.4.4. Python for AI Model Deployment

Python was used to develop and deploy the AI inference engine, leveraging libraries such as PyTorch and Hugging Face Transformers for audio-based diagnosis prediction.

The Python logo can be seen in Figure 6.4.4.

Figure 6.4.4: Python Logo

6.4.5. REST APIs for Communication

RESTful APIs facilitated communication between the mobile frontend, Appwrite backend, and AI inference engine. This decoupled architecture allowed independent scaling and streamlined integration.

A representation of REST APIs is shown in Figure 6.4.5.

Figure 6.4.5: REST API Representation

6.4.6. Docker for Containerization

Docker was employed to containerize both Appwrite services and the AI backend. This ensured consistent deployment, streamlined development, and enhanced scalability.

The Docker logo is illustrated in Figure 6.4.6.

Figure 6.4.6: Docker Logo

6.4.7. VS Code and Postman for Development and Testing

Visual Studio Code served as the primary development environment, providing extensive support for Dart, Flutter, and Python development.

The VS Code logo is shown in Figure 6.4.7.

Figure 6.4.7: VS Code Logo

Postman was used for testing API endpoints and simulating request flows during diagnosis.

The Postman logo is displayed in Figure 6.4.8.

Figure 6.4.8: Postman Logo

6.4.8. Android Studio for Android Emulation

Android Studio was utilized to run the Android emulator, enabling thorough testing of the mobile application on virtual Android devices and ensuring compatibility and performance.

The Android Studio logo is shown in Figure 6.4.9.

Figure 6.4.9: Android Studio Logo

6.4.9. Git and GitHub for Version Control

Version control was managed using Git, with GitHub as the remote repository. This setup supported collaborative development, issue tracking, and code versioning.

The Git and GitHub logos are presented in Figure 6.4.10.

Figure 6.4.10: Git and GitHub Logos

6.4.10. Summary

The integration of these technologies ensured a robust, maintainable, and scalable solution. From frontend development to AI inference and deployment, each tool played a critical role in delivering a reliable diagnostic experience to end users.

Conclusion

This chapter provided a comprehensive overview of the system architecture and implementation details of the AI-powered mobile diagnosis application. The system is structured into three main layers: the mobile frontend, the backend infrastructure, and the AI inference engine. Each component was designed to support secure, efficient, and scalable workflows for users seeking preliminary medical feedback.

The mobile application, developed using Flutter, offers a cross-platform interface for users to describe symptoms and receive AI-generated diagnostic results. Its intuitive design and responsive UI enhance user engagement and streamline the diagnosis submission process.

The backend, powered by Appwrite, handles authentication, data storage, and serverless function execution. It acts as a secure intermediary between the mobile client and the AI inference engine, ensuring encrypted data handling, user-specific data isolation, and real-time result synchronization.

The AI pipeline, built using Python and transformer-based models, interprets audio symptom descriptions and returns structured diagnostic suggestions. It integrates seamlessly with the backend via REST APIs, enabling efficient model inference workflows.

Together, these components form a robust, modular architecture that supports accurate, secure, and real-time delivery of diagnostic information. The technology stack and integration strategy ensure future scalability, maintainability, and adaptability to more advanced diagnostic features.

Conclusion and Perspectives

Conclusion

This project presented the design and implementation of an AI-powered mobile application for breathing analysis and the detection of respiratory anomalies. The system combines mobile development with advanced AI techniques to deliver a modern, accessible diagnostic tool that supports early screening for conditions such as asthma, pneumonia, COPD, and sleep apnea.

The application allows users to record their breathing sounds, upload medical reports, and interact with an AI assistant. Through the use of Flutter for frontend development and Appwrite as a secure backend platform, the system ensures a smooth, cross-platform experience while maintaining user privacy and data integrity. On the AI side, the pipeline includes respiratory sound classification using deep learning models and personalized feedback powered by large language models (LLMs), enabling both automated diagnosis and natural language interaction.

The modular system architecture, built around RESTful APIs and containerized services, enabled seamless integration between the frontend, backend, and AI components. Tools like Docker, Git, Postman, and VS Code contributed to a robust and maintainable development workflow.

Perspectives

While the current version of the application demonstrates the feasibility and effectiveness of AI-assisted respiratory analysis, several avenues remain for future improvement:

- **Improved Audio Processing:** Incorporating advanced audio enhancement techniques, such as adaptive noise cancellation and real-time denoising, would increase the robustness of predictions in noisy environments.

- **Model Optimization and On-Device Inference:** Future iterations could explore model compression and quantization techniques to allow inference directly on the mobile device, improving response time and reducing backend dependency.
- **Dataset Expansion:** Increasing the diversity and size of the training dataset — including data from various age groups, devices, and environments — would improve the model's generalization to real-world scenarios.
- **User Feedback Integration:** Allowing users to provide structured feedback on the accuracy of diagnoses and AI assistant responses could help continuously refine the system.
- **Medical Integration:** Long-term perspectives include integration with clinical workflows or partnerships with healthcare providers to support telehealth and digital triage systems.
- **Security and Regulatory Compliance:** Future versions should consider compliance with medical data regulations (e.g., GDPR, HIPAA) and formal security audits to ensure safe deployment in healthcare settings.

In conclusion, this project highlights the potential of combining mobile technologies with AI to support respiratory healthcare. With continued development and validation, this system can evolve into a trusted tool for early screening, patient monitoring, and AI-powered clinical assistance.

Bibliography

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention Is All You Need*. Advances in Neural Information Processing Systems, 30 (NeurIPS 2017).

[2] Xu, Canwen, Xiao, Yixin, Ma, Yutai, Ji, Heng, & Wang, Yining. (2024). *BGE-M3: A Versatile Embedding Model for Retrieval, Classification, and Re-ranking*. arXiv preprint arXiv:2403.03692.

[3] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Riedel, S. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv preprint arXiv:2005.11401.

[4] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Ram, O., Bambroo, V., Lample, G., Scialom, T. (2024). *LLaMA 3: Open Foundation and Instruction-Tuned Language Models*. arXiv preprint arXiv:2404.14219.

[5] Perna, D., Tagarelli, A. (2019). *Deep auscultation: Predicting respiratory anomalies and diseases via recurrent neural networks*. Proceedings of the 2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2821–2827.

[6] Demir, F., Sengur, A. (2021). *A deep CNN-LSTM model for classification of lung sounds*. Biomedical Signal Processing and Control, 70, 102896.

[7] Ibrahim, A., Hussain, A. (2020). *Lung sound classification using recurrent neural networks and short-time Fourier transform*. Computers in Biology and Medicine, 122, 103771.

[8] Aygun, M., Sert, E. (2022). *Respiratory sound classification using EfficientNet and spectrogram augmentation*. IEEE Access, 10, 45329–45340.

[9] Rocha, B. M., Filos, D., Mendes, L., Vogiatzis, I., Perantoni, E., Kaimakamis, E., ... Nascimento, J. C. (2017). *An open access database for the evaluation of respiratory sound classification algorithms*. Physiological Measurement, 38(9), N1–N13.

[10] Wei-Hung Hsu, Hung-Yi Lee, Po-Han Yang, Arsha Nagrani, Joel Shor, Luciana Ferrer, and Ignacio Lopez Moreno. CLAP: Learning Audio-Text Joint Embeddings from Large-Scale Webly-Supervised Data. *arXiv preprint arXiv:2302.03102*, 2023.