Phase 3: Implementation of Project

Title: AI-Powered Early Disease Detection System

Objective

The objective of Phase 3 is to develop a functional, testable version of the AI-based early disease detection system by implementing selected strategies. This includes integrating data-driven models, enhancing the user interface, testing real-world usability, ensuring data privacy, and refining outputs based on feedback. The aim is to bridge the gap between prototype and a deployable product.

1. AI-Based Diagnostic Modeling

Overview

Train machine learning and deep learning models to predict early signs of diseases using medical images and lab reports.

Implementation

- Use CNN for imaging (e.g., chest X-rays) and ML models (e.g., XGBoost) for lab report analysis.
- Train models using publicly available datasets.
- Evaluate performance using metrics like accuracy, precision, and recall.

Outcome

A reliable backend system capable of analyzing patient data and providing early diagnostic insights.

2. User Interface and Dashboard Development

Overview

Design a simple, intuitive web-based interface for users (doctors, technicians) to interact with the system.

Implementation

- Develop the dashboard using HTML, CSS, and a framework like React or Flask.
- Integrate features like data upload, diagnostic results, and model explanations.
- Test the UI with sample users for usability and flow.

Outcome

A functioning front-end interface that allows smooth user interaction with the diagnostic system.

3. Data Privacy and Security Measures

Overview

Ensure all user data is securely handled, stored, and transmitted, maintaining medical data confidentiality.

Implementation

- Apply end-to-end encryption and secure login.
- Anonymize patient data.
- Align with standard data privacy regulations like HIPAA.

Outcome

A secure system that builds user trust by protecting sensitive medical information.

4. Model Explainability and Transparency

Overview

Make AI predictions interpretable to gain trust from healthcare professionals.

Implementation

- Use tools like Grad-CAM for image explanation and SHAP for lab data.
- Show visual cues and reasons behind each prediction on the dashboard.

Outcome

Improved user confidence and transparency in AI-driven decisions.

5. Lightweight Offline-Enabled Version

Overview

Build a version of the tool that works in low-resource or offline settings.

Implementation

- Optimize code and model sizes.
- Enable basic offline functionality using local storage and lightweight model loading.

Outcome

Wider accessibility, especially in rural or remote areas with limited internet access.

Challenges and Solutions

- **Challenge**: Model overfitting during training
- **Solution**: Use cross-validation and regularization techniques to prevent overfitting.
- **Challenge**: UI issues for non-tech users
- **Solution**: Conduct usability testing and simplify the design based on user feedback.
- **Challenge**: Integrating multiple data formats (images, text)
- **Solution**: Standardize input processing and create a unified backend data pipeline.
- **Challenge**: Maintaining accuracy with lightweight models
- **Solution**: Balance performance with model size using optimization methods and pruning.
- **Challenge**: Ensuring continuous data privacy compliance
- **Solution**: Conduct regular audits and integrate automated privacy checks into the system.

Outcomes of Phase 3

- A working system prototype integrating backend models and a front-end dashboard.
- Secure and interpretable AI predictions for disease detection.
- Successfully tested on synthetic and sample real-world data.
- Positive feedback from initial users regarding usability and output clarity.
- A scalable foundation ready for limited real-world deployment.

Next Steps for Phase 4

In Phase 4, the team will focus on:

1. Conduct broader real-world testing with healthcare professionals and institutions.
2. Optimize performance based on real-user feedback and expand disease coverage.
3. Integrate cloud deployment and database support for multi-user access.
4. Prepare documentation and training materials for stakeholders.
5. Plan regulatory certification steps and pre-launch publicity.

SCREENSHOTS OF CODE and PROGRESS

Data Preprocessing (for medical images)

```python
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split

data_dir = 'chest_xray_dataset/'
img_size = 224

X = []
y = []

for label in ['NORMAL', 'PNEUMONIA']:
    path = os.path.join(data_dir, label)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
        resized = cv2.resize(img_array, (img_size, img_size))
        X.append(resized)
        y.append(0 if label == 'NORMAL' else 1)

X = np.array(X).reshape(-1, img_size, img_size, 1) / 255.0
y = np.array(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

CNN Model for Disease Classification

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(img_size, img_size, 1)),
    MaxPooling2D(2,2),

    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

Model Training

```python
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
```

Grad-CAM for Explainability

main.py

```python
import matplotlib.pyplot as plt
import tensorflow.keras.backend as K

def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output, model.output]
    )

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(predictions[0])
        loss = predictions[:, pred_index]

    grads = tape.gradient(loss, conv_outputs)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    conv_outputs = conv_outputs[0]

    heatmap = conv_outputs @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)
    heatmap = np.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()

# Example usage
heatmap = make_gradcam_heatmap(np.expand_dims(X_test[0], axis=0), model, 'conv2d_1')
plt.matshow(heatmap)
plt.show()
```

Basic Flask API for Model Deployment (Optional)

main.py

```python
from flask import Flask, request, jsonify
import numpy as np
from tensorflow.keras.models import load_model

app = Flask(__name__)
model = load_model('model.h5')

@app.route('/predict', methods=['POST'])
def predict():
    img = request.files['image']
    img_array = cv2.imdecode(np.fromstring(img.read(), np.uint8), cv2.IMREAD_GRAYSCALE)
    resized = cv2.resize(img_array, (224, 224)).reshape(-1, 224, 224, 1) / 255.0
    prediction = model.predict(resized)
    result = 'PNEUMONIA' if prediction[0][0] > 0.5 else 'NORMAL'
    return jsonify({'prediction': result})

if __name__ == '__main__':
    app.run(debug=True)
```