**🩺 Project: Medical Expert System with AI Fallback**

**✅ Project Content**

This project combines a **rule-based medical expert system** with an **AI fallback mechanism** using OpenAI’s GPT models. It provides basic medical advice and explanations based on user-described symptoms. If a match isn't found in the rule-based system, it uses a language model (GPT-4 or GPT-4o-mini) to generate advice dynamically.

Key components:

* **Rule-based inference engine** using a simple symptom-to-advice mapping.
* **Fallback to OpenAI GPT** for symptoms not found in the knowledge base.
* **Gradio web interface** for easy interaction.

**Project Code:**

pip install openai

pip install gradio

import gradio as gr

import openai

# Set your OpenAI API key here or via environment variable

openai.api\_key = "YOUR\_OPENAI\_API\_KEY"

class MedicalExpertSystem:

    def \_\_init\_\_(self):

        self.knowledge\_base = [

            (["fever"],

             "Take paracetamol and rest. Drink plenty of fluids. If high fever persists, consult a doctor.",

             "Fever is usually a sign of infection. Rest and medication help manage it."),

            (["headache"],

             "Rest in a quiet, dark room. Drink water. Avoid screen time.",

             "Headaches are often caused by stress or dehydration."),

            (["cold"],

             "Take vitamin C, stay warm, and rest. Steam inhalation may help.",

             "A cold can be managed with warmth, fluids, and rest."),

            (["cough"],

             "Drink warm water with honey. Avoid cold drinks. If severe, take cough syrup.",

             "Cough may be due to throat irritation or infection."),

            (["nausea"],

             "Eat light food. Drink ginger tea or electrolyte water.",

             "Nausea is often linked to digestive issues."),

            (["diarrhea"],

             "Stay hydrated with ORS. Eat a bland diet. Avoid dairy and spicy foods.",

             "Dehydration is the main risk with diarrhea."),

        ]

        self.working\_memory = []

    def infer(self, user\_input):

        self.working\_memory.clear()

        self.working\_memory.extend(user\_input.lower().split())

        for conditions, advice, explanation in self.knowledge\_base:

            if all(cond in self.working\_memory for cond in conditions):

                return advice, explanation

        return None, None  # No match found

def call\_openai\_api(user\_input):

    prompt = (

        f"You are a helpful medical assistant. The user has described symptoms as follows:\n"

        f"'{user\_input}'\n"

        f"Please provide medical advice and a brief explanation suitable for a general audience."

    )

    try:

        response = openai.ChatCompletion.create(

            model="gpt-4o-mini",  # or "gpt-4" / "gpt-4o" / whichever you have access to

            messages=[

                {"role": "system", "content": "You are a helpful medical expert."},

                {"role": "user", "content": prompt}

            ],

            max\_tokens=300,

            temperature=0.7,

        )

        text = response['choices'][0]['message']['content']

        # Try splitting into advice and explanation, or just return whole response

        if "Explanation:" in text:

            parts = text.split("Explanation:")

            advice = parts[0].strip()

            explanation = parts[1].strip()

        else:

            advice = text.strip()

            explanation = "Generated medical explanation by AI."

        return advice, explanation

    except Exception as e:

        return "Sorry, AI service is unavailable.", str(e)

# Instantiate system

system = MedicalExpertSystem()

def medical\_advice\_interface(user\_input):

    advice, explanation = system.infer(user\_input)

    if advice is None:

        # fallback to AI if no rule-based match

        advice, explanation = call\_openai\_api(user\_input)

    return advice, explanation

# Launch Gradio app

gr.Interface(

    fn=medical\_advice\_interface,

    inputs=gr.Textbox(lines=2, placeholder="Enter your symptoms, e.g., 'I have fever and cough'"),

    outputs=[

        gr.Textbox(label="🩺 Medical Advice"),

        gr.Textbox(label="📘 Explanation")

    ],

    title="🤖 Medical Expert System + AI",

    description="Enter your symptoms and get rule-based advice or AI-enhanced guidance."

).launch()

**🛠️ Key Technologies Used**

| **Technology** | **Purpose** |
| --- | --- |
| **Python** | Core programming language |
| **Gradio** | For building the web-based user interface |
| **OpenAI API** | To access GPT-based language models for AI-enhanced responses |
| **Rule-Based Logic** | A simple decision-making system for common symptoms |

**📄 Description**

The system operates in two phases:

1. **Rule-Based Medical System**:
   * Matches user symptoms to predefined rules in a small knowledge\_base.
   * If matched, it returns the corresponding advice and explanation.
2. **AI-Powered Fallback (using GPT-4/4o-mini)**:
   * If no match is found in the knowledge base, it sends the input to GPT via OpenAI API.
   * The model returns a contextual advice and explanation.

This hybrid approach combines the transparency and simplicity of rule-based systems with the flexibility and understanding of large language models.

**💬 Output Example**

**User Input:**

I have fever and headache

**Output:**

**🩺 Medical Advice**:

Take paracetamol and rest. Drink plenty of fluids. If high fever persists, consult a doctor.

**📘 Explanation**:

Fever is usually a sign of infection. Rest and medication help manage it.

**🔍 Further Research / Enhancements**

1. **Expand Knowledge Base**:
   * Add more symptoms and rules (e.g., chest pain, rash, sore throat).
   * Use more complex condition checking (e.g., combinations of symptoms).
2. **Natural Language Processing**:
   * Implement NLP tools (spaCy, NLTK) to better parse and interpret user symptoms.
3. **Confidence Scoring**:
   * Score rule-based matches based on keyword similarity or fuzzy matching.
4. **Patient History Integration**:
   * Let users store previous inputs for context-aware advice.
5. **Data Validation / Safety Nets**:
   * Add warnings: "Not a substitute for medical consultation."
6. **Multilingual Support**:
   * Translate input/output to other languages for broader accessibility.
7. **Mobile App Version**:
   * Build a mobile-friendly version using tools like React Native or Flutter.
8. **Voice Input / Output**:
   * Use speech recognition and text-to-speech for accessibility.

**Cats vs Dogs Image Classification**

**Project Description**

This project implements a binary image classification model to distinguish between images of cats and dogs. The model uses transfer learning with the pre-trained MobileNetV2 architecture on the TensorFlow Cats vs Dogs dataset.

**Key Technologies**

* Python
* TensorFlow 2.x
* TensorFlow Datasets
* Transfer Learning (MobileNetV2)
* Matplotlib for visualization

**Project Content**

* Dataset loading and preprocessing
* Transfer learning with MobileNetV2 (pretrained on ImageNet)
* Model training and evaluation
* Image prediction function with visualization

**Code Snippet**

import tensorflow as tf

import tensorflow\_datasets as tfds

import matplotlib.pyplot as plt

from io import BytesIO

import PIL.Image

import requests

IMG\_SIZE = 160

# Load dataset

(train\_ds, val\_ds), ds\_info = tfds.load(

'cats\_vs\_dogs',

split=['train[:80%]', 'train[80%:]'],

with\_info=True,

as\_supervised=True

)

# Preprocessing function

def format\_image(image, label):

image = tf.image.resize(image, (IMG\_SIZE, IMG\_SIZE))

image = image / 255.0

return image, label

train\_ds = train\_ds.map(format\_image).batch(32).prefetch(buffer\_size=tf.data.AUTOTUNE)

val\_ds = val\_ds.map(format\_image).batch(32).prefetch(buffer\_size=tf.data.AUTOTUNE)

# Load pretrained MobileNetV2 model without classifier head

base\_model = tf.keras.applications.MobileNetV2(input\_shape=(IMG\_SIZE, IMG\_SIZE, 3),

include\_top=False,

weights='imagenet')

base\_model.trainable = False # Freeze base model

# Build the model

model = tf.keras.Sequential([

base\_model,

tf.keras.layers.GlobalAveragePooling2D(),

tf.keras.layers.Dense(1, activation='sigmoid') # Binary classification output

])

# Compile the model

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

# Train the model

history = model.fit(train\_ds, validation\_data=val\_ds, epochs=3)

# Prediction function

def predict\_image(url):

img = PIL.Image.open(BytesIO(requests.get(url).content)).resize((IMG\_SIZE, IMG\_SIZE))

img\_array = tf.keras.preprocessing.image.img\_to\_array(img)

img\_array = tf.expand\_dims(img\_array, 0) / 255.0

prediction = model.predict(img\_array)[0][0]

label = "Dog" if prediction > 0.5 else "Cat"

plt.imshow(img)

plt.title(f"Prediction: {label} ({prediction:.2f})")

plt.axis('off')

plt.show()

# Example usage

predict\_image("https://upload.wikimedia.org/wikipedia/commons/3/3a/Cat03.jpg")

**Output**

* The model will train on 80% of the Cats vs Dogs dataset and validate on the remaining 20%.
* After training, the predict\_image function downloads an image from a URL, preprocesses it, and displays the image along with the predicted label (Cat or Dog) and the model’s confidence score.
* Example output:   
    
  *Prediction: Cat (0.95)*

**Further Research**

* **Fine-tuning:** Unfreeze part of the MobileNetV2 layers and retrain to improve accuracy.
* **Data augmentation:** Apply image transformations to increase dataset variety.
* **Other architectures:** Experiment with different pretrained models like ResNet, EfficientNet, or DenseNet.
* **Multi-class classification:** Extend the model to classify other animal species.
* **Deploy model:** Build a web or mobile app that uses the model for real-time predictions.

.

# 🎬 Project: IMDB Movie Review Sentiment Analysis Using LSTM

## 📁 Project Content

* Load the IMDB movie review dataset.
* Clean and preprocess text data (remove HTML, punctuation, stopwords).
* Convert text to sequences and pad them.
* Build and train an LSTM-based deep learning model for binary sentiment classification.
* Evaluate model performance on test data.
* Test the model on custom sample movie reviews.

## 📝 Complete Project Code

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import re

import nltk

nltk.download('stopwords')

nltk.download('punkt')

from nltk.corpus import stopwords

from bs4 import BeautifulSoup

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from keras.layers import Dropout

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Embedding, LSTM

import pandas.testing as tm

# Load dataset

movie\_reviews = pd.read\_csv("/content/IMDB Dataset.csv")

# Check shape and nulls

print(movie\_reviews.shape)

print(movie\_reviews.isnull().sum())

# Visualize sentiment distribution

sns.countplot(x='sentiment', data=movie\_reviews)

plt.show()

# Remove HTML tags

def strip\_html(text):

soup = BeautifulSoup(text, "html.parser")

return soup.get\_text()

movie\_reviews['review'] = movie\_reviews['review'].apply(strip\_html)

# Remove punctuations and single chars

def remove\_punctuations(text):

pattern = r'[^a-zA-Z0-9\s]'

text = re.sub(pattern, '', text)

text = re.sub(r"\s+[a-zA-Z]\s+", ' ', text) # single chars

text = re.sub(r'\s+', ' ', text) # multiple spaces

return text

movie\_reviews['review'] = movie\_reviews['review'].apply(remove\_punctuations)

# Customize stopwords list to keep negations

stopword\_list = stopwords.words('english')

updated\_stopword\_list = [w for w in stopword\_list if w != 'not' and not w.endswith("n't")]

# Remove stopwords (excluding negations)

from nltk.tokenize import word\_tokenize

stop\_words = set(updated\_stopword\_list)

def remove\_stopwords(text):

words = word\_tokenize(text)

filtered = [word for word in words if word.lower() not in stop\_words]

return ' '.join(filtered)

movie\_reviews['review'] = movie\_reviews['review'].apply(remove\_stopwords)

# Convert sentiment to binary labels

movie\_reviews['sentiment'] = movie\_reviews['sentiment'].apply(lambda x: 1 if x == "positive" else 0)

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(movie\_reviews['review'].values, movie\_reviews['sentiment'].values,

test\_size=0.20, random\_state=42)

# Tokenization and sequences

tokenizer = Tokenizer(num\_words=5000)

tokenizer.fit\_on\_texts(X\_train)

X\_train\_tok = tokenizer.texts\_to\_sequences(X\_train)

X\_test\_tok = tokenizer.texts\_to\_sequences(X\_test)

vocab\_size = len(tokenizer.word\_index) + 1

maxlen = 100

X\_train\_pad = pad\_sequences(X\_train\_tok, padding='post', maxlen=maxlen, truncating='post')

X\_test\_pad = pad\_sequences(X\_test\_tok, padding='post', maxlen=maxlen, truncating='post')

print('Number of unique words in the corpus:', vocab\_size)

# Build LSTM model

model = Sequential()

model.add(Embedding(input\_dim=vocab\_size, output\_dim=100, input\_length=maxlen))

model.add(LSTM(64, return\_sequences=True))

model.add(Dropout(0.3))

model.add(LSTM(32))

model.add(Dropout(0.3))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())

# Train model

history = model.fit(X\_train\_pad, y\_train, batch\_size=128, epochs=3, verbose=1, validation\_split=0.2)

# Evaluate model

y\_test = np.array(y\_test)

score, acc = model.evaluate(X\_test\_pad, y\_test, batch\_size=128)

print(f'Test score: {score}')

print(f'Test accuracy: {acc}')

print(f'Accuracy: {acc:.2%}')

# Test on new samples

test\_samples = [

"This movie is fantastic! I really like it because it is so good!",

"Good movie!",

"Maybe I like this movie.",

"Not to my taste, will skip and watch another movie",

"If you like action, then this movie might be good for you.",

"Bad movie!",

"Not a good movie!",

"This movie really sucks! Can I get my money back please?"

]

test\_samples\_tokens = tokenizer.texts\_to\_sequences(test\_samples)

test\_samples\_tokens\_pad = pad\_sequences(test\_samples\_tokens, maxlen=maxlen)

pred = model.predict(test\_samples\_tokens\_pad)

# Display predictions

for text, p in zip(test\_samples, pred):

sentiment = "Positive" if p[0] > 0.5 else "Negative"

print(f"Review: {text}\nPrediction Score: {p[0]:.4f} --> Sentiment: {sentiment}\n")

## 🔑 Key Technologies

* **Python** — main programming language
* **Pandas, NumPy** — data manipulation
* **Matplotlib, Seaborn** — data visualization
* **NLTK** — natural language processing (tokenization, stopwords)
* **BeautifulSoup** — HTML parsing and cleaning
* **TensorFlow / Keras** — deep learning framework for model building
* **Scikit-learn** — train-test splitting and evaluation

## 📄 Description

The project focuses on sentiment classification of movie reviews from the IMDB dataset by building an LSTM neural network:

* First, data is loaded and explored to ensure balance and no missing values.
* Reviews are cleaned by stripping HTML tags and removing punctuations.
* Stopwords are removed except negations (not, n't) to preserve sentiment meaning.
* Text is tokenized and sequences are padded to a uniform length.
* An LSTM model is constructed with two LSTM layers and dropout for regularization.
* The model is trained to predict positive or negative sentiment.
* Finally, the model is tested on unseen reviews and some custom sample sentences.

## 🎯 Expected Output

* Dataset shape: (50000, 2)
* Sentiment distribution plot showing roughly equal positive and negative counts.
* Model summary printed with layer info.
* Training progress showing loss and accuracy per epoch.
* Test accuracy ~85% (varies based on random seed and training)
* Sample predictions with sentiment scores and labels, e.g.:

Review: This movie is fantastic! I really like it because it is so good!

Prediction Score: 0.9567 --> Sentiment: Positive

Review: Not a good movie!

Prediction Score: 0.1034 --> Sentiment: Negative

Review: This movie really sucks! Can I get my money back please?

Prediction Score: 0.0412 --> Sentiment: Negative

## 🔍 Further Research & Improvements

* **Hyperparameter tuning:** Explore different LSTM units, embedding sizes, batch sizes, learning rates.
* **Pretrained embeddings:** Use GloVe, Word2Vec or FastText embeddings instead of training from scratch.
* **Bidirectional LSTM:** Improve context understanding by reading sequences forward and backward.
* **More epochs:** Longer training might increase accuracy.
* **Advanced preprocessing:** Lemmatization, handling slang, emojis, and contractions.
* **Other architectures:** Test CNNs, Transformer models, or ensemble models.
* **Explainability:** Use LIME or SHAP to interpret model predictions.
* **Deployment:** Build a simple web app using Flask, FastAPI, or Gradio for live predictions.
* **Multilingual support:** Adapt pipeline to other languages or multilingual reviews.