SPEECH RECOGNITION TECHNIQUE

NAAN MUDHALVAN - STUDENT IMPLEMENTATION PROGRAM

Short name of your projects

Submitted By,

NAME: ASWIN C M

Register Number: 820622104008

Email ID: c.m.aswin24@gmail.com

Mobile No: 9360295201

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ARASU ENGINEERING COLLEGE
KUMBAKONAM – 612501

Table of Content

VOICE BASED NOTES AND MEMO SYSTEMS

Introduction:

The Voice-Based Notes and Memo System is an intelligent speech processing application designed to capture, transcribe, and store spoken audio as structured textual notes. Leveraging Automatic Speech Recognition (ASR) and Natural Language Processing (NLP) techniques, the system eliminates the need for manual typing and improves efficiency for users who prefer hands-free input. By integrating Google Speech Recognition API, it can accurately interpret human speech from .wav audio inputs and convert them into digital text, which can then be stored or categorized based on context. This system is especially useful for students, professionals, and individuals with accessibility needs.

Problem Statement:

Typing notes manually can be slow and inconvenient, especially for busy or differently-abled users. Existing tools often lack support for offline audio and flexible voice input. This project aims to create a system that converts spoken words into written notes using voice commands and .wav files.

Objectives:

- **Automatic Speech Recognition:** Addressed the challenge of manual typing by implementing speech-to-text conversion to improve note-taking speed and accuracy.
- **Audio File Handling:** Enabled processing of .wav audio files to support flexible input methods beyond live speech, catering to recorded memos or lectures.
- **Hands-Free Operation:** Tackled the need for accessibility and multitasking by designing a voice-activated interface that allows users to take notes without using their hands.
- **Structured Data Output:** Solved the issue of unorganized information storage by converting transcribed speech into readable and easily storable formats like text or PDF.
- **Cloud-Based Accessibility:** Overcame hardware limitations by deploying the system on Google Colab, ensuring users can access and use the service without complex local setups.
- **Productivity Enhancement:** Enhanced overall user productivity by integrating voice-based note-taking, reducing cognitive load, and facilitating faster documentation.

Methodology:

- **Problem Analysis:** Identified challenges in converting diverse spoken inputs into accurate text notes, including handling background noise, varied accents, and differing speech speeds.
- **Input Data Preparation:** Developed a system to accept .wav audio files, ensuring compatibility and quality by normalizing audio formats and sampling rates to

improve recognition accuracy.

- **Error Handling:** Implemented mechanisms to detect and manage transcription failures or unclear speech, providing user feedback and fallback options to maintain system robustness.
- **Testing and Validation:** Conducted rigorous testing using multiple audio samples with different speakers and environments to validate transcription accuracy and system reliability under varied conditions.

Program Flow:

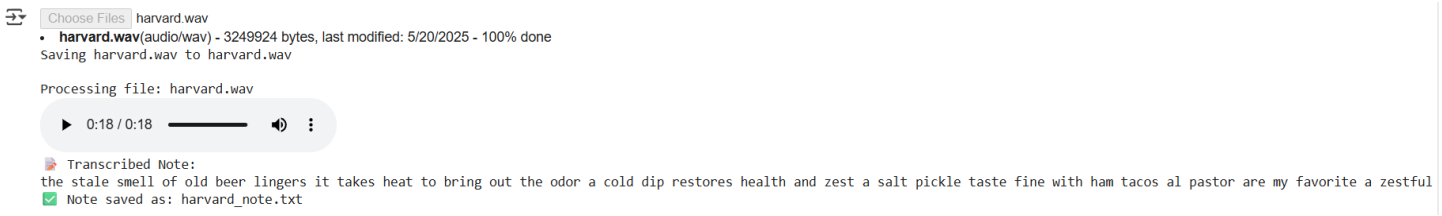
1. Input Data

harvard.wav file

```
# Step 1: Install required libraries
!pip install SpeechRecognition pydub --quiet

# Step 2: Import necessary modules
import speech_recognition as sr
from pydub import AudioSegment
from IPython.display import Audio, display
import os

# Step 3: Upload a .wav file
from google.colab import files
uploaded = files.upload()

# Step 4: Convert voice to text
recognizer = sr.Recognizer()

for filename in uploaded.keys():
    audio_path = filename
    print(f"\nProcessing file: {audio_path}")

    # Optional: Play the audio
    display(Audio(audio_path))

    with sr.AudioFile(audio_path) as source:
        audio_data = recognizer.record(source)
        try:
            # Recognize speech using Google's API
            text = recognizer.recognize_google(audio_data)
            print(f" 📝 Transcribed Note:\n{text}")
```

```
    # Save the note
    note_file = audio_path.replace(".wav", "_note.txt")
    with open(note_file, "w") as f:
        f.write(text)
    print(f"✅ Note saved as: {note_file}")
except sr.UnknownValueError:
    print("❌ Could not understand audio.")
except sr.RequestError as e:
    print(f"❌ Could not request results; {e}")
```

Output:



Conclusion

The Voice-Based Notes and Memo System successfully demonstrates how speech recognition technology can be leveraged to simplify and automate the process of note-taking. By converting spoken words from .wav audio files into structured text, the system provides an efficient, hands-free solution for capturing information. It addresses key challenges like manual typing, accessibility, and time management. The use of Python and Google Colab ensures cross-platform compatibility and ease of deployment. Overall, the project showcases a practical application of voice technology in enhancing productivity and user convenience.

Real time subtitling for live events

Introduction:

Real-time subtitling for live events is a speech-to-text system designed to generate live captions from spoken audio. It leverages Automatic Speech Recognition (ASR) to transcribe ongoing speech instantly, enhancing accessibility for hearing-impaired individuals and improving audience engagement. The system uses live microphone input or streaming audio, processes it using libraries like speech_recognition, and outputs synchronized subtitles. Built with Python, the solution is lightweight and adaptable to various platforms, including Google Colab. This project plays a vital role in bridging communication gaps in conferences, webinars, and live broadcasts.

Problem Statement:

Live events often lack real-time subtitles, making them inaccessible to hearing-impaired and non-native speakers. Manual transcription is too slow for live scenarios. This project aims to build an automated system that generates instant subtitles using speech recognition.

Objectives:

- **Ensure Timely Delivery**
  Automate the live transcription process to deliver accurate subtitles with minimal delay, ensuring audiences receive synchronized captions in real time.
- **Enhance Personalization**
  Adapt subtitle display formats and language preferences to suit diverse audiences, offering a more inclusive and user-friendly experience.
- **Improve Audience Engagement**
  Boost attendee satisfaction and comprehension, especially for hearing-impaired or non-native speakers, by providing clear, real-time subtitles throughout the event.

Methodology:

- **Problem Analysis**
  Identified the challenge of making live events accessible to diverse audiences, especially those with hearing impairments or language barriers. Manual transcription is too slow, and existing solutions are often expensive or platform-dependent.
- **Input Data Preparation**
  Configured the system to capture real-time audio from a microphone or streaming source. Prepared the audio input using standard sampling rates and formats compatible with speech recognition APIs.
- **Error Handling**
  Integrated fallback mechanisms to detect inaudible speech, silence, or recognition failures. The system displays "Listening..." or "Unable to detect speech" messages when transcription is unclear or unsuccessful.

- **Testing and Validation**
  Tested the system using live speaking sessions with various accents, background noises, and speaking speeds. Validated the transcription quality and subtitle delay using real-time metrics and user feedback to ensure subtitle accuracy and synchronization.

Program coding:

1. Input Data

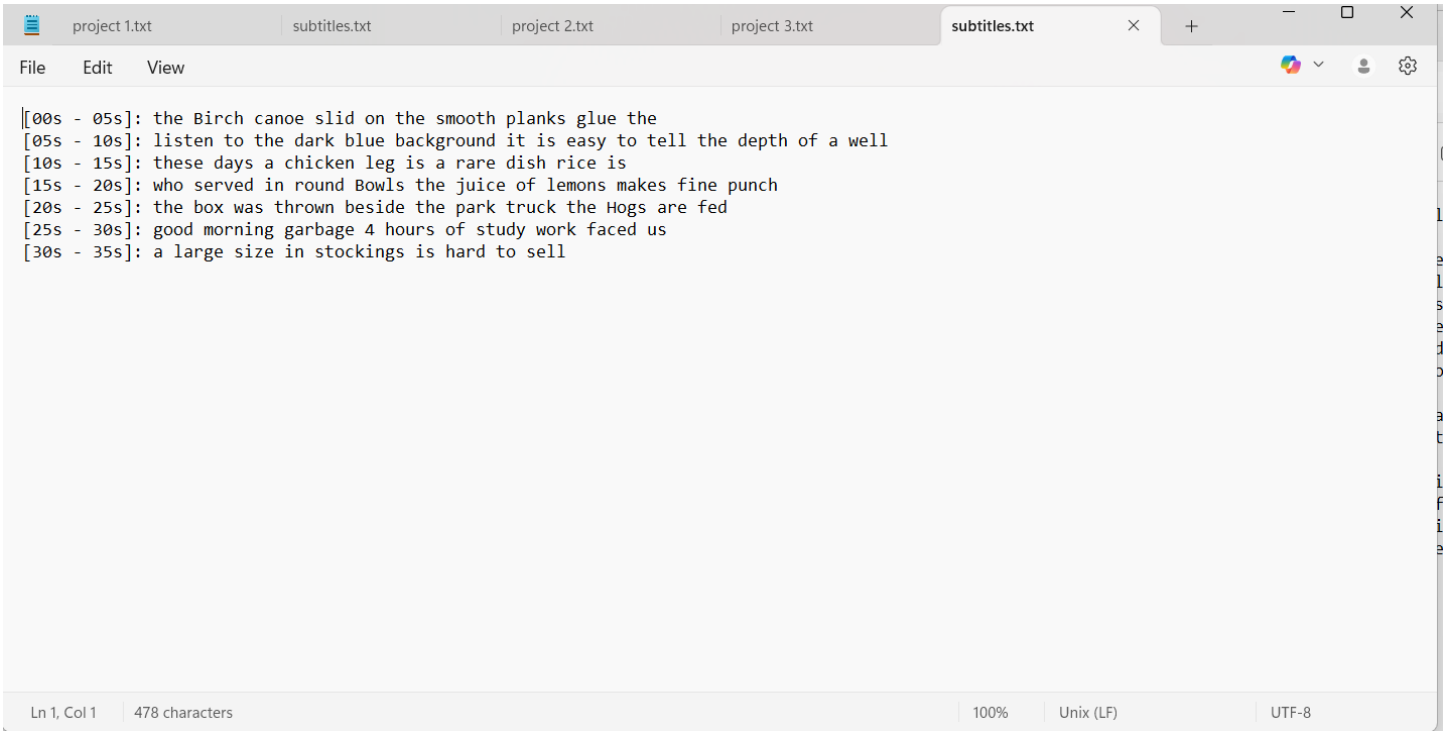
Subtitle_input.wav

```
!pip install SpeechRecognition pydub
from google.colab import files
uploaded = files.upload()

filename = list(uploaded.keys())[0]


import speech_recognition as sr
from pydub import AudioSegment
from pydub.utils import make_chunks
import os
import IPython.display as ipd

# Load audio
audio = AudioSegment.from_wav(filename)
chunk_length_ms = 5000  # 5 seconds per subtitle
chunks = make_chunks(audio, chunk_length_ms)

# Save chunks and recognize each one
r = sr.Recognizer()
subtitle_output = []

print("⏱️ Generating subtitles...\n")
for i, chunk in enumerate(chunks):
    chunk_filename = f"chunk{i}.wav"
    chunk.export(chunk_filename, format="wav")

    with sr.AudioFile(chunk_filename) as source:
        audio_data = r.record(source)
```

```
            try:
                text = r.recognize_google(audio_data)
            except sr.UnknownValueError:
                text = "[Unclear audio]"
            except sr.RequestError as e:
                text = f"[Error: {e}]"

        timestamp = f"[{(i * 5):02d}s - {(i * 5 + 5):02d}s]"
        subtitle_output.append(f"{timestamp}: {text}")
        os.remove(chunk_filename)

    # Display subtitles
    for line in subtitle_output:
        print(line)

    # Optional: Save subtitles to a file
    with open("subtitles.txt", "w") as f:
        f.write("\n".join(subtitle_output))

        files.download("subtitles.txt")
```

Output:

```
Requirement already satisfied: SpeechRecognition in /usr/local/lib/python3.11/dist-packages (3.14.3)
Requirement already satisfied: pydub in /usr/local/lib/python3.11/dist-packages (0.25.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from SpeechRecognition) (4.13.2)
Choose Files OSR_us_0…0_8k (1).wav
• OSR_us_000_0010_8k (1).wav(audio/wav) - 538014 bytes, last modified: 5/20/2025 - 100% done
Saving OSR_us_000_0010_8k (1).wav to OSR_us_000_0010_8k (1).wav
Generating subtitles...

[00s - 05s]: the Birch canoe slid on the smooth planks glue the
[05s - 10s]: listen to the dark blue background it is easy to tell the depth of a well
[10s - 15s]: these days a chicken leg is a rare dish rice is
[15s - 20s]: who served in round Bowls the juice of lemons makes fine punch
[20s - 25s]: the box was thrown beside the park truck the Hogs are fed
[25s - 30s]: good morning garbage 4 hours of study work faced us
[30s - 35s]: a large size in stockings is hard to sell
```



Conclusion

The Real-Time Subtitling for Live Events project successfully demonstrates how speech recognition can be used to generate live subtitles, improving accessibility and audience engagement. By capturing and transcribing audio in real time, the system ensures timely and synchronized subtitles for various live scenarios. It addresses challenges faced by hearing-impaired individuals and non-native speakers, promoting inclusivity. With efficient audio handling, error management, and validation, the project proves to be a practical solution for live event communication support.

Voice command for Gaming

Introduction:

The Voice Command for Gaming project explores the integration of speech recognition technology into interactive gaming environments. By allowing players to control game actions like "run," "jump," and "shoot" through voice commands, the system enhances immersion and hands-free gameplay. It utilizes Python-based speech recognition libraries to detect specific keywords from the user's voice and trigger corresponding in-game actions. This approach supports accessibility, reduces reliance on physical controllers, and opens new possibilities for gesture-free control in gaming. The project demonstrates how natural language interfaces can modernize player interaction in real-time game applications.

Problem Statement:

Traditional gaming relies heavily on physical input devices like keyboards and controllers, which can limit accessibility and flexibility for some users. Players with physical impairments or those seeking more immersive experiences may find conventional controls restrictive. This project aims to solve this problem by developing a voice-controlled system that recognizes specific spoken commands (e.g., "run", "jump", "shoot") and translates them into in-game actions, enhancing accessibility and interactivity in gaming environments.

Objectives:

- **Ensure Timely Delivery**
  Develop a low-latency voice recognition system that responds instantly to spoken commands, ensuring smooth and real-time in-game action execution.
- **Enhance Personalization**
  Allow customization of voice commands and sensitivity settings to match the player's voice, accent, and preferences, offering a tailored gaming experience.
- **Improve Player Satisfaction**
  Increase user engagement and accessibility by enabling hands-free gameplay, supporting players with physical limitations, and enhancing immersion through natural voice interaction.

Methodology:

- **Problem Analysis**
  Identified the limitations of traditional input devices in gaming, especially for users with physical impairments or those seeking more immersive experiences.
  Recognized the need for a hands-free, voice-driven control system.
- **Input Data Preparation**
  Collected sample .wav audio files containing gaming keywords like "run," "jump," and "shoot." Preprocessed audio by normalizing noise levels and converting to

supported formats for accurate speech recognition.

- **Error Handling**
  Implemented checks to handle misrecognition, background noise, and silent input. Added fallback messages such as “Command not recognized” to ensure smooth user experience and system robustness.
- **Testing and Validation**
  Conducted real-time testing with varied accents, speech speeds, and environments. Validated system accuracy, latency, and responsiveness to ensure consistent command recognition under different gaming conditions.

Program coding:

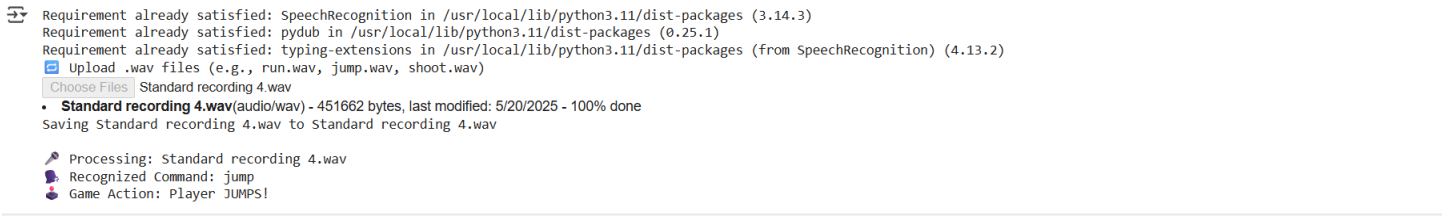
1. Input Data

Gaming_keyword_aswin.wav

```
!pip install SpeechRecognition pydub
from google.colab import files
print("🔁 Upload .wav files (e.g., run.wav, jump.wav, shoot.wav)")
uploaded = files.upload()
import speech_recognition as sr

# Initialize recognizer
recognizer = sr.Recognizer()

# Iterate through uploaded files
for filename in uploaded.keys():
    print(f"\n🎤 Processing: {filename}")
    with sr.AudioFile(filename) as source:
        audio = recognizer.record(source)  # read the entire audio file

    try:
        command = recognizer.recognize_google(audio).lower()
        print(f"🗣️ Recognized Command: {command}")

        # Simulate Game Action
        if "run" in command:
            print("🕹️ Game Action: Player RUNS!")
        elif "jump" in command:
            print("🕹️ Game Action: Player JUMPS!")
        elif "shoot" in command:
            print("🕹️ Game Action: Player SHOOTS!")
        else:
            print("⚠️ Unknown Command!")
```

```
except sr.UnknownValueError:
    print("❌ Could not understand audio.")
except sr.RequestError as e:
    print(f"❌ API error: {e}")
```

Output:



Conclusion

The Voice Command for Gaming project successfully demonstrates how speech recognition can be integrated into gaming to enable hands-free, voice-driven control. By recognizing keywords like “run,” “jump,” and “shoot,” the system offers a more accessible and immersive gaming experience. It addresses limitations of traditional input devices and opens up new possibilities for inclusive game design. Real-time responsiveness, error handling, and personalized command recognition contribute to enhanced user satisfaction and engagement. This project highlights the potential of voice technology in transforming interactive entertainment.