PneumonoBot

2024 - Generative AI Bootcamp

Submitted by:

Muhammad Faraz Ali 2024-GenAI-33

Muhammad Yaqoob 2024-GenAI-38

Supervised by:

Ms. Ayesha Azam

Department of Computer Science

University of Engineering and Technology

Table of Content

Table Of Figures

List Of Tables

1. Introduction

1.1. Background

Pneumonia is a significant global health issue, affecting millions of people each year, particularly young children, the elderly, and individuals with compromised immune systems. According to the World Health Organization (WHO), pneumonia is the leading infectious cause of death among children under the age of five, contributing to over 740,000 child deaths in 2019. Pneumonia is a form of acute respiratory infection that affects the lungs, causing the alveoli—small sacs responsible for oxygen exchange—to fill with fluid or pus, leading to breathing difficulties and reduced oxygen intake.

The disease is caused by a variety of pathogens, including viruses, bacteria, and fungi. Common bacterial causes include *Streptococcus pneumoniae* and *Haemophilus influenzae type b (Hib)*, while viral causes often involve the respiratory syncytial virus. In populations with weakened immune systems, such as HIV-infected infants, pneumonia caused by fungi, such as *Pneumocystis jiroveci*, can be particularly deadly.

Despite its severity, pneumonia can be prevented through vaccination, adequate nutrition, and improved environmental conditions, such as reducing indoor air pollution. It can also be treated effectively with low-cost antibiotics. However, many children, especially in low-resource settings, do not receive the treatment they need due to barriers like limited access to healthcare services, lack of medical professionals, and inadequate health infrastructure. In regions such as sub-Saharan Africa and southern Asia, where healthcare resources are limited, pneumonia remains a major contributor to child mortality.

In recent years, advances in artificial intelligence (AI) and machine learning have provided new opportunities for improving pneumonia diagnosis and treatment. Automated diagnostic tools that classify chest X-rays using AI can help address the shortage of radiologists, particularly in underserved areas. Additionally, conversational AI systems can provide patients and caregivers with crucial information about pneumonia, its symptoms, and treatment options, helping to bridge the knowledge gap and promote timely medical intervention. Combining these technologies can create a robust solution to improve healthcare access and outcomes for those at risk of pneumonia, especially children.

1.2. Problem Statement

Pneumonia is a life-threatening respiratory infection that affects millions of people globally, particularly young children, the elderly, and individuals with weakened immune systems. Early detection and accurate diagnosis of pneumonia are critical for ensuring effective treatment and reducing the risk of complications or death. Chest X-ray imaging is the most widely used diagnostic tool to identify pneumonia; however, it requires interpretation by trained radiologists, which can be a resource-intensive and time-consuming process. This challenge is exacerbated

in underserved regions where access to healthcare professionals and diagnostic services is limited. In such scenarios, delayed diagnosis can lead to severe complications, including prolonged illness or death, due to lack of timely medical intervention.

Moreover, the lack of accessible and reliable health information regarding pneumonia often leaves patients and caregivers uncertain about symptoms, preventive measures, and treatment options. Patients frequently struggle to access trustworthy resources for understanding the disease, and this can hinder them from seeking timely care or following prescribed treatments effectively. There is an evident need for a system that not only provides an accurate diagnostic solution but also empowers users with reliable information related to pneumonia.

In this context, the problem is to develop a dual-functionality system that integrates artificial intelligence (AI) for medical image classification and natural language processing (NLP) for conversational interaction. Specifically, the system should leverage a machine learning model to automatically classify chest X-ray images as either 'Pneumonia' or 'Normal,' thereby providing a quick and accurate diagnosis. This would not only reduce the burden on radiologists and medical professionals but also ensure that patients receive timely diagnostic feedback, especially in resource-limited environments.

In addition to diagnostic capabilities, the system should incorporate a chatbot feature that enables users to ask questions related to pneumonia and receive accurate, informative responses. The chatbot should be designed to assist users with queries about symptoms, preventive measures, treatment options, and general health advice on pneumonia. This will bridge the information gap by offering easily accessible educational resources, promoting awareness, and encouraging informed decision-making among users.

1.3. Motivation

Pneumonia remains a critical global health challenge, particularly affecting vulnerable populations such as young children and individuals in low-resource settings. Despite being preventable and treatable, pneumonia continues to be the leading infectious cause of death among children worldwide. In 2019 alone, it claimed the lives of 740,180 children under the age of five, accounting for 14% of all child deaths in this age group. This tragic statistic underscores the pressing need for innovative solutions to combat pneumonia, particularly in regions such as southern Asia and sub-Saharan Africa, where mortality rates are disproportionately high.

The infectious nature of pneumonia, caused by various agents such as viruses, bacteria, and fungi, compounds the difficulty in managing the disease. Common pathogens like Streptococcus pneumoniae, Haemophilus influenzae type b (Hib), and the respiratory syncytial virus present significant threats, while certain populations, such as HIV-infected infants, face even higher risks due to weakened immune systems. The transmission of these pathogens through airborne droplets or blood, particularly during childbirth, further complicates prevention efforts. For these

reasons, timely and accurate diagnosis, followed by appropriate treatment, is crucial in preventing the progression of pneumonia to severe or fatal stages.

While pneumonia is largely preventable through immunization, proper nutrition, and addressing environmental factors like indoor air pollution, these interventions remain inaccessible to many. Alarmingly, only one-third of children with bacterial pneumonia receive the antibiotics they need, often due to limited access to healthcare facilities or trained medical professionals. In settings where healthcare resources are scarce, delayed treatment can lead to catastrophic outcomes. This gap in access to timely medical intervention highlights the need for low-cost, scalable technologies that can assist in the diagnosis and management of pneumonia.

Furthermore, many caregivers lack the knowledge to recognize the early signs of pneumonia, exacerbating the problem. Viral and bacterial pneumonia present similarly, often causing confusion. Symptoms such as rapid breathing, chest indrawing, and wheezing, while indicative of the disease, are frequently misinterpreted or dismissed, delaying care. Additionally, overcrowded living conditions, parental smoking, and poor hygiene further elevate the risk of children contracting pneumonia, highlighting the importance of educating caregivers and families.

This project aims to address these challenges by developing an AI-powered chatbot system capable of classifying chest X-ray images to detect pneumonia and providing instant feedback. Alongside image classification, the system will also offer an interactive platform where users can ask questions about pneumonia, its symptoms, treatments, and prevention. This combination of diagnostic and educational support seeks to bridge the healthcare gap, particularly in underserved areas, by providing both automated medical insights and reliable health information.

By leveraging AI technology to support pneumonia diagnosis and improve access to health education, this project holds the potential to make a substantial impact on reducing child mortality rates and empowering families to take informed actions regarding their children's health.

1.4. Scope

The scope of this project focuses on the development and implementation of an AI-powered chatbot system designed to assist in the diagnosis and management of pneumonia. The system will consist of two main functionalities:

1. Chest X-ray Image Classification: Using a ViT machine learning model, the system will classify chest X-ray images into two categories: 'Pneumonia' or 'Normal'. This automated diagnostic tool will assist healthcare providers, especially in low-resource settings, by providing accurate and timely pneumonia diagnosis. The system aims to achieve high

accuracy in distinguishing between healthy lung images and those indicative of pneumonia, reducing the need for manual interpretation by trained radiologists.

2. Conversational AI for Pneumonia Information: The chatbot will provide users with an interactive platform to ask questions related to pneumonia. It will respond to queries about pneumonia symptoms, prevention, treatment options, and general health advice. The chatbot will be equipped with a predefined knowledge base that is designed to deliver medically accurate and evidence-based responses, ensuring users receive reliable health information.

This dual functionality will enable the system to address both diagnostic and educational needs, empowering users with timely health insights and access to crucial medical information. The project targets healthcare facilities, caregivers, and individuals in resource-limited settings, where access to trained medical professionals and educational materials may be insufficient. Additionally, the system will be scalable and adaptable, allowing for future enhancements to incorporate other types of respiratory infections or health conditions.

By integrating AI into pneumonia diagnosis and education, this project aims to reduce pneumonia-related mortality, particularly among children, and to make healthcare more accessible to populations in need.

2. Dataset Description

The dataset utilized in this project is a collection of chest X-ray images designed for the classification of pneumonia versus normal cases. It comprises 5,863 images, organized into three primary folders: train, test, and val, each containing subfolders for the two image categories: Pneumonia and Normal. This dataset is derived from a study published in Cell and is primarily focused on anterior-posterior chest X-rays of pediatric patients.

1.1. Dataset folder Structure

The dataset is divided into the following three directories:

1. Train
2. Test
3. Val

Each directory is further categorized into two subfolders:

1. Pneumonia
2. Normal

1.2. Data Card

Figure 1. Dataset Card

1.3. Dataset Statistics

Category	**Train**	**Test**	**Val**	**Total**
Pneumonia	3875	390	8	4273
Normal	1341	234	8	1583
Total	5216	624	16	5856

Table 1. Dataset Statics

1.4. Dataset statistics Bar graph Visualization

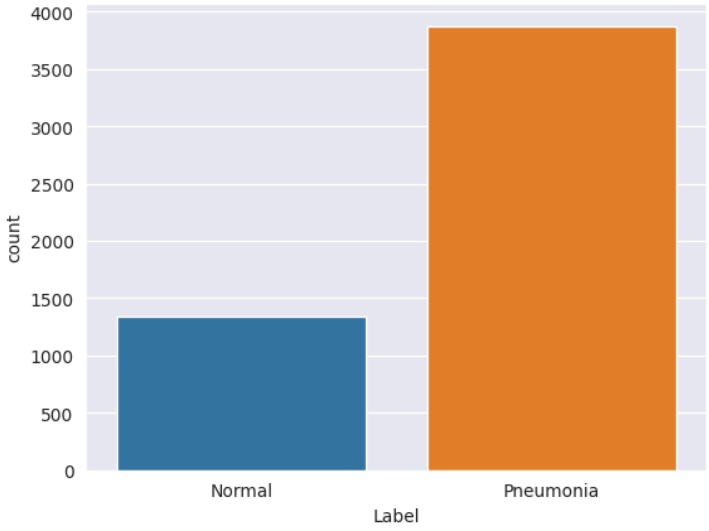

Figure 2. Dataset statistics with Bar graph

3. Methodology - for Pneumonia classification

3.1. Methodology diagram for Pneumonia Classification

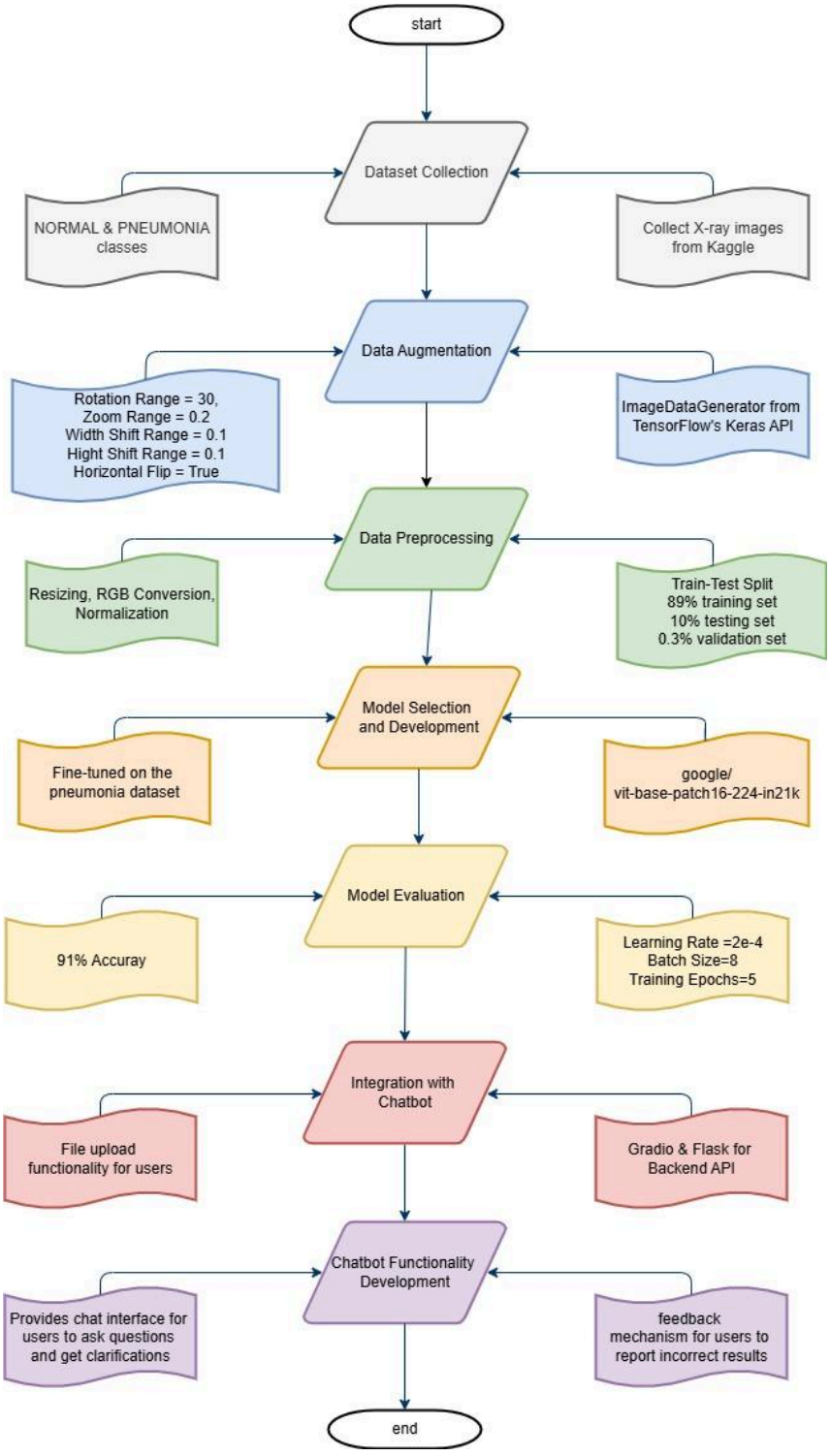

Figure 3. Pneumonia Classification Methodology

Below is the step by step explanation of all the steps followed during the pneumonia classification.

3.2. Dataset Collection

The first step involved was gathering X-ray images of both normal and pneumonia cases. The dataset used in this classification was Chest X-Ray Images (Pneumonia). The labels were the names of the folders. There were two classes in the dataset. First one was PNEUMONIA and the second one was NORMAL. The dataset contained a substantial number of both categories to ensure a balanced and representative sample, crucial for achieving accurate predictions during model training and evaluation. Further details regarding the dataset can be found on the dataset description section.
Below is the code snippet for the dataset collection.

```
!kaggle datasets download -d paultimothymooney/chest-xray-pneumonia
!unzip chest-xray-pneumonia.zip -d chest_xray > /dev/null 2>&1
path = '/kaggle/working/chest_xray/chest_xray/chest_xray'
dataset = load_dataset('imagefolder', data_dir=path)
print("✔")
```

3.3. Test-Train Split

We have used the default percentage for test-train split. The default split percentages for the load_dataset function in the Hugging Face datasets library are as follows:

- Training split: 90%
- Validation split: 5%
- Test split: 5%

```
from datasets import load_dataset

# Split dataset
dataset_train = dataset['train']
dataset_val = dataset['validation']
dataset_test = dataset['test']

num_classes = len(set(dataset_train['label']))
labels = dataset_train.features['label']
num_classes, labels
```

3.4. Data Augmentation

To expand the training dataset and prevent overfitting, data augmentation techniques were applied. Using TensorFlow's Keras ImageDataGenerator, a variety of transformations were performed on the images. These included operations such as rotation, zoom, width and height shifts, and horizontal flips. These transformations allowed the model to generalize better by simulating various conditions and variations that might appear in real-world X-ray images.

Category	Train	Test	Val	Total
Pneumonia	7750	390	8	8148
Normal	2682	234	8	2924
Total	10432	624	16	11072

Table 2. Dataset After Data Augmentation

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from PIL import Image
import matplotlib.pyplot as plt

# A function to convert images to a consistent format
def preprocess_images(images, target_size=(224, 224)):
    processed_images = []
    for img in images:
        if isinstance(img, np.ndarray):
            # Ensure images are in RGB format and resize
            img = Image.fromarray(img).convert("RGB")
        elif isinstance(img, Image.Image):
            img = img.convert("RGB")
        # Resize image
        img = img.resize(target_size)
        # Convert image to numpy array and normalize to [0, 1]
        img_array = np.array(img) / 255.0
        processed_images.append(img_array)
    return np.array(processed_images)

# Initialize ImageDataGenerator with data augmentation
```

```
datagen = ImageDataGenerator(
    rotation_range=30,
    zoom_range=0.2,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=False
)

def augment_images(images, labels):
    # Preprocess images
    images = preprocess_images(images)
    labels = np.array(labels)

    # Create a generator for augmented images
    datagen.fit(images)

    # Use the generator to get augmented images
    augmented_images = []
    augmented_labels = []
    for batch in datagen.flow(images, labels, batch_size=32,
shuffle=False):
        augmented_images.extend(batch[0])
        augmented_labels.extend(batch[1])
        if len(augmented_images) >= len(images):
            break

    return np.array(augmented_images), np.array(augmented_labels)

images = [item['image'] for item in dataset_train]
labels = [item['label'] for item in dataset_train]
augmented_images, augmented_labels = augment_images(images, labels)
```

3.5. Data Augmentation Visualization

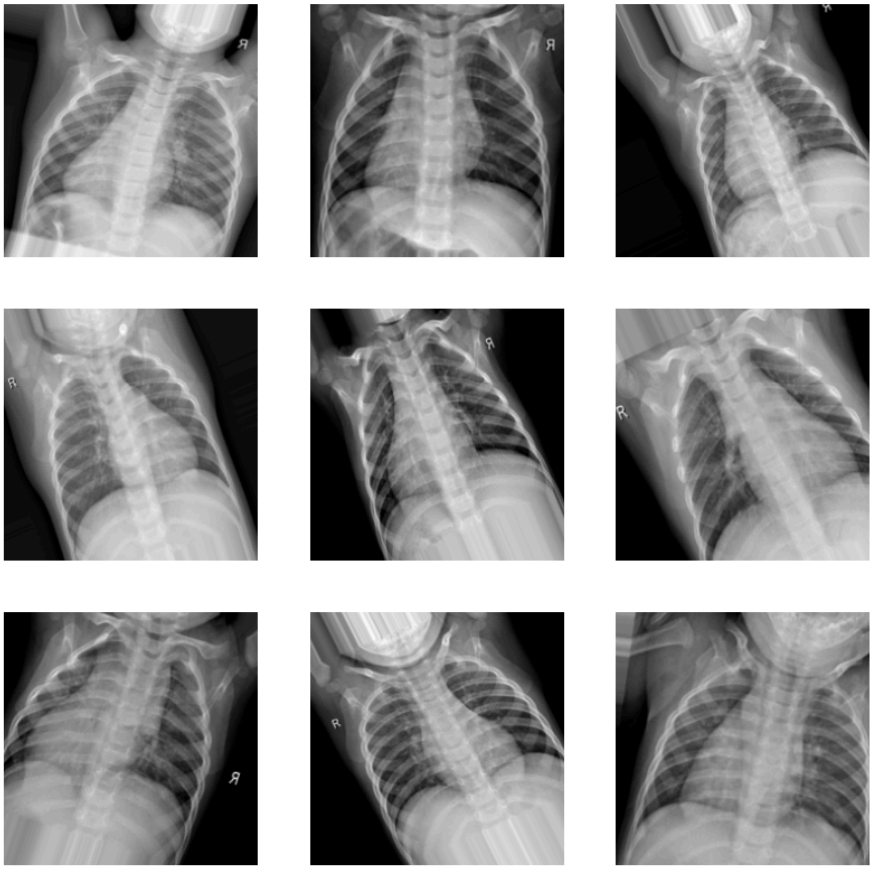

Figure 4. Data Augmentation Results Visualization

```
import matplotlib.pyplot as plt

# Display some augmented images
plt.figure(figsize=(10, 10))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[i])
    plt.axis('off')
plt.show()
```

3.6. Data Preprocessing

The preprocessing stage involved standardizing the input data to ensure that the images were consistent and ready for training. Each image was resized to a uniform shape, converted to the

RGB color space, and normalized to scale pixel values. The data was then split into training and testing sets to ensure that the model could be properly evaluated after development.

```
from PIL import Image

def preprocess(batch):
    # Convert images to RGB if they are not already
    images = [img.convert("RGB") if isinstance(img, Image.Image) else
Image.fromarray(img).convert("RGB") for img in batch['image']]
    inputs = feature_extractor(
        images,
        return_tensors='pt'
    )
    inputs['label'] = batch['label']
    return inputs


prepared_train = combined_dataset.with_transform(preprocess)
prepared_val = dataset_val.with_transform(preprocess)
prepared_test = dataset_test.with_transform(preprocess)
```

3.7. Model Selection and Development

For this classification task, a Vision Transformer (ViT) architecture was chosen due to its high performance on image classification problems. The google/vit-base-patch16-224-in21k pre-trained model was selected and fine-tuned using the pneumonia dataset. This allowed the model to leverage pre-existing knowledge gained from large-scale image datasets, thus improving the efficiency and accuracy of the pneumonia detection task.

3.7.1. Model Selection

```
from transformers import TrainingArguments
from transformers import ViTForImageClassification
import torch

model_name_or_path = 'google/vit-base-patch16-224-in21k'
feature_extractor =
ViTFeatureExtractor.from_pretrained(model_name_or_path)
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = ViTForImageClassification.from_pretrained(
    model_name_or_path,
    num_labels=num_classes
)

model.to(device)
```

3.7.2 Modal Development

```
from transformers import TrainingArguments
import wandb
import json

def collate_fn(batch):
    return {
        'pixel_values': torch.stack([x['pixel_values'] for x in batch]),
        'labels': torch.tensor([x['label'] for x in batch])
    }
metric = load_metric("accuracy",trust_remote_code=True)

def compute_metrics(p):
    return metric.compute(
        predictions=np.argmax(p.predictions, axis=1),
        references=p.label_ids

    )

training_args = TrainingArguments(
    output_dir="./Pneumonia_Dataset",
    per_device_train_batch_size=8,
    eval_strategy="steps",
    num_train_epochs=5,
    save_steps=100,
    eval_steps=100,
```

```
    logging_steps=10,
    learning_rate=2e-4,
    save_total_limit=2,
    remove_unused_columns=False,
    push_to_hub=False,
    load_best_model_at_end=True,
)
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=collate_fn,
    compute_metrics=compute_metrics,
    train_dataset=prepared_train,
    eval_dataset=prepared_val,
    tokenizer=feature_extractor,
)
# Your training code
train_results = trainer.train()

# Save the model and tokenizer
trainer.save_model()

# Save metrics
trainer.save_metrics("train", train_results.metrics)
trainer.save_state()

# Evaluate the model
metrics = trainer.evaluate(prepared_test)

# Save evaluation metrics
trainer.save_metrics("eval", metrics)
```

3.8. Model Evaluation

Once trained, the model was evaluated on a separate test dataset to assess its performance. The model achieved an accuracy rate of 91%, which indicates that it correctly classified the

majority of the test images. Additional performance metrics such as precision, recall, and F1 score were also calculated to ensure a comprehensive understanding of the model's strengths and areas for improvement.

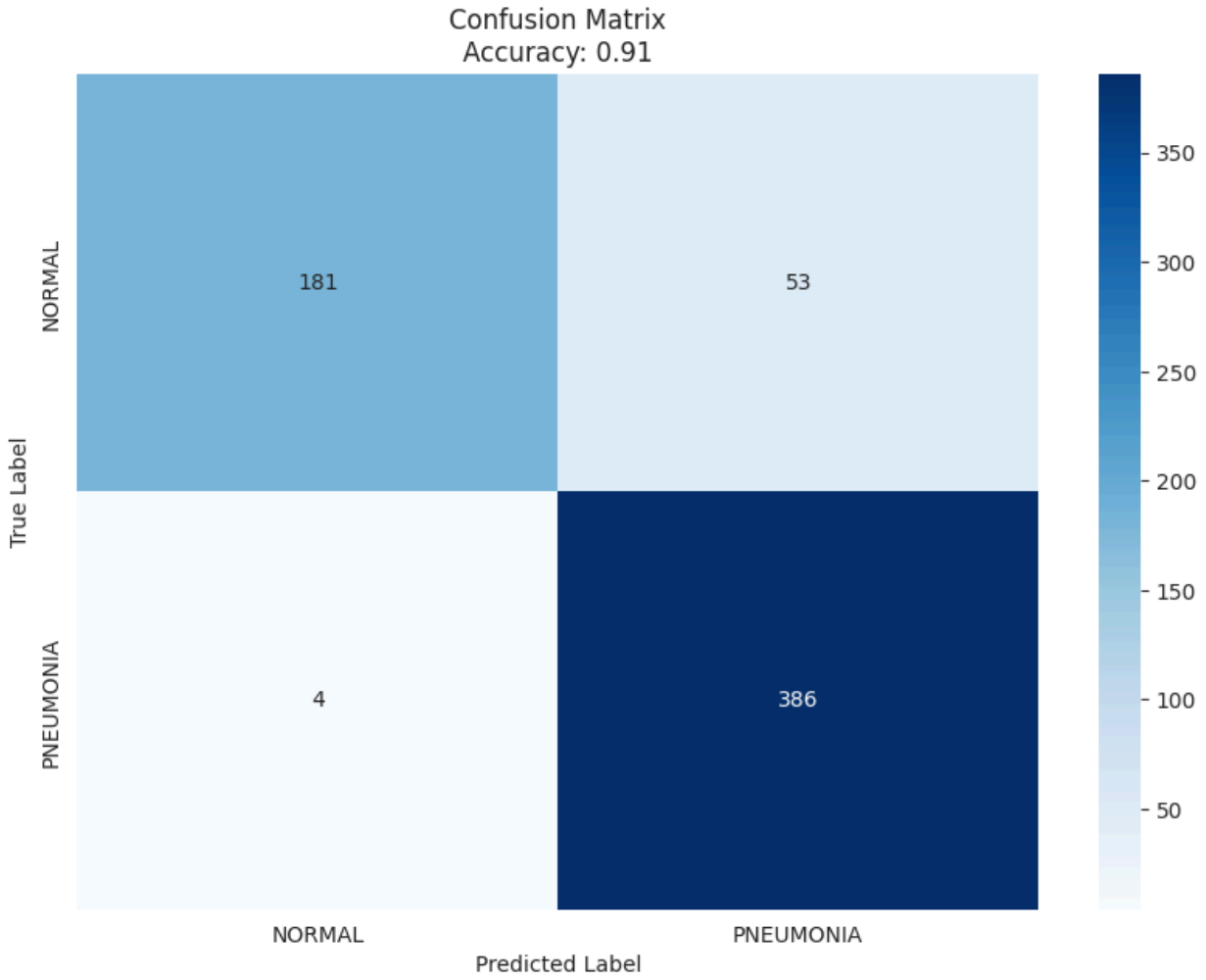

Figure 5. Confusion Matrix

```
# Make predictions on the test set
test_predictions = trainer.predict(prepared_test)
preds = np.argmax(test_predictions.predictions, axis=1)
labels = test_predictions.label_ids

image_filenames = []
for img in dataset_test['image']:
    if isinstance(img, str):
        image_filenames.append(img)
    elif hasattr(img, 'filename'):
image_filenames.append(os.path.basename(img.filename))
```

```
    else:
        image_filenames.append('Unknown')
# Compute the confusion matrix
cm = confusion_matrix(labels, preds)
accuracy = accuracy_score(labels, preds)

# Display the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=dataset_test.features['label'].names,
yticklabels=dataset_test.features['label'].names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title(f'Confusion Matrix\nAccuracy: {accuracy:.2f}')
plt.show()

print(f'Accuracy: {accuracy:.2f}')
```

3.9. Upload Modal to Hugging Face

```
from huggingface_hub import notebook_login

notebook_login()

from huggingface_hub import create_repo

# Create a new repository on Hugging Face Hub
repo_id = "M-Yaqoob/PneumonoBot"
create_repo(repo_id, repo_type="model", private=False)

from huggingface_hub import HfApi, HfFolder

# Load the model and tokenizer saved by Trainer
model_dir = "/kaggle/working/Pneumonia_Dataset"

# Upload the model to Hugging Face
api = HfApi()
```

```
api.upload_folder(
    folder_path=model_dir,
    repo_id="M-Yaqoob/PneumonoBot",
    repo_type="model",
    commit_message="Upload chest xray pneumonia model"
)
```

3.9. Integration with Chatbot

To provide a user-friendly interface, the model was integrated with a chatbot. The backend was implemented using Flask, with Gradio providing the interface for user interaction. Users could upload X-ray images directly into the system, and the chatbot would then return a classification indicating whether the image showed signs of pneumonia. This integration allowed for an easy and interactive way for users to obtain results from the model.

```
# Prediction endpoint
@app.route('/predict', methods=['POST'])
def predict():
    try:
      if 'file' not in request.files:
          return jsonify({'error': 'No file part'})

      file = request.files['file']
      if file.filename == '':
          return jsonify({'error': 'No selected file'})
      image = Image.open(file).convert("RGB")
      predicted_class, confidence = predict_tb(image)
      label_map = {0: 'No Pneumonia', 1: 'Pneumonia'}
      prediction = label_map[predicted_class]

      return jsonify({'label': prediction, 'confidence': confidence})
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

3.10. Chatbot Functionality Development

In addition to classification functionality, the chatbot was designed to engage users in conversation, answer common questions about pneumonia, and offer clarifications on the results. Furthermore, a feedback system was incorporated, allowing users to report incorrect

results or provide suggestions for improvement. This feedback mechanism is integral to refining the model and the overall user experience.

```
import gradio as gr
from qdrant_embeddings_rag_chatbot import model_inference, chatbot

# Chat interface block
with gr.Blocks(
        css="""
        .gradio-container .avatar-container {height: 40px; width: 40px
!important;}
        #duplicate-button {margin: auto; color: white; background:
#f1a139; border-radius: 100vh; margin-top: 2px; margin-bottom: 2px;}
        """,
) as chat:
    gr.ChatInterface(
        fn=model_inference,
        chatbot = chatbot,
        multimodal=True,
        autofocus=True,
        concurrency_limit=10,
    )

# Main application block
with gr.Blocks() as demo:
    gr.TabbedInterface([chat], ['💬 SuperChat'])

demo.queue(max_size=300)
demo.launch(share=True)
```

4. Methodology for RAG Implementation

The primary goal of this methodology is to build a retrieval-augmented generation (RAG) system that can accurately answer questions based on information extracted from a large corpus of text. This project is focused on answering queries related to pneumonia, utilizing a medical books

1. Pneumonia: Symptoms, Diagnosis and Treatment : Symptoms, Diagnosis and Treatment, edited by Micaela L. Suárez, and Steffani M. Ortega, Nova

2. PNEUMONIA UNMASKED: UNLOCKING THE SECRETS OF CHILDHOOD INFECTIONS by Arif Rohman Mansur and Ira Mulya Sari
3. Advanced Physiology and Pathophysiology: Essentials for Clinical Practice
4. An official website of the United States government

in PDF format.

4.1. Text Extraction

The first step involves extracting the raw text from the provided Pneumonia Books and the Official website of the United state in PDF format. This extracted text forms the basis for further processing and question-answering tasks.

```
from PyPDF2 import PdfReader
import re

def read_pdf_file(file_path):
    points = []
    # Open the PDF file
    with open(file_path, 'rb') as file:
        pdf_reader = PdfReader(file)
        # Iterate through each page in the PDF
        for page in pdf_reader.pages:
            text = page.extract_text()
            if text:
                points.append(text.strip())
    return points

def replace_double_spaces(text):
    cleaned_text = re.sub(r'\xa0', ' ', text)
    cleaned_text1 = re.sub(r'\u200b', '', cleaned_text)
    return cleaned_text1

def process_pdf_files(file_paths, chunking_type, max_chunk_size,
number_of_clusters=20):
    all_embeddings = []
    all_chunks = []

    for file_path in file_paths:
        print(f"Processing: {file_path}")
        Pneumonia = read_pdf_file(file_path)
```

```
        # Clean the text
        for i in range(len(Pneumonia)):
            Pneumonia[i] = replace_double_spaces(Pneumonia[i])

        Pneumonia_Str = "\n".join(Pneumonia)

        # Chunking
        if chunking_type == "Recursive":
            chunks = Recursive_Chunking(Pneumonia_Str,
chunk_size=max_chunk_size)
        elif chunking_type == "Semantic":
            chunks = semantic_chunking_with_attention(Pneumonia_Str,
n_clusters=number_of_clusters, max_chunk_size=max_chunk_size)

        # Get embeddings
        embedding = get_sentenceTF_embeddings(chunks)

        all_embeddings.extend(embedding)
        all_chunks.extend(chunks)

    return all_embeddings, all_chunks
```

4.2. Chunking Strategy

To ensure that the extracted text can be effectively processed by large language models (LLMs), the text is divided into smaller, more manageable segments called "chunks." This project employs Semantic Chunking with a chunk size of 512 tokens. The chunks are formed based on semantic meaning rather than arbitrary length boundaries, ensuring that each chunk represents a coherent unit of information. This step is critical for retaining context within each chunk.

```
import torch
from transformers import BertTokenizer, BertModel
from sklearn.cluster import KMeans


def get_bert_embeddings(sentences):
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
    model = BertModel.from_pretrained('bert-base-uncased')

    # Tokenize sentences
    inputs = tokenizer(sentences, return_tensors='pt', padding=True,
truncation=True, max_length=128)

    # Get BERT embeddings
    with torch.no_grad():
        outputs = model(**inputs)

    # Use the [CLS] token representation as the sentence embedding
    embeddings = outputs.last_hidden_state[:, 0, :].numpy()

    return embeddings

def semantic_chunking_with_attention(text, n_clusters=5,
max_chunk_size=512):
    # Split the input text into sentences based on '.'
    sentences = [sentence.strip() for sentence in text.split('.') if
sentence.strip()]

    # Get BERT embeddings for sentences
    embeddings = get_bert_embeddings(sentences)

    # Perform KMeans clustering
    kmeans = KMeans(n_clusters=n_clusters,
random_state=5).fit(embeddings)
    labels = kmeans.labels_

    # Group sentences by cluster
    clusters = {}
    for sentence, label in zip(sentences, labels):
        if label not in clusters:
            clusters[label] = []
        clusters[label].append(sentence)

    # Create chunks from clusters with max_chunk_size constraint
    chunks = []
    for cluster in clusters.values():
```

```
        current_chunk = ""
        for sentence in cluster:
            if len(current_chunk) + len(sentence) + 1 <= max_chunk_size:
                current_chunk += sentence + ". "
            else:
                chunks.append(current_chunk.strip())
                current_chunk = sentence + ". "
        if current_chunk:
            chunks.append(current_chunk.strip())

    return chunks
```

4.3. Embeddings Creation

After chunking the text, an embedding model is used to convert each chunk into vector representations that capture the semantic meaning of the text. In this project, the SentenceTransformer model (all-MiniLM-L6-v2) is employed for embedding generation. The embeddings are high-dimensional vectors that allow for effective similarity search and information retrieval later in the pipeline.

```
from sentence_transformers import SentenceTransformer
def get_sentenceTF_embeddings(sentences):
  model = SentenceTransformer('all-MiniLM-L6-v2')
  embeddings =[]
  for chunk in sentences:
    embeddings.append(model.encode(chunk))
  print(len(embeddings))
  return embeddings

def Embed_stenteceTF(sentence):
  model = SentenceTransformer('all-MiniLM-L6-v2')
  return model.encode(sentence)
```

4.4. Storage in Qdrant Vector Database

The generated embeddings are stored in a Qdrant Vector Database, a high-performance vector database that is designed for similarity search and nearest neighbor search. This storage

enables efficient querying of the chunks based on their embeddings, which is essential for retrieving relevant information during the query-answering process.

4.4.1. Create PointStruct objects

```
from qdrant_client.models import PointStruct
import uuid

# Create PointStruct objects
points = [
    PointStruct(
        id=str(uuid.uuid4()),
        vector=embedding,
        payload={"text": doc}
    )
    for i, (doc, embedding) in enumerate(zip(chunks, embeddings))
]
```

4.4.2. Initialize Qdrant Client

```
from qdrant_client import QdrantClient
import json

with open('config.json', 'r') as file:
    config = json.load(file)


url = config['qdrant_url']
api_key = config['qdrant_api_key']

# Initialize Qdrant client
qdrant_client = QdrantClient(url=url, api_key=api_key)
```

4.4.3. Create Collection

```
from qdrant_client.models import CollectionParams, VectorParams
```

```
# Define the dimension of your vectors and distance metric
vector_dim = 384
distance_metric = "Cosine"

# Define vector configuration
vector_params = VectorParams(size=vector_dim, distance=distance_metric)

# Create the collection
qdrant_client.create_collection(
    collection_name="pneumonoBot",
    vectors_config=vector_params
)
```

4.4.4. Insert to Qdrant

```
# Insert points into Qdrant
qdrant_client.upsert(
    collection_name="pneumonoBot",
    points=points
)
```

4.5. Query Embedding and Retrieval

When a user inputs a query, the same SentenceTransformer model (all-MiniLM-L6-v2) is applied to convert the query into an embedding. This query embedding is then compared with the embeddings stored in the Qdrant Vector Database. Using similarity search, the top 10 related chunks are retrieved, which are considered the most relevant to the query.

4.5.1. Create Embeddings of a Query

```
from sentence_transformers import SentenceTransformer

query = ["How to treat pneumonia"]
query_embedding_response = get_sentenceTF_embeddings(query)

def get_sentenceTF_embeddings(sentences):
  model = SentenceTransformer('all-MiniLM-L6-v2')
```

```
  embeddings =[]
  for chunk in sentences:
    embeddings.append(model.encode(chunk))
  print(len(embeddings))
  return embeddings

def Embed_stenteceTF(sentence):
  model = SentenceTransformer('all-MiniLM-L6-v2')
  return model.encode(sentence)
```

4.5.2. Convert embedding response to a list

```
query_embedding = query_embedding_response[0].tolist()
```

4.6. Context Creation

The retrieved chunks are combined to create a context for answering the query. This context is essentially a collection of the most relevant chunks of information, which is passed to the large language model (LLM) to generate a precise answer.

```
# Perform similarity search and retrieve Context
results = qdrant_client.search(
    collection_name="pneumonoBot",
    query_vector=query_embedding,
)
```

4.7. Integration with Chat History

In addition to the retrieved context, the system incorporates the Chat History from the ongoing conversation with the user. This step ensures that the model is aware of the conversational context and can generate more coherent and contextually relevant responses.

```
messages = []
prompt = {"role":"system", "content": custom_prompt(query, results)}
messages.append(prompt)
```

4.8. Answer Generation via LLaMA 3.1

The final context, including both the retrieved chunks and the chat history, is fed into the LLaMA 3.1 language model. LLaMA 3.1 generates a response that answers the user's query based on the given context. This large language model is capable of understanding and generating complex natural language, making it ideal for this task.

```
res = client.chat.completions.create(
  model="meta/llama-3.1-8b-instruct",
  messages=messages,
  temperature=0.2,
  top_p=0.7,
  max_tokens=1024,
  stream=True
)
```

4.9. Final Output

The system then delivers the generated response to the user. The answer is informed by both the most relevant chunks of the text and the prior conversation, resulting in a more accurate and context-aware output.

4.10. Methodology Diagram for RAG

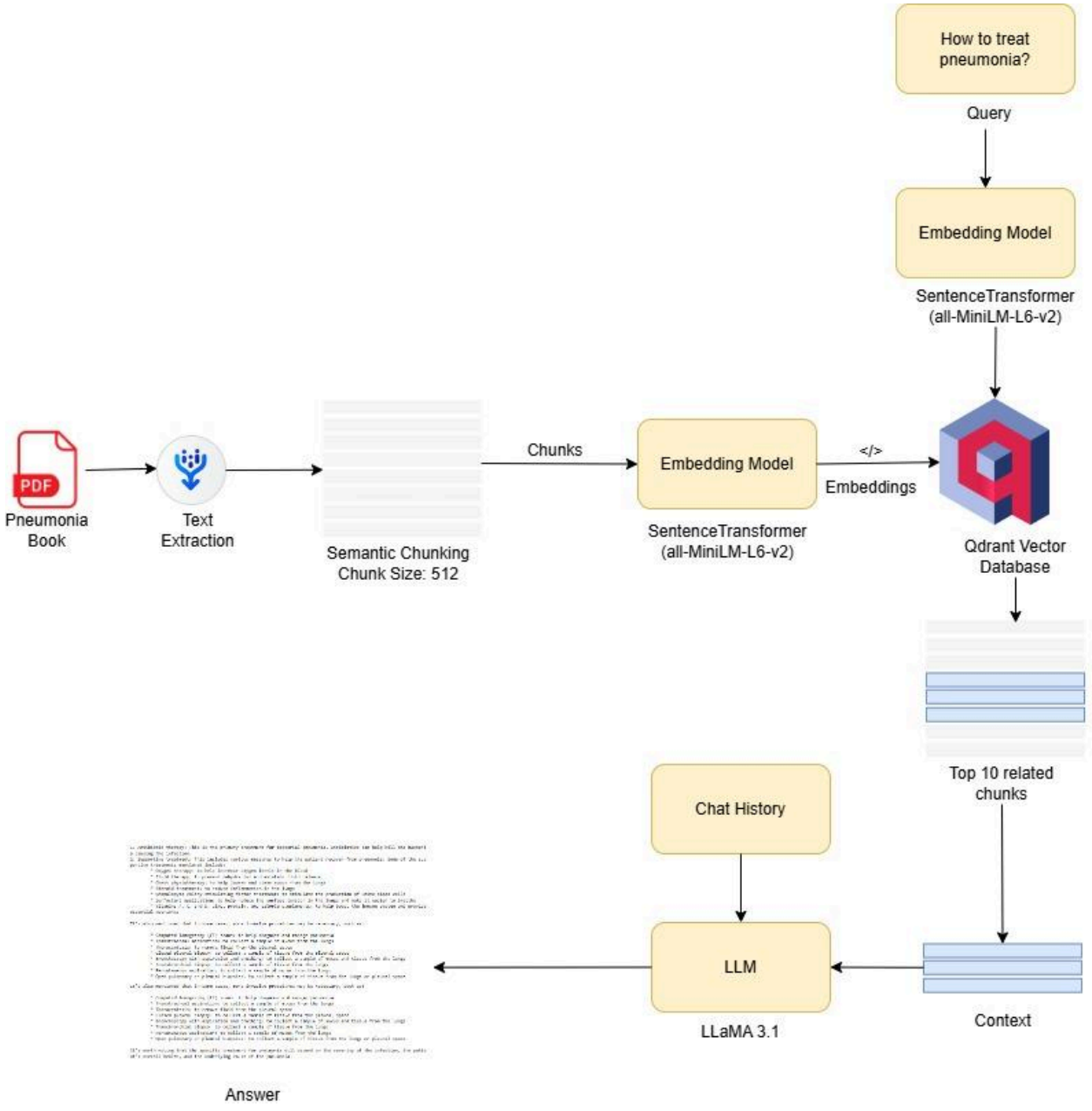

Figure 7. Retrieval Augmented Generation Methodology

5. Methodology diagram for a chatbot

Below is the simplest methodology diagram that we think any layman can understand. It is really easy to understand the flow of a chatbot where the user query to the chatbot related to the pneumonia and in the backside implementation, the chatbot gets the first 10 documents by applying the similarity search at Qdrant. We have also passed the History of the user chatting to

llama. So, it can produce better results. Finally, at last the generated response is visible to the user.

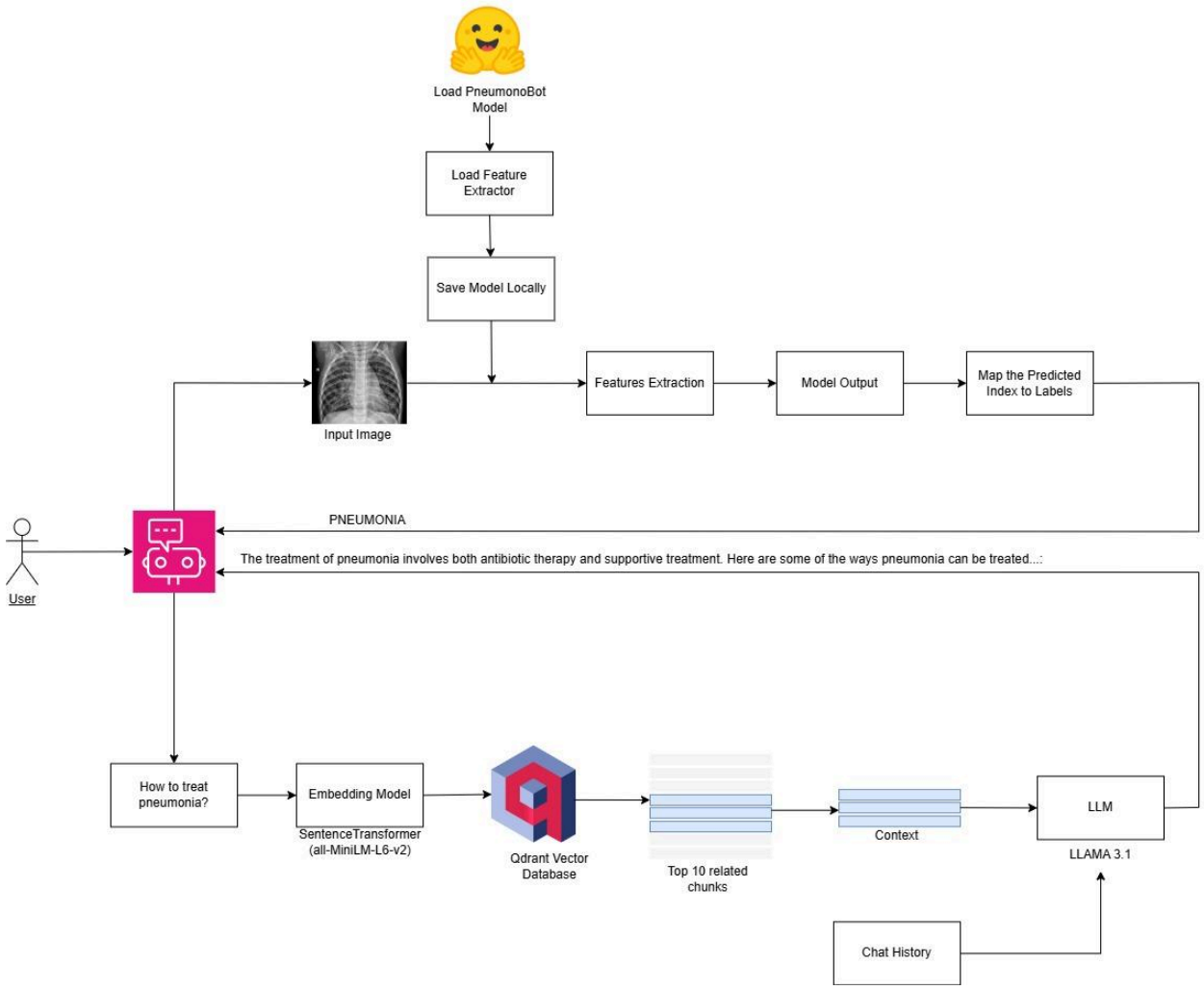

Figure 8. Methodology diagram for a chatbot

6. Tools Used

6.1. Hugging Face

We have uploaded our model to Hugging Face, a powerful platform that allows easy deployment and sharing of machine learning models. Hugging Face provides a centralized location where models can be stored, accessed, and reused efficiently, making it ideal for various NLP, computer vision, and other AI-based applications. By uploading the model to Hugging Face, we benefit from seamless integration into production environments and can easily call the model via their API. This enables us to test and use the model across different platforms, significantly improving accessibility and collaboration.

Furthermore, Hugging Face's infrastructure takes the burden of managing resources off our shoulders. Instead of worrying about maintaining servers or manually configuring environments, we can rely on Hugging Face's robust hosting capabilities. This allows us to focus more on developing and improving our model, knowing that it's backed by a scalable and reliable system. From testing to production, Hugging Face plays a crucial role in ensuring that our machine learning efforts are more efficient and effective.

Hugging face model can be found at: [PneumonoBot - Hugging Face Link]

Below is the screenshot of the Hugging Face model page.

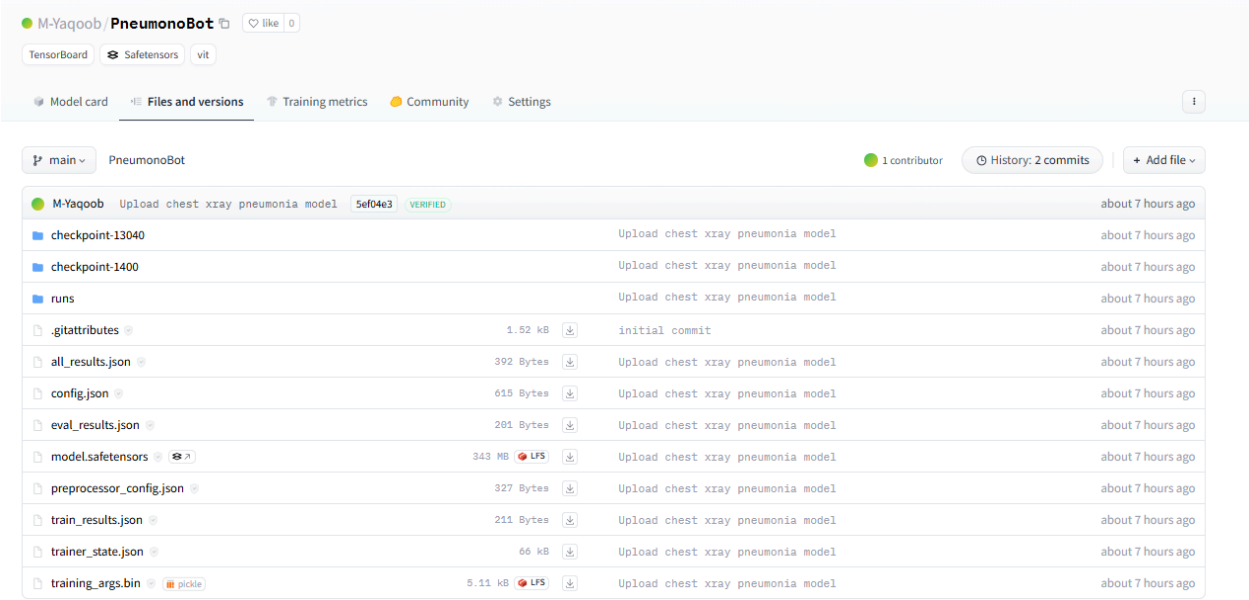

Figure 9. Tools Used - Hugging face

6.2. GitHub

We utilized GitHub to manage and organize our codebase efficiently, keeping all project files in well-structured directories. This helped us maintain a clean workflow as we progressed through the development phases.

- **Api Folder**: This folder contains the Flask API. In case we host our model, this API can be used to interact with it seamlessly.
- **Essentials:** This folder contains the books that were used for the Retrieval-Augmented Generation (RAG) process.
- **Pneumonia Classification Using CNN**: Initially, we implemented a Convolutional Neural Network (CNN) to classify pneumonia. However, the model achieved an accuracy of only 77%-80%. Due to the limited accuracy, we shifted our approach to fine-tuning a pre-trained model, specifically Vision Transformer (ViT).
- **Pneumonia Classification Using ViT**: This directory holds the code for fine-tuning the ViT model. We initially fine-tuned ViT without applying data augmentation but achieved

only 87% accuracy. After incorporating data augmentation techniques, we improved the accuracy to 91%, meeting our desired performance targets.

- **Save Embeddings Locally**: At first, we planned to save the embeddings locally for easier access. While this approach worked perfectly, we decided to explore a more creative solution and transitioned to uploading our embeddings to Qdrant.
- **Save Embeddings to Qdrant**: This folder chronicles the entire process of saving embeddings to Qdrant, from encountering errors to successfully uploading all embeddings. Early challenges included difficulty in finding the correct embedding model. Eventually, we resolved these issues by utilizing the Sentence Transformer model, which proved to be more suitable.
- **Save Model from Hugging Face to Locally**: This folder demonstrates how we download our model from Hugging Face and save it locally for ease of use. This eliminates the need to repeatedly load the model from Hugging Face during testing and usage.
- **Chatbot Folder**: The chatbot implementation can be found here. It contains two versions of the chatbot—one using local embeddings and another using embeddings from Qdrant. Currently, we are using the Qdrant-based embeddings for chatbot functionality.

Here is the repo link for the complete code: [PneumonoBot - GitHub Link]

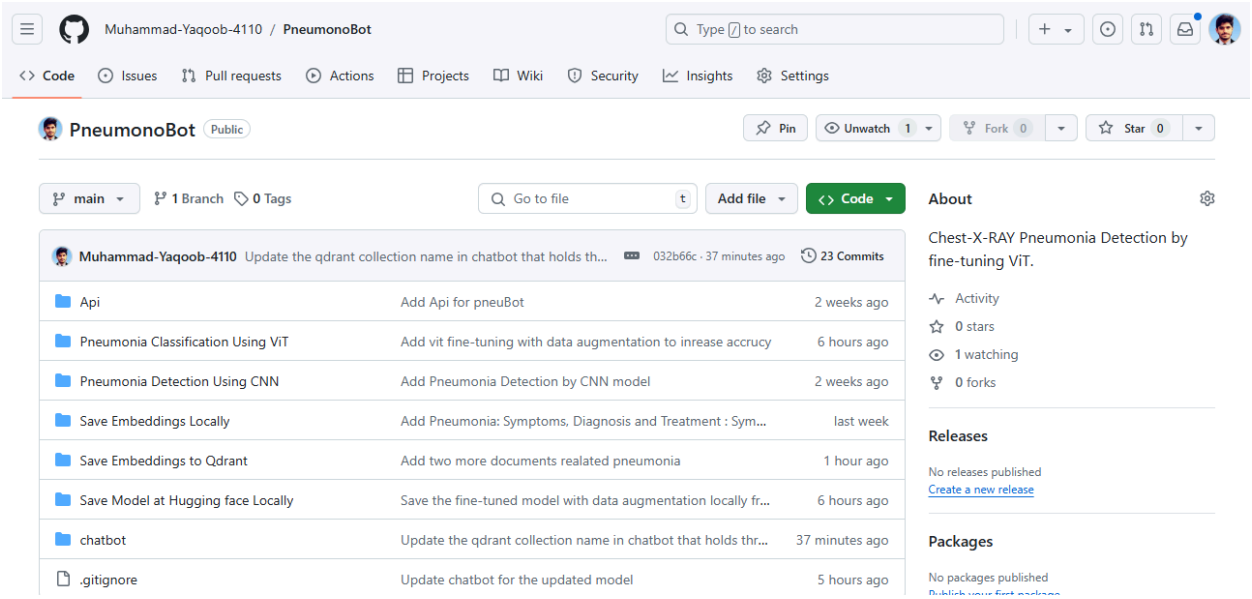

Figure 10. Tools Used GitHub

6.3. Qdrant

We have successfully uploaded all of the embeddings in vector form to Qdrant, a powerful vector database optimized for similarity search. To organize our data effectively, we created a collection named "pneumonoBot" within Qdrant. This collection stores the embeddings and metadata of documents related to pneumonia. By using Qdrant's upsert functionality, we continuously update and insert new documents into the collection, enabling fast and efficient

retrieval of relevant information when queried by the model. This setup ensures that our system remains scalable and capable of handling complex similarity searches for pneumonia-related content.

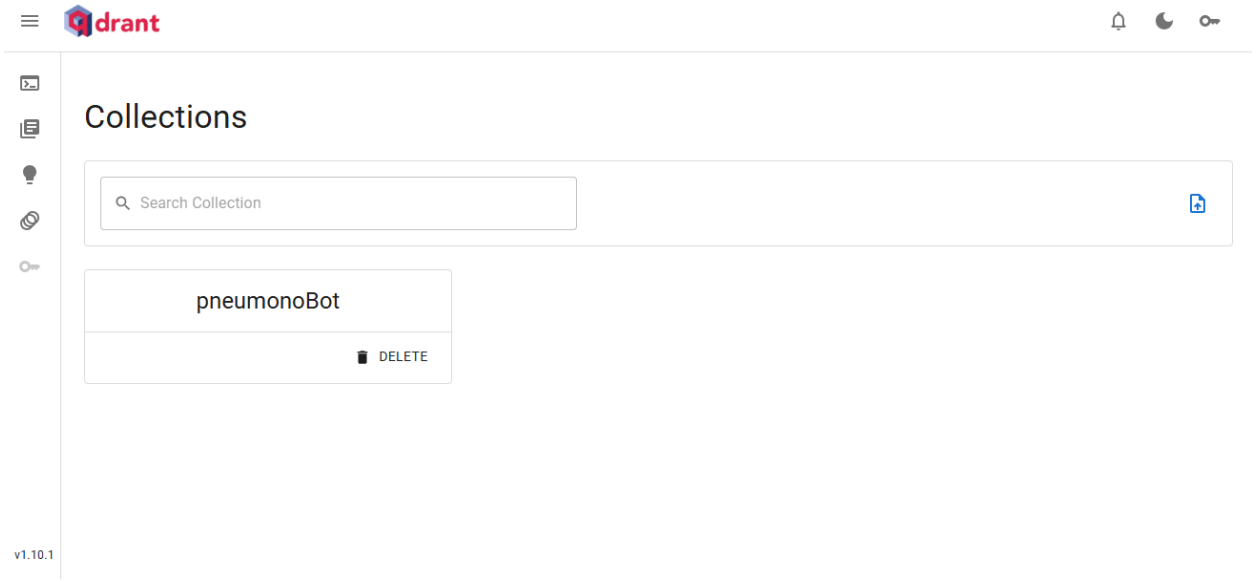

Figure 11. Tools Used - Qdrant

7. Run the pneumonoBot locally

7.1. Clone the repository

```
git clone https://github.com/Muhammad-Yaqoob-4110/PneumonoBot.git
```

7.2. Load the Modal from Hugging Face

To run the PneumonoBot locally, first, load the model from Hugging Face:

```
from transformers import ViTForImageClassification, ViTFeatureExtractor

# Load the model and feature extractor from Hugging Face
model_name = "M-Yaqoob/PneumonoBot"
model = ViTForImageClassification.from_pretrained(model_name)
feature_extractor = ViTFeatureExtractor.from_pretrained(model_name)

# Save the model and feature extractor locally
```

```
save_directory = "./vit_classification_pneumonobot"
model.save_pretrained(save_directory)
feature_extractor.save_pretrained(save_directory)
```

7.3. Install Dependencies

Navigate to the client directory and install the required dependencies:

```
cd chatbot
pip install -r requirements.txt
```

7.4. Configuration

Create a configuration file named config.json in the PneumonoBot/chatbot directory with the following content:

```
{
  "qdrant_url": "QDRANT_URL",
  "qdrant_api_key": "QDRANT_API_KEY",
  "nvidia_api_key": "NVIDIA_API_KEY",
  "lamini_api_key": "LAMINI_API_KEY",
  "gemini_api_key": "GEMINI_API_KEY"
}
```

- To get the nvidia api key, follow this Nvidia API Key
- To get the Qdrant URL land api key follow the official documentation Qdrant docs
- To get the Lamini Api follow this Lamini Api Key
- You can get your free Google Gemini Api key by following this: Get your Google Gemini Api Key

7.5. Running the Application

To run the gradio application, run:

```
gradio app.py
```

It will start the server, Feel free to try out PneumonoBot by uploading X-ray images and asking questions related to pneumonia through the chatbot.

8. Comparison Table

Aspect	CheXNet	Inception-v3	PneumonoBot
Architectue	121-layer DenseNet	Inception-v3 with Transfer Learning	Vision Transformer (ViT) + Fine-tuned LLaMA
Accuracy	76%	88%	91%

Table 3. Comparison table

9. Conclusion

In this project, we successfully developed an image classification model for pneumonia detection using chest X-ray images and integrated it with a user-friendly chatbot interface. Through a systematic approach involving data augmentation, preprocessing, and fine-tuning a Vision Transformer (ViT) model, we achieved an accuracy of 91%. This performance reflects the model's ability to generalize well on unseen data, making it a viable tool for assisting in medical diagnostics.

The chatbot integration further enhanced the system's usability by providing an interactive platform where users can upload images and receive instant feedback on their classifications. This, combined with the ability to ask questions and submit feedback, ensures that the system is not only technically effective but also engaging and accessible to non-technical users.

Overall, the combination of advanced machine learning techniques and a focus on user experience has resulted in a robust solution that could contribute to more efficient and accessible healthcare diagnostics. Future improvements could focus on further enhancing accuracy, expanding the chatbot's capabilities, and incorporating real-time feedback to continuously improve the model's performance.

10. References

All references in this report are accessible via hyperlinks for easy access.