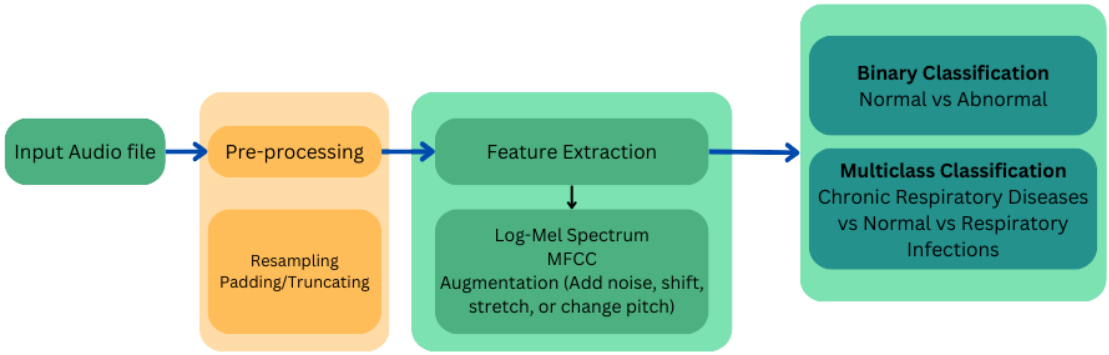
AmplifierHealth Respiratory Sound Classifier (Take-Home Project)

This project involves developing a machine learning model to classify respiratory sounds into diagnostic categories using the ICBHI 2017 Challenge Dataset. The pipeline includes data preprocessing, feature extraction, model training, evaluation, and deployment. You can overall view on the project below:



Repository

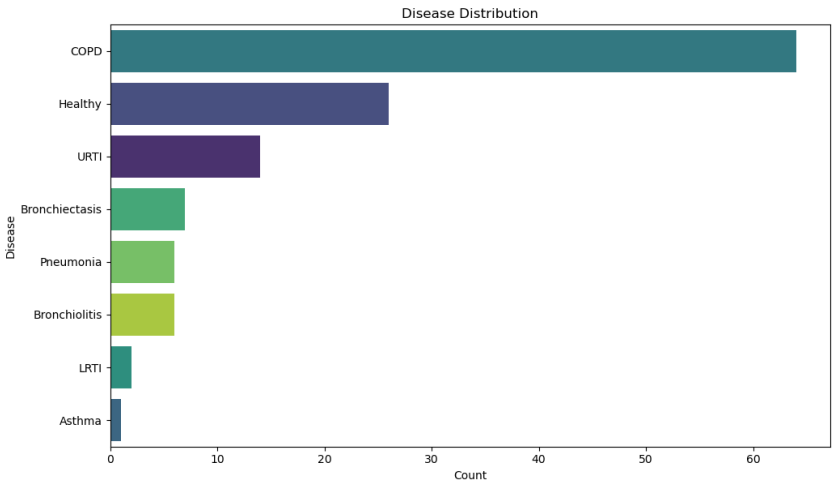
You can access the repositories for this project here: GitHub Repository, HuggingFace Also, you can check out GitHub Actions for a taste of CI/CD 😊

Dataset Overview

- **Total Patients**: 126 (8 classes)
- **Most Common Disease**: COPD (64 patients)
- **Least Common Disease**: Asthma (1 patient)

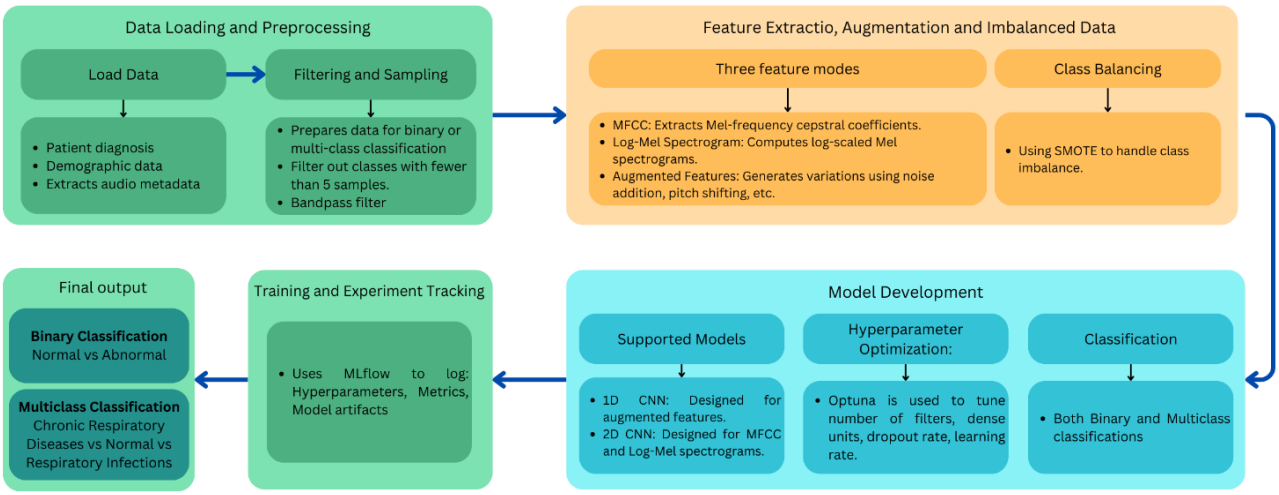
Diagnosis Distribution

As you can see, dataset is highly imbalanced and is skewed toward abnormal patterns (e.g., COPD has more items than healthy cases). Moreover, the descriptive variables (such as Adult BMI and Child Weight) have too many missing values. There might be a benefit to apply imputation techniques and create a dual-stream models that fuse descriptive data with audio files, however, my time was limited to implement such a model (will be talking about design choice in next sections).



Model Training Scheme

In summary, the overview of my framework is depicted in image below. In next sections, I will describe each part in-depth.



Data Preparation

In preprocessing, I did typical down sampling to 16KHz, and band-pass filtering. I also used SMOTE and augmentation to balance the dataset and enhance ratio of class samples. If I had more time, I would spend more time in making this step better than its current version. Also I would add more design choices here to see what are the effects of different preprocessing in combination with different features and model types (reminder: exploring system's design space is kind of what I did in this project)

Types of Models and Input Features

Since there was only a short amount of time, I tried to mimic how I would approach such a classification problem. Therefore, I created a workflow that tests different options, from data processing to feature extraction and model architecture, to determine the best-performing model/configuration. I tried to develop as modular as possible, so you could build up on current codes and integrate models like Audio Spectrogram Transformers (that apparently have higher accuracy based on Challenge's leaderboard).

The current workflow supports six distinct models, combining two classification settings (binary and multi-class) with three feature extraction modes (MFCC, Log-Mel Spectrogram, and Augmented Features):

Feature Mode	Definition	Representation Type	Model Type Required
MFCC	Represents short-term power spectrum using the Mel scale, emphasizing human auditory perception.	2D Matrix	2D CNN
Log-Mel	A spectrogram emphasizing quieter sounds by applying a logarithmic scale to Mel frequencies.	2D Matrix	2D CNN
Augmented	Features derived from MFCCs with added variability through augmentation techniques such as noise addition, pitch shifting, and time stretching.	1D Sequence	1D CNN

Why Both Binary and Multi-Class?

The project aims to create a comprehensive testing framework capable of testing different input types, models, and outputs. This framework allows for easy substitution of components, such as implementing an Audio Spectrogram Transformer for testing.

- **Binary Classification**: Distinguishes Normal from Abnormal cases. Simpler but prone to overfitting.
- **Multi-Class Classification**: Groups abnormal cases into specific categories (e.g., Chronic Respiratory Diseases, Respiratory Infections). Since data was imbalanced, I tried to find a mid-point granularity for classification (not binary, and not 8-class classification).

The training scheme for the project employs binary cross-entropy loss for binary classification tasks and categorical cross-entropy for multi-class problems. To mitigate overfitting, dropout layers are incorporated as a form of regularization. Additionally, early stopping is implemented by monitoring the validation loss. Hyperparameter optimization is carried out using Optuna, a framework that systematically tunes critical parameters to enhance model performance. Parameters optimized include the number of filters in convolutional layers, the number of units in dense layers, dropout rates, and learning rates. The model selection criteria are guided by its performance on unseen test data (in terms of precision, recall, F1-score, confusion matrix, and ROC-AUC). MLflow logs are available "mlruns" directory.

Model Performance Metrics

Clasification setting	Feature type	Accuracy	Precision	Recall	F1 Score	ROC-AUC	Model architecture
Binary	Augmented MFCC	0.0567	1	0.0567	0.1073	None	1D-CNN
	Log Mel	0.0072	1	0.0072	0.0144	None	2D-CNN
	MFCC	0.3116	1	0.3116	0.4751	None	2D-CNN
multiclass	Augmented MFCC	0.4783	0.7935	0.4783	0.4736	0.9777	1D-CNN
	Log Mel	0.8623	0.7436	0.8623	0.7986	0.4988	2D-CNN
	MFCC	0.8696	0.7763	0.8696	0.8157	0.85	2D-CNN

Optuna significantly improved the performance of different model by trying different model configurations. However, you can see binary models lead to absolute non-reliable performance. I think MFCC features has better reliability in both binary and multiclass classification. Noteworthy to mention that in this project my main goal was creating the overall workflow (that explores different models, and has a deployment layer at the end), and I couldn't put more than a day on model training itself. I believe that the performance could enhance by using different features and model architectures.

Model Deployment and metrics logging

As said, a streamlit UI hosts developed models on HuggingFace. This tool allows you to classify respiratory sounds into various categories using pre-trained models. Choose one of the two modes below based on your needs:

1. Quick Multiclass Mode: A fast and straightforward way to classify audio files using a multiclass model with augmented features.
2. Flexible Mode: Customize the classification process by selecting your preferred model type (binary/multi) and feature type (MFCC, Log-Mel, or Augmented).

You can also have access to metrics dashboard to monitor live metrics including request counts, response times, and error rates. These metrics are collected through Prometheus. Github repo has step-by-step tutorial on how to run Prometheus and visualize it via Grafana.