import os

import warnings

import google.generativeai as genai

from gtts import gTTS

from langdetect import detect

import speech\_recognition as sr

from langchain\_google\_genai import ChatGoogleGenerativeAI

from langchain.memory import ConversationBufferMemory

from langchain.prompts import PromptTemplate

from langchain\_core.runnables import RunnablePassthrough

import pygame

import io

import time

# Suppress Warnings

warnings.filterwarnings("ignore")

# API Key - Replace with your actual API key or use environment variable

API\_KEY = "AIzaSyAwnVLm-zIyR\_FCl9VFqdLXQ3Weu8N9y7A"

if not API\_KEY or API\_KEY != "AIzaSyAwnVLm-zIyR\_FCl9VFqdLXQ3Weu8N9y7A":

raise ValueError("Missing valid API Key! Set GOOGLE\_API\_KEY as an environment variable.")

# Configure Gemini API

genai.configure(api\_key=API\_KEY)

# Set up LangChain with Gemini

llm = ChatGoogleGenerativeAI(model="gemini-1.5-pro", google\_api\_key=API\_KEY)

# Define memory to retain chat history

memory = ConversationBufferMemory(memory\_key="chat\_history", return\_messages=True)

# Define a prompt template for Kairo, the universal language AI

prompt = PromptTemplate(

input\_variables=["chat\_history", "user\_input", "detected\_language"],

template="""You are Kairo, an advanced AI guide created to assist in communication across all languages and cultures.

Your key capabilities:

- You respond in EXACTLY the same language the user speaks to you

- You maintain perfect fluency and natural expression in ANY language

- You understand cultural context and nuances specific to each language

- You adapt your responses to be culturally appropriate

Your communication style:

- Formal yet warm

- Clear and concise

- Natural and conversational

- Include occasional cultural references or sayings appropriate to the language

The user's detected language code is: {detected\_language}

Chat history:

{chat\_history}

User: {user\_input}

Respond ONLY in the same language as the user input. Do not translate or include any language tags.

Kairo:"""

)

# Function to load memory history

def get\_chat\_history(\_):

return memory.load\_memory\_variables({}).get("chat\_history", "")

# Create the chatbot chain

def get\_response(user\_input, detected\_lang):

return chat\_chain.invoke({

"user\_input": user\_input,

"detected\_language": detected\_lang

})

# Initialize pygame mixer for audio playback

pygame.mixer.init()

# Improved Speech Recognition with retry and language options

def listen\_for\_speech(language=""):

recognizer = sr.Recognizer()

# Set recognition parameters for better accuracy

recognizer.energy\_threshold = 300 # Increase sensitivity

recognizer.dynamic\_energy\_threshold = True

recognizer.pause\_threshold = 0.8 # Shorter pause to detect end of speech

max\_attempts = 3

attempt = 0

while attempt < max\_attempts:

with sr.Microphone() as source:

print(f"Listening (Attempt {attempt+1}/{max\_attempts})...")

recognizer.adjust\_for\_ambient\_noise(source, duration=0.5)

try:

audio = recognizer.listen(source, timeout=10, phrase\_time\_limit=15)

print("Recognizing...")

# Try with language preference if specified

if language and language != "auto":

try:

text = recognizer.recognize\_google(audio, language=language)

print(f"You said: {text}")

return text

except:

pass # Fall back to auto-detection if specific language fails

# Auto language detection

text = recognizer.recognize\_google(audio)

print(f"You said: {text}")

return text

except sr.WaitTimeoutError:

print("No speech detected. Please try again.")

except sr.UnknownValueError:

print("Could not understand audio. Please speak more clearly.")

except sr.RequestError as e:

print(f"Recognition service error: {e}")

break # Exit on service error

attempt += 1

print("Voice recognition unsuccessful. Please try again.")

return ""

# Enhanced language detection

def detect\_language(text):

try:

# Robust detection for short phrases

if len(text.split()) <= 3:

# For very short inputs, we need more context

# This is a simple approach - you might want to use a more sophisticated method

words = text.lower().split()

# Simple keyword checking for common words in different languages

# Expand this dictionary with more languages and common words as needed

language\_markers = {

'en': ['hi', 'hello', 'the', 'is', 'and', 'what', 'who', 'how'],

'es': ['hola', 'como', 'qué', 'el', 'la', 'es', 'y', 'por'],

'fr': ['bonjour', 'salut', 'le', 'la', 'est', 'et', 'qui', 'comment'],

'de': ['hallo', 'guten', 'der', 'die', 'das', 'ist', 'und', 'wie'],

'hi': ['नमस्ते', 'कैसे', 'हो', 'क्या', 'है', 'और', 'कौन', 'कैसा'],

'ta': ['வணக்கம்', 'எப்படி', 'நான்', 'நீங்கள்', 'என்ன', 'மற்றும்'],

'kn': ['ನಮಸ್ಕಾರ', 'ಹೇಗೆ', 'ನಾನು', 'ನೀವು', 'ಏನು', 'ಮತ್ತು'],

'te': ['నమస్కారం', 'ఎలా', 'నేను', 'మీరు', 'ఏమి', 'మరియు']

}

for word in words:

for lang, markers in language\_markers.items():

if word in markers:

return lang

# Standard language detection for longer phrases

return detect(text)

except:

return 'en' # Default to English if detection fails

# Improved Text-to-Speech with enhanced pronunciation

def text\_to\_speech(text, lang\_code='en'):

if not text.strip():

return

try:

# Map language codes to more specific locale codes for better pronunciation

tld\_map = {

'en': 'com', # US English

'en-uk': 'co.uk', # UK English

'en-au': 'com.au', # Australian English

'es': 'es', # Spanish

'fr': 'fr', # French

'de': 'de', # German

'hi': 'co.in', # Hindi

'ta': 'co.in', # Tamil

'kn': 'co.in', # Kannada

'te': 'co.in', # Telugu

'ja': 'co.jp', # Japanese

'ko': 'co.kr', # Korean

'zh': 'com.hk', # Chinese

'ru': 'ru', # Russian

'ar': 'ae', # Arabic

'pt': 'com.br', # Portuguese

'it': 'it', # Italian

}

# Get the TLD based on language or default to .com

tld = tld\_map.get(lang\_code, 'com')

# Create gTTS object with appropriate settings for better pronunciation

tts = gTTS(text=text, lang=lang\_code, tld=tld, slow=False)

# Save to in-memory file

fp = io.BytesIO()

tts.write\_to\_fp(fp)

fp.seek(0)

# Play directly

pygame.mixer.music.load(fp)

pygame.mixer.music.play()

# Wait for playback to finish

while pygame.mixer.music.get\_busy():

pygame.time.Clock().tick(10)

except Exception as e:

print(f"Text-to-Speech Error: {e}")

print(f"Response text: {text}")

# Initialize the chat chain

chat\_chain = (

RunnablePassthrough.assign(chat\_history=get\_chat\_history)

| prompt

| llm

)

# Chatbot Function with improved voice interaction

def chatbot():

# Initial greeting in English

welcome\_message = "Greetings, I am Kairo, your multilingual AI guide. I can understand and respond in any language you speak to me. How may I assist you today?"

print(f"Kairo: {welcome\_message}")

text\_to\_speech(welcome\_message, 'en')

# Store last detected language for consistency

current\_language = 'en'

while True:

# Use last detected language to improve recognition

user\_input = listen\_for\_speech(current\_language).strip()

if not user\_input:

continue

# Detect language from user input

detected\_lang = detect\_language(user\_input)

current\_language = detected\_lang # Update current language

# Check for exit commands in multiple languages

exit\_phrases = [

"exit", "quit", "bye", "goodbye", "stop", # English

"salir", "adiós", "chao", "hasta luego", # Spanish

"sortir", "au revoir", "adieu", # French

"beenden", "tschüss", "auf wiedersehen", # German

"अलविदा", "बाय", "निकास", # Hindi

"வெளியேறு", "பிரியாவிடை", "வணக்கம்", # Tamil

"ನಿರ್ಗಮಿಸು", "ವಿದಾಯ", "ಹೋಗ್ತಿನಿ", # Kannada

"నిష్క్రమించు", "వీడ్కోలు", "బై" # Telugu

]

if any(exit\_word in user\_input.lower() for exit\_word in exit\_phrases):

# Create a culturally appropriate farewell in the detected language

response = get\_response(f"Say a warm and culturally appropriate farewell in this language", detected\_lang)

farewell = response.content

print(f"Kairo: {farewell}")

text\_to\_speech(farewell, detected\_lang)

break

# Get response in detected language

response = get\_response(user\_input, detected\_lang)

kairo\_response = response.content

# Print and speak response

print(f"Kairo: {kairo\_response}")

text\_to\_speech(kairo\_response, detected\_lang)

# Save to memory

memory.save\_context({"input": user\_input}, {"output": kairo\_response})

# Run the chatbot when the script is executed directly

if \_name\_ == "\_main\_":

chatbot()