Voice-to-Text Using Large Language Models (LLMs)

Introduction

Voice-to-text technology has transformed the way we interact with devices and process spoken information. By leveraging Large Language Models (LLMs), voice-to-text systems achieve unparalleled accuracy, multilingual support, and adaptability to complex audio conditions. This document explores key models such as OpenAI Whisper and Google Speech-to-Text API, comparing their features, capabilities, and applications. Additionally, it highlights the role of LLMs in enhancing speech recognition and transcription workflows.

OpenAI Whisper

Overview

OpenAI Whisper is a state-of-the-art speech recognition model designed for high accuracy in transcription tasks. It excels in handling diverse audio environments, including real-world noise, varying accents, and overlapping speech. Whisper is particularly valuable for applications requiring multilingual transcription and offline functionality.

Features

- **Multilingual Transcription**: Supports a wide range of languages, making it ideal for global applications.
- **Offline Capability**: Operates locally without requiring internet connectivity, ensuring privacy and reduced latency.
- **High Accuracy**: Maintains robust performance in noisy and complex audio environments, such as crowded areas or call centers.

Implementation

Whisper is open-source and easily integrates into Python-based workflows. Below is a sample implementation:

```
import pyaudio
import wave
import numpy as np
import whisper
import time

RATE = 16000
CHUNK = 1024
FORMAT = pyaudio.paInt16
CHANNELS = 1
SILENCE_THRESHOLD = 500
SILENCE_DURATION = 2

def is_silent(data_chunk):
    """Check if the chunk of audio is silent based on a threshold."""
    return np.max(np.abs(data_chunk)) < SILENCE_THRESHOLD

def record_audio():
    """Record audio from the microphone until the user stops talking."""
    p = pyaudio.PyAudio()
    stream = p.open(format=FORMAT, channels=CHANNELS, rate=RATE, input=True,
frames_per_buffer=CHUNK)

    print("Listening... Speak now.")
    frames = []
    silence_start_time = None

    try:
        while True:
            data = stream.read(CHUNK)
            audio_chunk = np.frombuffer(data, dtype=np.int16)
            frames.append(data)

            if is_silent(audio_chunk):
                if silence_start_time is None:
                    silence_start_time = time.time()
                elif time.time() - silence_start_time > SILENCE_DURATION:
                    print("Silence detected. Stopping recording.")
                    break
            else:
                silence_start_time = None

    except KeyboardInterrupt:
        print("\nRecording stopped by user.")

    stream.stop_stream()
    stream.close()
    p.terminate()
```

```
    filename = "temp_recording.wav"
    wf = wave.open(filename, "wb")
    wf.setnchannels(CHANNELS)
    wf.setsampwidth(p.get_sample_size(FORMAT))
    wf.setframerate(RATE)
    wf.writeframes(b"".join(frames))
    wf.close()

    return filename


def transcribe_audio(file_path):
    """Transcribe the recorded audio using the Whisper model."""
    model = whisper.load_model("small")
    print("Transcribing...")
    result = model.transcribe(file_path)
    return result["text"]
```

Advantages

- Operates without an internet connection, ensuring privacy and security.
- Handles diverse audio conditions effectively, including noise and overlapping dialogue.

Google Speech-to-Text API

Overview

Google Speech-to-Text API is a cloud-based transcription service known for its real-time capabilities and integration with other Google tools. It offers advanced features such as speaker diarization, enabling identification of different speakers in a conversation.

Features

- **Real-Time Streaming**: Provides immediate transcription of live audio.
- **High Accuracy**: Adapts to domain-specific vocabulary for improved precision.

- **Speaker Diarization**: Distinguishes between multiple speakers in a conversation.

Implementation

The API is simple to integrate but requires an internet connection. Below is an example usage:

```
import pyaudio
import wave
import numpy as np
import time
from googleapiclient.discovery import build
import googleapiclient.errors
import base64
import json

API_KEY = "AIzaSyCk5Imrq4N02LkKBRyJ3feWxRu37hTmdtc"

RATE = 16000
CHUNK = 1024
FORMAT = pyaudio.paInt16
CHANNELS = 1
SILENCE_THRESHOLD = 500
SILENCE_DURATION = 2
FILENAME = "temp_recording.wav"

def get_speech_client():
    """Create a Google Cloud Speech-to-Text API client."""
    return build("speech", "v1", developerKey=API_KEY)

def is_silent(audio_chunk):
    """Check if the audio chunk is below the silence threshold."""
    return np.max(np.abs(audio_chunk)) < SILENCE_THRESHOLD

def record_audio_with_silence():
    """Record audio until silence is detected."""
    p = pyaudio.PyAudio()
    stream = p.open(format=FORMAT, channels=CHANNELS, rate=RATE, input=True,
frames_per_buffer=CHUNK)

    print("Listening... Speak now.")
    frames = []
    silence_start_time = None

    try:
        while True:
```

```
            data = stream.read(CHUNK)
            audio_chunk = np.frombuffer(data, dtype=np.int16)
            frames.append(data)

            if is_silent(audio_chunk):
                if silence_start_time is None:
                    silence_start_time = time.time()
                elif time.time() - silence_start_time > SILENCE_DURATION:
                    print("Silence detected. Stopping recording.")
                    break
            else:
                silence_start_time = None

    except KeyboardInterrupt:
        print("\nRecording stopped manually.")

    stream.stop_stream()
    stream.close()
    p.terminate()

    wf = wave.open(FILENAME, "wb")
    wf.setnchannels(CHANNELS)
    wf.setsampwidth(p.get_sample_size(FORMAT))
    wf.setframerate(RATE)
    wf.writeframes(b"".join(frames))
    wf.close()

    print("Audio recording saved as:", FILENAME)
    return FILENAME

def transcribe_audio_google(file_path, speech_client):
    """Transcribe the recorded audio using Google Cloud Speech-to-Text."""
    print("Transcribing...")

    start_time = time.time()

    with open(file_path, "rb") as audio_file:
        content = audio_file.read()
        encoded_audio = base64.b64encode(content).decode("utf-8")

    request_payload = {
        "config": {
            "encoding": "LINEAR16",
            "languageCode": "en-US",
            "sampleRateHertz": RATE,
        },
        "audio": {
            "content": encoded_audio
```

```
        },
    }

    try:
        response =
speech_client.speech().recognize(body=request_payload).execute()

        end_time = time.time()
        transcription_time = end_time - start_time
        print(f"Time taken to transcribe: {transcription_time:.2f} seconds")

        if "results" in response:
            for result in response["results"]:
                transcript = result["alternatives"][0]["transcript"]
                print("\nTranscription:", transcript)
        else:
            print("No transcription results.")

    except googleapiclient.errors.HttpError as err:
        print(f"Error during transcription: {err}")

if __name__ == "__main__":
    speech_client = get_speech_client()
    audio_file = record_audio_with_silence()
    transcribe_audio_google(audio_file, speech_client)
```

Advantages

- Seamless integration with Google's ecosystem, including Google Cloud and Google Workspace.
- Scalable and suitable for enterprise-level applications.

Comparison: OpenAI Whisper vs. Google Speech-to-Text API

Feature	OpenAI Whisper	Google Speech-to-Text API
Accuracy	High	High
Multilingual Support	Yes	Limited

Feature	OpenAI Whisper	Google Speech-to-Text API
Real-Time Capabilities	Yes (Local)	Yes (Cloud)
Offline Functionality	Yes	No
Cost	Free (Self-hosted)	Paid (API usage)

Applications of Voice-to-Text Systems Using LLMs

1. **Customer Service**
    - Automating transcription of customer support calls.
    - Analyzing customer sentiment through live call transcription.
2. **Healthcare**
    - Transcribing doctor-patient consultations for record-keeping.
    - Supporting real-time note-taking during medical conferences.
3. **Education**
    - Capturing lecture audio and converting it into text for study materials.
    - Assisting students with disabilities through live transcription.
4. **Content Creation**
    - Transcribing interviews for journalism.
    - Converting podcasts and video audio into written formats.
5. **Legal and Compliance**
    - Recording and transcribing legal proceedings for documentation.
    - Ensuring compliance by maintaining accurate meeting records.

Conclusion

Voice-to-text technology powered by LLMs like OpenAI Whisper and Google Speech-to-Text API offers transformative potential across industries. With advanced features such as multilingual support, high accuracy, and real-time transcription, these tools cater to diverse use cases. By understanding their strengths and choosing the right solution for specific needs, businesses and developers can harness the power of LLMs to improve efficiency, accessibility, and innovation in voice-to-text workflows.