PROJECT :

Building an End-to-End Speech Recognition Pipeline: Signal Processing, Acoustic Modeling, and Performance Evaluation

Problem Statement:

Speech recognition systems are critical for applications like virtual assistants, transcription services, and voice-controlled devices. However, raw audio signals often contain background noise, making accurate speech recognition

challenging. Additionally, extracting meaningful features from audio signals and building robust acoustic models require advanced signal processing and machine learning techniques.

The goal of this project is to design and implement a complete speech

recognition pipeline that includes noise reduction, feature extraction (e.g., MFCCs), voice activity detection (VAD), and acoustic modeling using Hidden Markov Models (HMMs) and deep learning techniques. The system will be evaluated for accuracy and performance.

Business Use Cases:

1. Call Center Automation

a. Automate transcription and sentiment analysis of customer calls.

2. Accessibility Tools

a. Develop tools for individuals with hearing impairments by

converting spoken content into readable text.

3. Voice Assistants

a. Enhance the accuracy of voice assistants in understanding user

commands across different accents and environments.

4. Meeting Transcription

a. Provide real-time transcription services for business meetings,

enabling better record-keeping and collaboration.

5. Voice-Controlled Devices

a. Enhance the reliability of voice commands in IoT devices.

Approach:

Data Collection and Cleaning

- Collect a speech corpus dataset containing clean and noisy audio samples.
- Preprocess the data by normalizing volume levels, removing silence, and segmenting audio into frames.
- Apply noise reduction techniques (e.g., spectral subtraction, Wiener filtering).

Data Analysis

- Extract features such as MFCCs, pitch, and energy from the preprocessed audio signals.
- Perform Voice Activity Detection (VAD) to identify speech segments and discard non-speech portions.
- Visualization
- Visualize spectrograms of raw and processed audio signals.
- Plot MFCCs and other extracted features to understand their distribution.
- Compare noise-reduced signals with original signals using waveforms. Advanced Analytics
- Train a Hidden Markov Model (HMM) for acoustic modeling using the extracted features.
- Implement a simple deep learning model (e.g., CNN or RNN) for comparison.
- Evaluate the performance of both models using metrics like Word Error Rate (WER) and accuracy.

Power BI Integration

Use Power BI to create dashboards showing:

- Accuracy metrics of different models.
- Comparison of noise reduction techniques.
- Feature distributions and correlations

Visualization

- Waveform Plots : Raw vs. noise-reduced audio signals.
- Spectrograms : Time-frequency representation of audio.
- Feature Plots : MFCCs, pitch, and energy distributions.
- Power BI Dashboard : Interactive visualizations for business stakeholders. Exploratory Data Analysis (EDA)
- Analyze the distribution of audio durations and sampling rates.
- Identify common types of noise in the dataset.
- Explore the correlation between extracted features (e.g., MFCCs and pitch).
- Evaluate the effectiveness of VAD in isolating speech segments.
- Compare the performance of different noise reduction techniques.

Speech-to-Text Python Project:

1. Requirements

- Python 3.11.1
- speechrecognition library
- pyaudio library

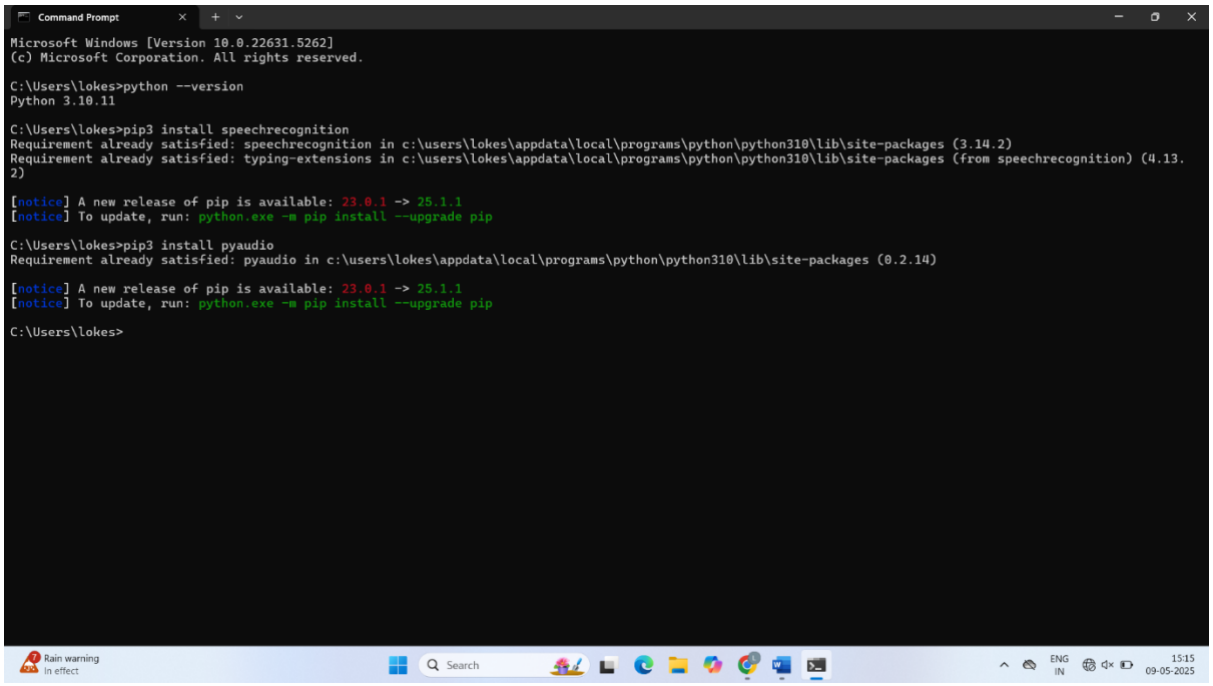
2. Installation Steps

- Install SpeechRecognition:

  ✓ pip3 install speechrecognition

- Install PyAudio:

  ✓ pip3 install pyaudio



Python Code :

Speech Recogonition.py

```
import speech_recognition as sr
import os
import time
import threading
import tkinter as tk
from tkinter import scrolledtext, Button, Label, filedialog, messagebox, ttk
import pyaudio
import sys

class SpeechTranscriber:
    def __init__(self):
        self.recognizer = sr.Recognizer()
        self.is_listening = False
        self.transcript_file = "transcript.txt"
        self.transcript_text = ""
        self.mic_list = []
        self.selected_mic_index = None
        self.setup_ui()

    def detect_microphones(self):
        try:
            p = pyaudio.PyAudio()
            self.mic_list = []
            info_text = "Available microphones:\n"
            for i in range(p.get_device_count()):
                dev_info = p.get_device_info_by_index(i)
                if dev_info['maxInputChannels'] > 0:
```

```
                name = dev_info['name']
                self.mic_list.append((i, name))
                info_text += f"{i}: {name}\n"
            p.terminate()
            if not self.mic_list:
                messagebox.showerror("Error", "No microphones detected in your system!")
                self.text_area.insert(tk.END, "ERROR: No microphones detected.\n")
                return False
            self.text_area.insert(tk.END, info_text)
            return True
        except Exception as e:
            messagebox.showerror("Error", f"Error detecting microphones: {str(e)}")
            self.text_area.insert(tk.END, f"ERROR: {str(e)}\n")
            return False

    def setup_ui(self):
        self.root = tk.Tk()
        self.root.title("Speech Recognition Transcriber")
        self.root.geometry("700x600")
        self.root.configure(bg="#f0f0f0")

        Label(self.root, text="Speech Recognition Transcriber", font=("Arial", 16, "bold"),
bg="#f0f0f0").pack(pady=10)

        mic_frame = tk.Frame(self.root, bg="#f0f0f0")
        mic_frame.pack(fill=tk.X, padx=10, pady=5)

        Label(mic_frame, text="Select Microphone:", bg="#f0f0f0").pack(side=tk.LEFT)
        self.mic_var = tk.StringVar()
```

```
        self.mic_dropdown = ttk.Combobox(mic_frame, textvariable=self.mic_var,
state="readonly", width=40)
        self.mic_dropdown.pack(side=tk.LEFT, padx=5)
        Button(mic_frame, text="↻", command=self.refresh_microphones, bg="#4db6ac",
fg="white").pack(side=tk.LEFT, padx=5)
        Button(mic_frame, text="Test Mic", command=self.test_microphone, bg="#7986cb",
fg="white").pack(side=tk.LEFT, padx=5)

        self.text_area = scrolledtext.ScrolledText(self.root, wrap=tk.WORD, width=60,
height=18, font=("Arial", 12))
        self.text_area.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

        self.status_label = Label(self.root, text="Initializing...", bg="#f0f0f0", fg="blue")
        self.status_label.pack(pady=5)

        button_frame = tk.Frame(self.root, bg="#f0f0f0")
        button_frame.pack(pady=10)

        self.start_button = Button(button_frame, text="Start Listening",
command=self.toggle_listening,
                                   bg="#4CAF50", fg="white", width=15, state=tk.DISABLED)
        self.start_button.grid(row=0, column=0, padx=5)

        Button(button_frame, text="Save Transcript", command=self.save_transcript,
               bg="#2196F3", fg="white", width=15).grid(row=0, column=1, padx=5)

        Button(button_frame, text="Clear", command=self.clear_transcript,
               bg="#f44336", fg="white", width=15).grid(row=0, column=2, padx=5)

        Button(button_frame, text="Troubleshoot", command=self.show_troubleshooting,
               bg="#FF9800", fg="white", width=15).grid(row=1, column=1, padx=5, pady=10)
```

```
        self.root.protocol("WM_DELETE_WINDOW", self.on_closing)
        self.root.after(500, self.init_microphones)

    def init_microphones(self):
        if self.detect_microphones():
            self.refresh_mic_dropdown()
        else:
            self.status_label.config(text="No microphones detected!", fg="red")

    def refresh_microphones(self):
        self.status_label.config(text="Refreshing microphone list...", fg="blue")
        if self.detect_microphones():
            self.refresh_mic_dropdown()
            self.status_label.config(text="Microphone list refreshed", fg="green")
        else:
            self.status_label.config(text="Failed to detect microphones", fg="red")

    def refresh_mic_dropdown(self):
        if not self.mic_list:
            return
        mic_names = [f"{index}: {name}" for index, name in self.mic_list]
        self.mic_dropdown['values'] = mic_names
        if mic_names:
            self.mic_dropdown.current(0)
            self.selected_mic_index = self.mic_list[0][0]
            self.start_button.config(state=tk.NORMAL)
            self.status_label.config(text="Ready - Microphone selected", fg="green")
```

```
    def test_microphone(self):
        try:
            if not self.mic_list:
                messagebox.showerror("Error", "No microphones available to test")
                return
            selection = self.mic_dropdown.get()
            if not selection:
                messagebox.showerror("Error", "Please select a microphone first")
                return
            index = int(selection.split(":")[0])
            self.status_label.config(text="Testing microphone... Speak now", fg="blue")
            self.text_area.insert(tk.END, "\nTesting microphone... Say something...\n")
            microphone = sr.Microphone(device_index=index)
            with microphone as source:
                self.recognizer.adjust_for_ambient_noise(source, duration=1)
                try:
                    self.text_area.insert(tk.END, "Listening for 5 seconds...\n")
                    audio = self.recognizer.listen(source, timeout=5, phrase_time_limit=5)
                    self.text_area.insert(tk.END, "Processing audio...\n")
                    text = self.recognizer.recognize_google(audio)
                    self.text_area.insert(tk.END, f"Test successful! Heard: \"{text}\"\n")
                    self.status_label.config(text="Microphone test successful", fg="green")
                    messagebox.showinfo("Success", f"Microphone test successful!\nHeard:
\"{text}\"")
                    self.selected_mic_index = index
                except sr.WaitTimeoutError:
                    self.text_area.insert(tk.END, "No speech detected during test.\n")
                    self.status_label.config(text="No speech detected", fg="orange")
                except sr.UnknownValueError:
```

```
            self.text_area.insert(tk.END, "Speech not recognized.\n")
            self.status_label.config(text="Speech not recognized", fg="orange")
        except sr.RequestError as e:
            error_msg = f"API error: {e}"
            self.text_area.insert(tk.END, f"{error_msg}\n")
            self.status_label.config(text="API Error", fg="red")
    except Exception as e:
        self.text_area.insert(tk.END, f"Error testing microphone: {str(e)}\n")
        self.status_label.config(text="Microphone test failed", fg="red")

def toggle_listening(self):
    if not self.is_listening:
        if self.selected_mic_index is None:
            messagebox.showerror("Error", "Please select and test a microphone first")
            return
        self.is_listening = True
        self.start_button.config(text="Stop Listening", bg="#f44336")
        self.status_label.config(text="Listening...", fg="#2196F3")
        self.listening_thread = threading.Thread(target=self.listen_and_transcribe)
        self.listening_thread.daemon = True
        self.listening_thread.start()
    else:
        self.is_listening = False
        self.start_button.config(text="Start Listening", bg="#4CAF50")
        self.status_label.config(text="Stopped", fg="orange")

def listen_and_transcribe(self):
    try:
        microphone = sr.Microphone(device_index=self.selected_mic_index)
```

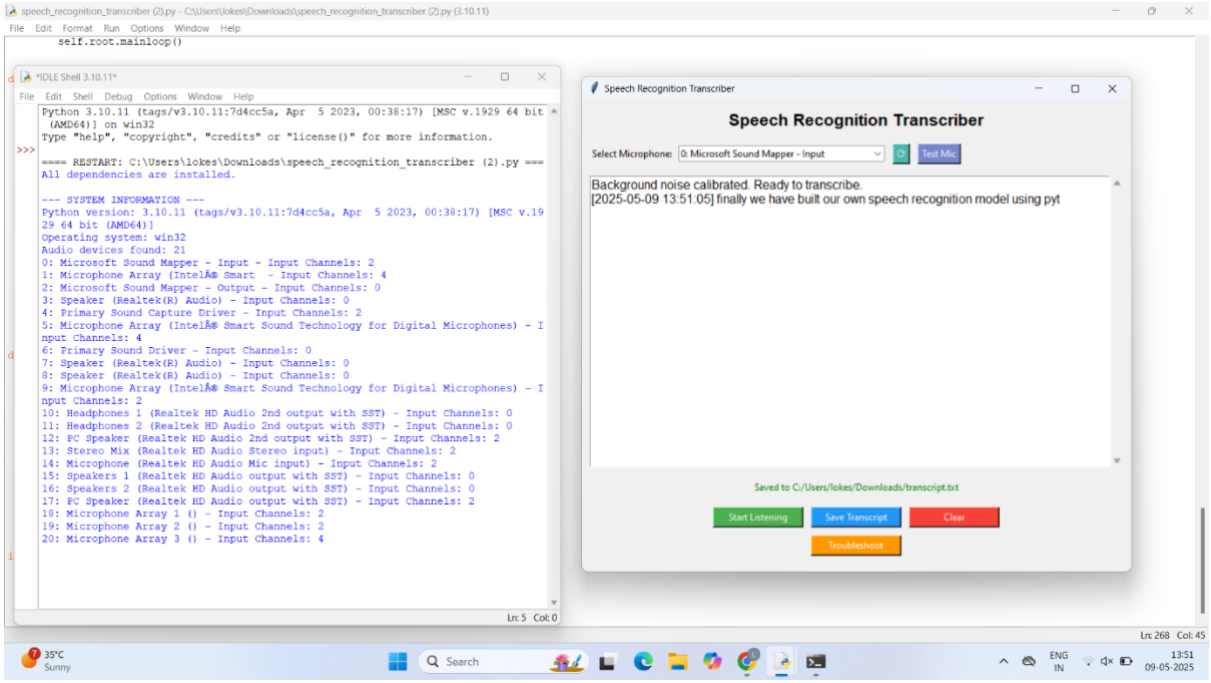
```
        with microphone as source:
            self.recognizer.adjust_for_ambient_noise(source, duration=1)
            self.text_area.insert(tk.END, "Background noise calibrated. Ready to
transcribe.\n")
            while self.is_listening:
                try:
                    self.status_label.config(text="Listening...", fg="#2196F3")
                    audio = self.recognizer.listen(source, timeout=10)
                    self.status_label.config(text="Processing...", fg="purple")
                    text = self.recognizer.recognize_google(audio)
                    if text:
                        timestamp = time.strftime("[%Y-%m-%d %H:%M:%S] ", time.localtime())
                        full_text = f"{timestamp}{text}\n"
                        self.transcript_text += full_text
                        self.text_area.insert(tk.END, full_text)
                        self.text_area.see(tk.END)
                        with open(self.transcript_file, "a", encoding="utf-8") as f:
                            f.write(full_text)
                        self.status_label.config(text="Success - Listening...", fg="green")
                except sr.WaitTimeoutError:
                    self.status_label.config(text="Timeout - Listening again...", fg="orange")
                except sr.UnknownValueError:
                    self.status_label.config(text="Speech not recognized", fg="orange")
                except sr.RequestError as e:
                    self.status_label.config(text=f"API error: {e}", fg="red")
    except Exception as e:
        self.status_label.config(text="Critical Error", fg="red")
        self.text_area.insert(tk.END, f"CRITICAL ERROR: {str(e)}\n")
        self.is_listening = False
```

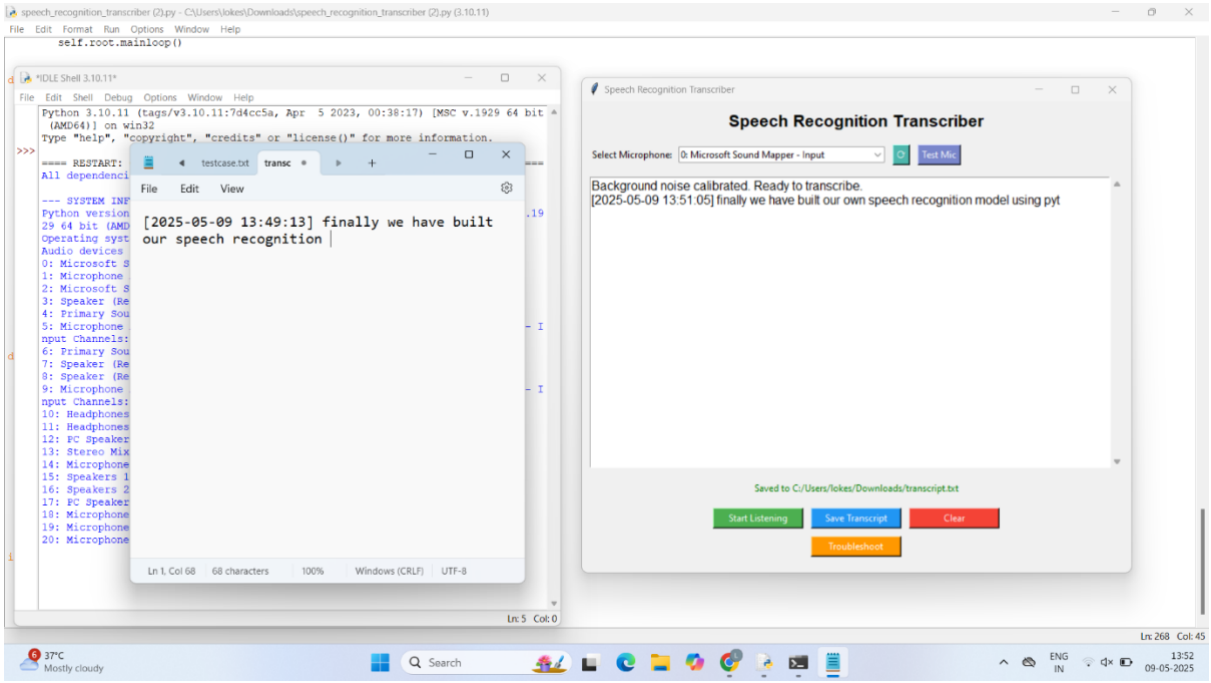
```
        self.start_button.config(text="Start Listening", bg="#4CAF50")

    def save_transcript(self):
        if not self.transcript_text:
            self.status_label.config(text="No transcript to save", fg="orange")
            return
        filename = filedialog.asksaveasfilename(defaultextension=".txt",
                                                filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")])
        if filename:
            with open(filename, "w", encoding="utf-8") as f:
                f.write(self.transcript_text)
            self.status_label.config(text=f"Saved to {filename}", fg="green")

    def clear_transcript(self):
        self.transcript_text = ""
        self.text_area.delete(1.0, tk.END)
        self.status_label.config(text="Transcript cleared", fg="green")

    def show_troubleshooting(self):
        troubleshoot_text = """
MICROPHONE TROUBLESHOOTING GUIDE:

1. Check physical connection.
2. Ensure mic is selected as default input.
3. Allow mic access in OS settings.
4. Close other apps using the mic.
5. Update mic drivers.
6. Try refreshing and retesting mic.
7. Restart system if needed.
```

```
"""
        messagebox.showinfo("Microphone Troubleshooting", troubleshoot_text)
        self.text_area.insert(tk.END, "\n--- TROUBLESHOOTING GUIDE ---\n" +
troubleshoot_text + "\n")
        self.text_area.see(tk.END)

    def on_closing(self):
        self.is_listening = False
        self.root.destroy()

    def run(self):
        self.root.mainloop()



def check_dependencies():
    missing_packages = []
    try:
        import speech_recognition
    except ImportError:
        missing_packages.append("SpeechRecognition")
    try:
        import pyaudio
    except ImportError:
        missing_packages.append("PyAudio")
    if missing_packages:
        print("Missing packages:", ", ".join(missing_packages))
        try:
            import subprocess
            for pkg in missing_packages:
```

```
            subprocess.call(["pip", "install", pkg])
        except Exception as e:
            print("Failed to install:", e)
    else:
        print("All dependencies are installed.")


def show_system_info():
    print("\n--- SYSTEM INFORMATION ---")
    print(f"Python version: {sys.version}")
    print(f"Operating system: {sys.platform}")
    try:
        import pyaudio
        p = pyaudio.PyAudio()
        print(f"Audio devices found: {p.get_device_count()}")
        for i in range(p.get_device_count()):
            info = p.get_device_info_by_index(i)
            print(f"{i}: {info['name']} - Input Channels: {info['maxInputChannels']}")
        p.terminate()
    except Exception as e:
        print(f"Failed to get audio info: {e}")


if __name__ == "__main__":
    check_dependencies()
    show_system_info()
    app = SpeechTranscriber()
    app.run()
```

Output Screenshot :





Results :

The results should include:

- A speech recognition pipeline that effectively reduces noise and extracts meaningful features.
- An acoustic model trained using HMMs and deep learning techniques.
- Improved accuracy compared to baseline models.
- Insights into the strengths and weaknesses of traditional vs. modern approaches.