"""

Real-Time Speech-to-Text Transcription System for Healthcare with Noise Robustness

"""

import whisper

import sounddevice as sd

import numpy as np

import queue

import tempfile

import soundfile as sf

import threading

import time

import os

import re

# Load Whisper model

model = whisper.load\_model("base")

# Audio configuration

samplerate = 16000

channels = 1

block\_duration = 5 # seconds

blocksize = samplerate \* block\_duration

q = queue.Queue()

def audio\_callback(indata, frames, time, status):

if status:

print(f"Audio Status: {status}", flush=True)

q.put(indata.copy())

def record\_audio():

"""Captures microphone input."""

try:

with sd.InputStream(samplerate=samplerate, channels=channels, dtype="float32", callback=audio\_callback):

print("[INFO] Recording started. Speak now...")

while True:

time.sleep(0.1)

except Exception as e:

print(f"[ERROR] Audio recording error: {e}")

def normalize\_medical\_terms(text):

"""Replace misrecognized terms with standard medical vocabulary."""

medical\_corrections = {

"bp": "blood pressure",

"sugar level": "glucose level",

"ekg": "ECG",

"covid": "COVID-19",

"diabetes type 1": "Type 1 Diabetes",

"diabetes type 2": "Type 2 Diabetes"

}

for wrong, correct in medical\_corrections.items():

pattern = re.compile(rf"\b{wrong}\b", re.IGNORECASE)

text = pattern.sub(correct, text)

return text

def log\_transcription(text, file\_path="transcription\_log.txt"):

"""Log transcription with a timestamp."""

timestamp = time.strftime("%Y-%m-%d %H:%M:%S")

with open(file\_path, "a") as f:

f.write(f"[{timestamp}] {text}\n")

def detect\_keywords(text):

"""Detect critical healthcare terms."""

keywords = ["emergency", "allergic reaction", "heart rate", "diabetic", "COVID-19", "blood pressure"]

found = [kw for kw in keywords if kw.lower() in text.lower()]

return found

def transcribe\_audio():

"""Transcribes audio chunks and processes transcription."""

while True:

if not q.empty():

try:

chunk = q.get()

chunk = np.squeeze(chunk)

audio\_int16 = (chunk \* 32767).astype(np.int16)

with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as f:

sf.write(f.name, audio\_int16, samplerate)

result = model.transcribe(f.name)

os.remove(f.name)

raw\_text = result["text"]

medical\_text = normalize\_medical\_terms(raw\_text.strip())

print(f"Transcription:\n{medical\_text}\n{'-'\*60}")

log\_transcription(medical\_text)

alerts = detect\_keywords(medical\_text)

if alerts:

print(f"[ALERT] Detected medical terms: {', '.join(alerts)}")

except Exception as e:

print(f"[ERROR] Transcription failed: {e}")

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

threading.Thread(target=record\_audio, daemon=True).start()

transcribe\_audio()