**Voice Recognition and Transcription Project: Detailed Explanation and AI/ML Concepts**

This project involves developing a voice recognition and transcription system using machine learning (ML) and artificial intelligence (AI) techniques. The primary goal is to allow users to record an audio clip and transcribe the content into text using speech-to-text models.

Here's a detailed breakdown of the project, its internal working, the AI/ML concepts used, the reasons behind choosing specific libraries, and the deep workings of each installed package.

**1. Overview of the Project**

The project is designed to:

1. **Record Audio**: Capture a short audio clip using a microphone (via sounddevice).
2. **Process the Audio**: Convert the recorded audio into text using an AI model for transcription (via whisper).
3. **Display Transcription**: Provide the transcribed text as an output.

It leverages **Whisper**, a state-of-the-art ASR (Automatic Speech Recognition) model by OpenAI, alongside other Python libraries for audio processing and file management.

**2. Main Libraries and Packages Used**

1. **whisper (Speech-to-Text Model)**
   * **Purpose**: This is the core AI component of the project. Whisper is an automatic speech recognition (ASR) model developed by OpenAI. It transcribes spoken language into text with high accuracy.
   * **Why Chosen**: Whisper is known for its high performance in transcription tasks, supporting multiple languages and being highly robust under noisy conditions, which is ideal for transcription purposes.
   * **Internal Working**:
     + Whisper uses **deep learning** techniques based on **transformers** (a type of neural network architecture) for sequence-to-sequence tasks. It takes audio features (like spectrograms) as input and outputs text sequences.
     + The model is pre-trained on large datasets, making it capable of handling diverse accents, background noise, and language variations.
     + Whisper processes the audio by breaking it into smaller chunks (frames), extracts spectrograms, and then uses an encoder-decoder transformer network to map the spectrograms to text sequences.
2. **sounddevice (Recording Audio)**
   * **Purpose**: This library is used for recording audio from the microphone.
   * **Why Chosen**: sounddevice is a simple and effective library to interact with sound devices. It provides a clean and straightforward API to record high-quality audio and also offers control over parameters like sample rate, number of channels, and duration.
   * **Internal Working**:
     + The library interfaces with the sound hardware and captures audio data in the form of numerical arrays.
     + It uses **blocking I/O**, meaning that the sd.rec() function waits for the recording to finish before returning control to the program.
3. **noisereduce (Noise Reduction)**

**Purpose**: This library is used to remove background noise from the recorded audio before it is transcribed by Whisper.

**Why Chosen**: Noisereduce is simple to use and effective in reducing environmental noise from audio recordings using signal processing techniques like spectral gating.

**Internal Working**:

* The noisereduce library performs spectral gating by estimating the background noise and subtracting it from the signal.
* To ensure that it doesn’t mistakenly suppress voice along with noise, the system extracts a short segment (e.g., the **first 1 second** of the audio) assuming it contains only background noise.
* This segment is used to build a **noise profile**, and then denoising is applied to the entire recording.
* The audio is also converted to **mono** before processing to simplify the signal and avoid stereo inconsistencies.
* The final cleaned audio is saved and passed to the transcription model.

**Important Note for Accurate Results**:

The user should **wait 1–2 seconds before speaking** during recording. This allows the system to capture clean background noise for accurate profiling. Otherwise, the denoiser may mistakenly remove parts of the speech signal.

1. **scipy (Saving Audio Files)**
   * **Purpose**: scipy is used for saving the recorded audio to a WAV file format, which is a standard format for storing raw audio data.
   * **Why Chosen**: scipy provides efficient methods for writing audio data to files and is compatible with various machine learning libraries, making it a reliable tool for audio data handling.
   * **Internal Working**:
     + The write() function from scipy.io.wavfile takes the audio data array, sample rate, and file name and writes the audio to the specified WAV file. It saves the audio in a binary format that is easy to process by models like Whisper.
2. **soundfile (Reading and Writing Audio Files)**  
   o **Purpose:** soundfile is used for reading and writing audio files in various formats with high accuracy and safety.  
   o **Why Chosen:** soundfile offers better precision and more reliable handling of audio files compared to scipy, supporting a wider range of audio formats and providing efficient file I/O operations.  
   o **Internal Working:**  
   ▪ soundfile uses the libsndfile library under the hood, which handles audio data at a low level.  
   ▪ It supports reading and writing in multiple audio file formats, including WAV, FLAC, and OGG.  
   ▪ The library allows precise control over audio data types and supports seamless conversion between file formats and numpy arrays for processing.  
   ▪ The read() and write() functions manage audio data safely, preserving the original sample rates and bit depths for accurate audio representation.
3. **numpy (Array Operations)**
   * **Purpose**: numpy is used for handling numerical data, especially the audio arrays.
   * **Why Chosen**: It is a fundamental library for scientific computing in Python, providing efficient array operations. Audio data recorded by sounddevice is essentially an array of values, and numpy is perfect for handling such data.
   * **Internal Working**:
     + numpy provides fast, efficient manipulation of large arrays, which is useful when processing and analyzing audio data (like rescaling, reshaping, etc.).
4. **openai-whisper (Speech Recognition AI)**
   * **Purpose**: This is the core library responsible for transcription, converting speech into text.
   * **Why Chosen**: OpenAI’s Whisper is pre-trained on a massive and diverse dataset, enabling it to transcribe audio in a variety of languages and under noisy conditions. Its robust performance in real-world scenarios makes it ideal for this application.
   * **Internal Working**:
     + Whisper uses an end-to-end neural network that takes raw audio (in the form of spectrograms) and outputs text.
     + The model is based on a transformer architecture, which is well-suited for sequence-to-sequence tasks like translation or transcription.
     + Whisper uses both supervised and unsupervised learning, trained on massive datasets, making it able to transcribe from multiple languages and accents with high accuracy.
5. **spaCy (NLP for Keyword Extraction)**

* **Purpose:** Used to extract key phrases and noun chunks from the transcribed text, providing meaningful insight from audio content.
* **Why Chosen:** spaCy offers efficient, production-grade NLP capabilities including part-of-speech tagging, parsing, and noun chunk detection. It's lightweight and highly accurate.
* **Internal Working:**
  + Tokenizes the transcribed text and tags each word with part-of-speech info.
  + Identifies noun phrases using syntactic dependencies.
  + Filters and ranks these phrases to extract the most relevant keywords from the transcript.

**3. AI/ML Concepts and Internal Depth**

1. **Audio Data Representation (Spectrogram)**
   * Audio is typically represented in a digital format as a sequence of samples. To feed audio into machine learning models like Whisper, the raw waveform is converted into a **spectrogram**, which is a time-frequency representation of the audio.
   * **Spectrograms** provide a 2D image-like representation of the audio signal, which can capture temporal and frequency patterns that are crucial for understanding speech.
2. **Deep Learning (Transformer Models)**
   * The core of Whisper's transcription ability lies in **transformers**. These are neural networks designed to handle sequential data by using self-attention mechanisms. This allows them to efficiently process and understand long sequences (like spoken words in an audio file).
   * **Self-Attention**: This mechanism allows the model to weigh the importance of different parts of the input sequence (audio frames in this case), helping it understand contextual relationships within the audio.
3. **Sequence-to-Sequence Learning**
   * Whisper is designed as a **sequence-to-sequence** model, where the input is a sequence of audio frames (or spectrogram), and the output is a sequence of text tokens.
   * **Encoder-Decoder Structure**: The encoder converts the input audio sequence into a dense representation, and the decoder generates the text sequence from this representation.
4. **Transfer Learning**
   * Whisper has been trained on a wide variety of datasets, meaning it leverages **transfer learning**. It uses knowledge learned from a large, diverse dataset to handle real-world audio and languages it wasn't specifically trained on. This allows Whisper to perform well in a variety of situations, even with accents, noise, or uncommon languages.

**4. Why These Libraries Were Chosen**

* **whisper**: It is state-of-the-art in speech recognition. Its ability to work with diverse audio inputs and handle noisy environments made it the best choice for transcription.
* **sounddevice**: This is one of the most user-friendly libraries for audio input in Python, which makes it simple to record real-time audio from the microphone and pass it to the transcription model.
* **scipy**: Its reliable WAV file saving functionality ensures that the audio is stored in a format that can be easily loaded and processed by Whisper.
* **numpy**: It’s crucial for handling the raw audio data and performing mathematical operations on it efficiently, particularly when manipulating large arrays or matrices.

**5. System Workflow**

1. Recording Audio: Audio is recorded using sounddevice and saved as a WAV file (output.wav) using scipy.
2. Noise Reduction (New): Before transcription, the recorded audio is processed with noisereduce to reduce background noise. The cleaned audio is saved as denoised\_output.wav.
3. Transcription: The denoised file is passed to OpenAI’s Whisper model, which transcribes it into text using deep learning.
4. Displaying and Saving Output: The transcription is printed to the console and saved to a timestamped .txt file.
5. Keyword Extraction: spaCy processes the transcription to extract relevant noun phrases, helping users identify key terms or topics in the spoken content.

**6. Project Structure**

* **record\_audio.py**: Script for recording audio from the microphone and saving it as output.wav.
* **transcribe.py:**
  + Now includes noise reduction using noisereduce before transcription.
  + Loads and transcribes denoised\_output.wav using Whisper.
  + Extracts keywords using spaCy and saves the result.

**7. Potential Improvements**

* **Noise Cancellation**: Implement noise reduction techniques to clean up audio before transcription, which will improve accuracy in noisy environments.
* **Multiple Languages**: Although Whisper supports multiple languages, adding a language detection system could enhance the model's accuracy for mixed-language environments.
* **Real-Time Transcription**: Implement real-time transcription while the audio is being recorded using a streaming model.

**Conclusion**

This project demonstrates the power of combining various Python libraries to achieve a robust voice recognition and transcription system. By leveraging **Whisper**, a pre-trained state-of-the-art ASR model, and **sounddevice**, a simple library for audio input, we can easily build a speech-to-text system. The use of **scipy** and **numpy** ensures smooth audio handling and data processing, while the deep learning concepts in Whisper’s transformer model make transcription accurate and efficient.

**8. Accuracy and Performance Expectations**

**Word Error Rate (WER) and Accuracy**

* Accuracy in speech-to-text systems is often measured using the Word Error Rate (WER), which calculates how many words in the transcribed text differ from the reference (ground truth) transcript.
* WER = (Substitutions + Deletions + Insertions) / Number of words in reference
* Accuracy = (1 - WER) × 100%

**Expected Accuracy Levels with Whisper**

| **Model Variant** | **Typical WER (%)** | **Approximate Accuracy (%)** |
| --- | --- | --- |
| Tiny | 25 - 35 | 65 - 75 |
| Base | 15 - 25 | 75 - 85 |
| Small | 10 - 20 | 80 - 90 |
| Medium | 7 - 15 | 85 - 93 |
| Large | 5 - 10 | 90 - 95+ |

* The current project uses the **base** Whisper model, which usually delivers **accuracy between 75% and 85%** on moderately clean audio.
* Accuracy depends heavily on factors such as:
  + **Audio quality:** Clear recordings with minimal background noise yield better transcription.
  + **Noise reduction:** Your implemented noise reduction step helps improve accuracy significantly.
  + **Speaker accent and clarity:** Standard accents and clear speech improve transcription results.
  + **Recording environment:** Quiet surroundings without overlapping voices or echoes enhance transcription quality.

**Improving Accuracy**

* Using larger Whisper models (medium or large) can increase accuracy but require more computational resources.
* Further improvements include:
  + Enhanced noise cancellation techniques.
  + Language and accent adaptation.
  + Real-time feedback and correction mechanisms.

**How to Evaluate Accuracy in the Project**

Evaluating how well the transcription works is an important part of the project. To measure this, I’ve used the jiwer library to calculate something called **Word Error Rate (WER)**. This gives a quick idea of how close the transcribed output is to the actual (ground truth) sentence.

**1. What is WER (Word Error Rate)?**

WER is a standard metric used in speech recognition. It compares the model’s output (what it thinks was said) to the real, correct sentence.

**Formula:**

WER=S+D+I/N

Where:

* **S** = Substitutions (words that were wrong)
* **D** = Deletions (words that were missed)
* **I** = Insertions (extra words that shouldn’t be there)
* **N** = Total number of words in the reference transcript

We can also calculate accuracy like this:

Accuracy=(1−WER)×100%Accuracy = (1 - WER) \times 100\%Accuracy=(1−WER)×100%

A lower WER and higher accuracy mean the transcription is more accurate.

**2. Files Required for Evaluation**

To evaluate WER, we need two text files:

* **reference.txt**: This is the correct transcript. You can create this manually or use a known correct sentence.
* **hypothesis.txt**: This is the transcript generated by Whisper (from transcribe.py).

Make sure both are plain .txt files, encoded in UTF-8, and contain just one sentence each (no extra formatting).