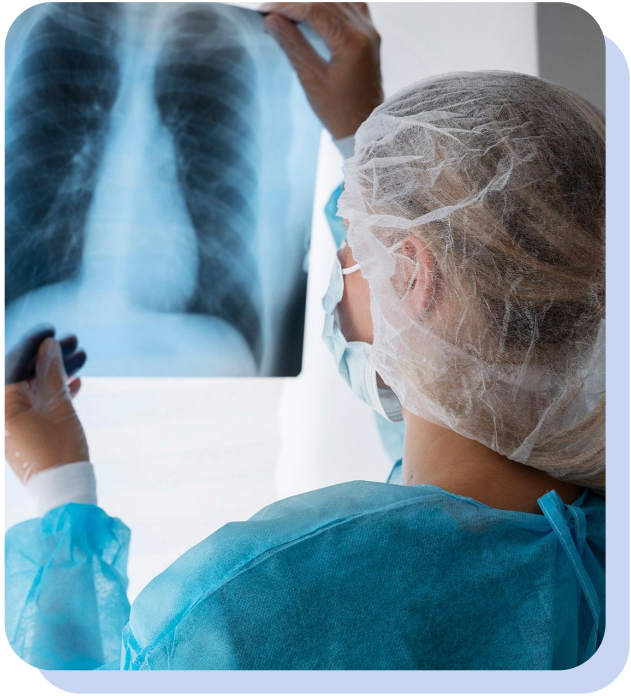
Respiratory Diagnosis Assistant Application

Team 8

Minjun Kim (mkim925@gatech.edu)
Nabin Kim (nabin@gatech.edu)
Sandra Kurian (skurian32@gatech.edu)
Seohee Yoon (syoon333@gatech.edu)

Our GOAL

- Automate lung sound classification for healthcare professionals for more effective patient care
- Develop a user-friendly app for querying lung sound files
    - Predict diseases based on the audio file
    - Instantly access relevant audio samples based on disease name and meta data
- Allow user contributions to improve database accuracy



COPD

Intended User

Medical Professionals

We can expect that the medical professionals would use our application to

- Streamline their workflow
- Access relevant patient information efficiently
- Interpret diagnostic results

Data

Available Dataset

- Demographic_info.text
  - 126 patient Information
  - Patient number, Age, Sex,...
- Audio files and text files
  - Lung sound files for each patient
  - Corresponding annotation files
  - Beginning of respiratory cycle, End of respiratory cycle,
- Patient_diagnosis.csv
  - Consist of patient number and disease name that the patient has
- Filename_format.txt
  - File naming convention

Data used for project

- Demographic_info.text
  - To show related patient information when user searches specific disease
- Audio files and txt files
  - To train our machine learning model for classifying a disease name with user input (audio files)
  - To provide further context for better diagnosis assist
- Patient_diagnosis.csv
  - To connect disease name that each patient has with the patient information and audio file

Database Design

MongoDB (NoSQL)

Used MongoDB to handle varying and complex data structures such as JSON files and array of embedded documents. Because we mainly retrieve data in bulks and in large volumes, we benefit from MongoDB's capability to read document-based data quickly.

Amazon S3

Used Amazon S3 to store large audio files and annotation text files. S3 is designed to handle large amounts of data in the form of objects.

Database Design

```
Patients
│
├── patient_id (AutoField, Primary Key)
├── age (IntegerField)
├── sex (CharField, max_length=10)
├── adult_bmi (FloatField, nullable)
├── child_weight (FloatField, nullable)
├── child_height (FloatField, nullable)
├── RespiratortyData (ArrayField)
│   └── Embedded Documents:
│       ├── id (IntegerField, Primary Key)
│       ├── recording_index (CharField, max_length=3)
│       ├── chest_location (CharField, max_length=2)
│       ├── acquisition_model (CharField, max_length=2)
│       ├── recording_equipment (CharField,
max_length=20)
│       ├── annotation_file (CharField, max_length=255)
│       ├── respiratory_cycles (JSONField)
│       └── sound_file_path (CharField, max_length=255)
├── average_cycle_duration(FloatField, nullable)
│
└── Diagnosis (Foreign Key Relationship to Diagnosis)
    └── Related Documents:
        ├── patient_id (ForeignKey, references
Patients.patient_id)
        └── diagnosis_name (CharField, max_length=100)
```

Our **Patients** model is the central model, managing data related to all patients.

patient_id is the primary key for this model, and this will later be used to refer **Patients** from **Diagnosis**.

RespiratoryData contains detailed data on respiratory analyses, and it is an array of embedded documents.

annotation_file and *sound_file_path* points to the annotation file (.txt) and sound file (.wav) in Amazon S3.

Diagnosis references to the **Patients** model and contains the diagnosis information (disease name).

We show two ways of how MongoDB handles relationships between documents, embedded documents and manual referencing.

We also showcase MongoDB's aggregation pipeline in calculating *average_cycle_duration*, enabling us to significantly reduce calculation time.

Django

Python web framework

Object-Relational Mapping (ORM)

- abstract details of SQL queries
- define database models using Python classes
    - Django handles the translation to SQL queries

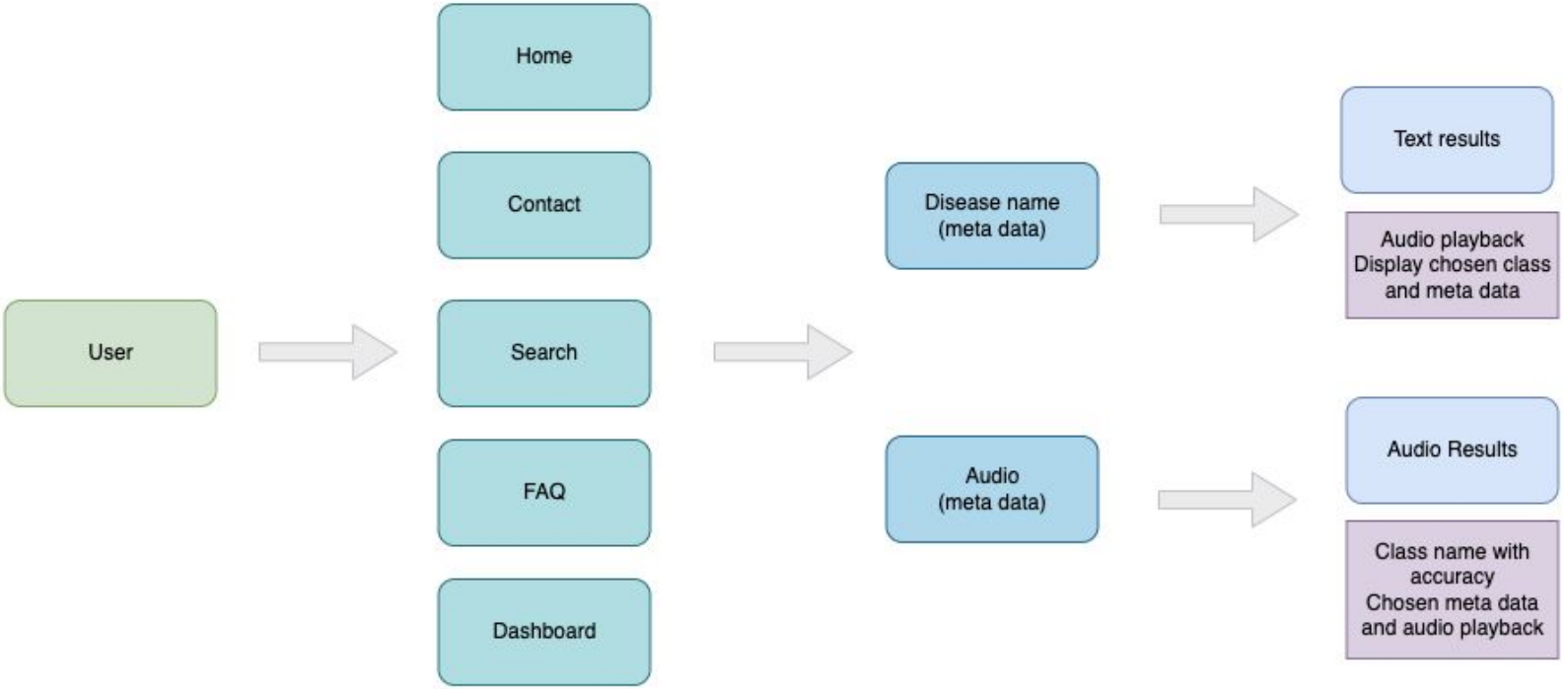
admin interface

- manage data through a web interface without having to write custom admin panels

scalable

- capable to handle high traffic loads
- (Instagram/Pinterest use Django)

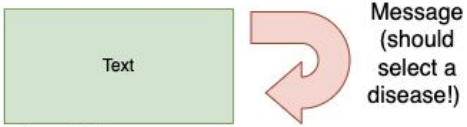
Application Layout



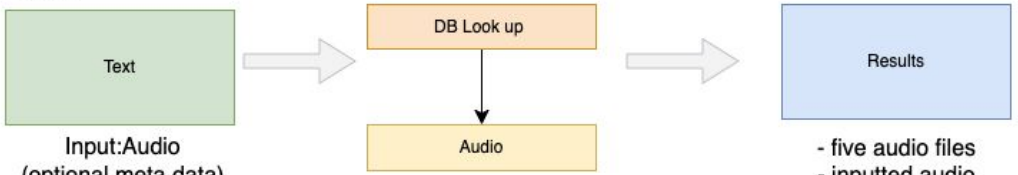
Search/Results Page

Text

Invalid

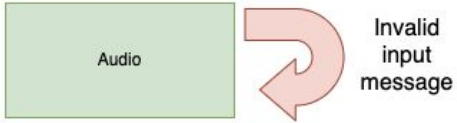


Valid

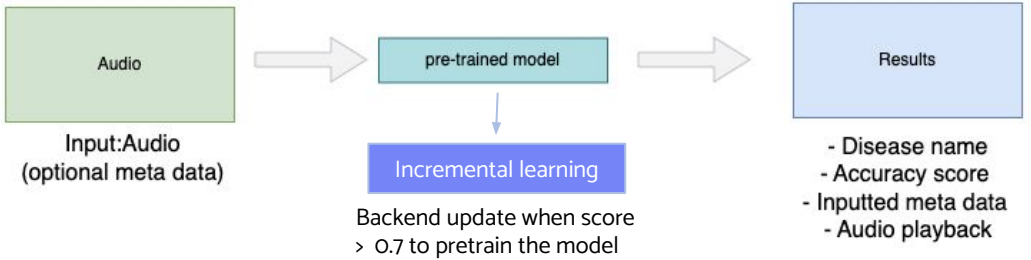


Audio

Invalid



Valid



Machine Learning Function

Audio Classification

- **Input**: valid Audio file (extension, file length, actual lung disease sound)
- **Process**: audio files using Mel Frequency Cepstral Coefficients (MFCC) to extract features
- **Output**: Disease name with similarity score & original audio file and demographics
- Trained model using these features instead of raw audio files

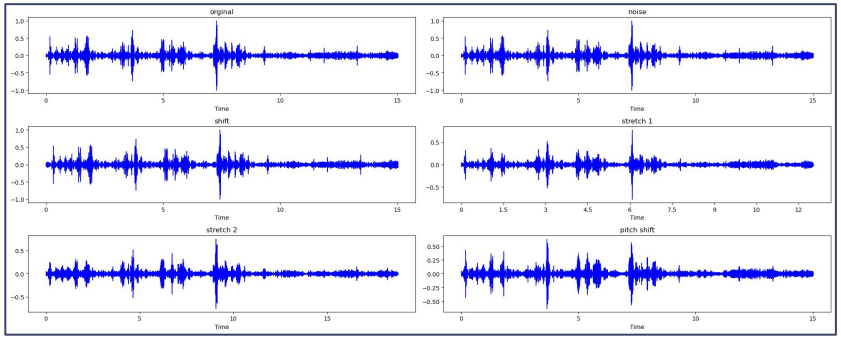
Limitations

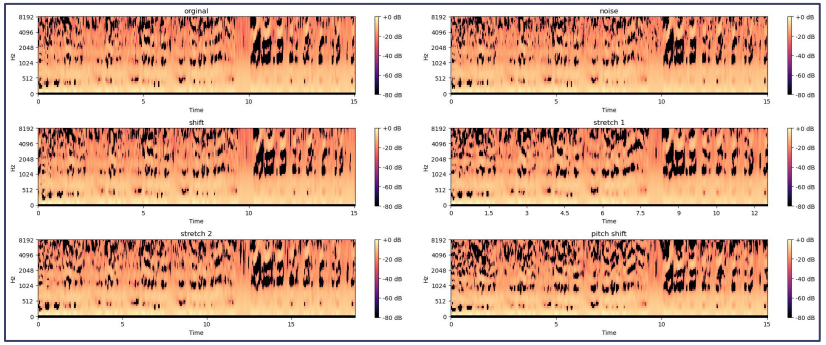
- Lower performance with metadata (chest location, bmi, age, etc) when trained together
- Limited number of classes
    - Currently, our model can classify 6 respiratory diseases for now

Dataset Problem

Dataset Size

Solution - Data Augmentation





Choosing Machine Learning Model

🔍 Best ML model for Audio Classification

- GRU (Gated recurrent units)
  - Designed for sequential data with temporal dependencies
    - Variable length sequences
      - Fewer parameters

- **GRU** has the best performances for all three metrics
- Used pandas, numpy, and tensorflow to implement GRU

Machine Learning Algorithm Comparison

Algorithm	Accuracy	Precision	Recall
GRU	**84%**	**87%**	**83%**
CNN	71%	77%	71%
SGD Classifier	68%	64%	64%
Ridge Classifier	67%	57%	64%

Challenge

Data Limitations

Initial model: Limited data ➡️ 70% accuracy.

Solution: Data augmentation ➡️ 83% accuracy.

Choosing database systems

We had many options in terms of database systems, but the question essentially came down to SQL or NoSQL.
Here are a few reasons why we chose MongoDB(NoSQL):

1. With our large volume of data, MongoDB's horizontal scalability is appealing.
2. With Django's ORM allows us to define data models and relationships as you would do in SQL. Along with this, through djongo, we can leverage benefits of both the ORM and MongoDB. This includes the previously mentioned manual referencing and embedded documents
3. MongoDB's aggregation pipeline enables real-time data processing and complex transformations directly within the database. This was shown in calculating average respiratory cycle.
4. We retrieve data in bulks and mainly all together. With MongoDB's referencing techniques, we don't have to perform join operations each time we retrieve data.
5. When integrating potential other differently structured datasets, MongoDB's dynamic schema allows us to modify our data fields with ease.

Takeaway

Machine learning

- Audio data processing
- Model exploration

Database management

- System selection factors
- Integration with Django
- Storing pre-trained models
- Database optimization

Possible Extensions

EHR

Integrate with existing EHR systems for streamlined data exchange and interoperability.

Collaborative features

Foster collaboration among medical professionals with integrated messaging and discussion features

Transfer learning

Integrate external datasets to enrich the existing database and utilize transfer learning for model adaptation (Hoang et al., 2023)

Mobile Accessibility

Ensure mobile accessibility for on-the-go access to patient data and diagnostic tools

References

- Dataset: Link

Hoang, T. V., Nguyen, Q. H., Nguyen, C. Q., Nguyen, P. X., & Nguyen, H. D. (2023, September 4). Sound-Dr: Reliable Sound Dataset and Baseline Artificial Intelligence System for Respiratory Illnesses. PHM Society Asia-Pacific Conference. https://papers.phmsociety.org/index.php/phmap/article/view/3604

Thanks

Do you have any questions?