Brief Report

Not peer-reviewed version

Precision Location Keyword Detection Using Offline Speech Recognition Technique

Mohsin Imam * and Gaurav Gupta

Posted Date: 7 November 2023

doi: 10.20944/preprints202310.0690.v2

Keywords: Keyword Detection; Audio Models; Speech Processing

Brief Report

Precision Location Keyword Detection Using Offline Speech Recognition Technique

Mohsin Imam [1,*] and Gaurav Gupta [2]

[1] Department of Computer Science, Atma Ram Sanatan Dharma College, University of Delhi, New Delhi, India
[2] Institute of System Studies & Analysis, Defence Research and Development Organization, New Delhi, India
* Correspondence: mohsinimam651@gmail.com

Abstract: This study introduces an original comprehensive system centered on identifying specific terms that indicate a user's position, particularly the numerical values representing latitude and longitude. This system not only detects these terms but also retrieves the corresponding numerical data for accurate and efficient determination of locations. The importance of this endeavour encompasses various fields, notably aiding offline operations of military personnel, who often lack internet access. In such contexts, precise awareness of location is vital for strategic manoeuvres, rescue operations, and navigating unfamiliar landscapes. The system empowers these personnel by allowing them to extract exact location coordinates from spoken terms, thereby enhancing their awareness even in challenging surroundings. Apart from its military utility, the project holds broader significance. Teams responding to emergencies, personnel involved in disaster management, and exploratory missions can all gain from this technology during disruptions in communication infrastructure. Furthermore, travelers, adventurers, and outdoor enthusiasts can utilize this system to accurately determine their positions in remote areas without relying on online maps.

Keywords: keyword detection; audio models; speech processing

The methodology of this study revolves around the application of offline speech recognition techniques to precisely transcribe spoken terms, achieving an accuracy of over 91.3% and a word error rate of 4.2%. For sound recognition, the OpenAI Whisper model was employed, and a conversion process from SpeechRecognition to AudioSegmentation was implemented, followed by transforming the audio into .wav format, we have also developed the interface of the app to use it efficiently using Streamlit. This was done to ensure seamless compatibility with the Whisper model and uninterrupted audio input. By training the system to identify specific linguistic cues linked to location, it achieves robust detection and extraction of relevant terms. This innovative approach eliminates the necessity for constant internet connectivity, rendering it exceptionally useful in remote, offline, and resource-limited situations.

Introduction

The process of speech recognition focuses on transforming spoken language into written text. This endeavour enhances and promotes more human-like communication, as the goal of this recognition method is to empower individuals to interact in a manner that is both instinctive and proficient. Implementing continuous speech recognition is a complicated task due to its need to handle various language features like word lengths, co-articulation effects, and casual pronunciations, each unique to different voices. One effective solution involves employing language models, particularly n-grams, to calculate the most accurate estimations for context-related probabilities.

Several reasons drive the need for adaptive interfaces incorporating speech recognition. Firstly, it aligns with the natural way humans communicate. Secondly, traditional touch interfaces pose challenges for composing lengthy texts. As a result, using speech recognition accelerates the utilization and retrieval of information in software and applications, circumventing the use of

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 7 November 2023 doi:10.20944/preprints202310.0690.v2

traditional input methods like keyboards, mice, and touchscreens. Progress in speech recognition methods has opened up the possibility of applying this technology across numerous contexts, particularly on portable gadgets. Individuals with unique requirements also experience advantages from these systems. Those unable to employ their hands, as well as individuals with visual impairments, utilize this innovation to communicate and manage diverse computer operations using voice commands. Various Application Programming Interfaces (APIs) & Speech recognition models are available to help incorporate voice recognition into software and applications. Examples include Web speech, Java speech, Google cloud speech, and Bing speech, among others. Yet, none of these enable offline recognition. This implies that users need an active internet connection, posing a significant obstacle. This issue is particularly critical in countries like Brazil, where just about half of the population has internet access (IBGE, 2014). This concern is notable due to the vital role voice recognition plays in enhancing accessibility. Speech recognition without an internet connection is computationally complex and needs lots of memory. Yet, this demand isn't restrictive, as modern smartphones are getting stronger. To manage offline recognition, an effective approach involves combining a neural network and a statistical model. This combo offers efficient processing and memory use, helping save smartphone battery and other resource limited devices.

We've introduced a method that involves using the foundational structure of Whisper, a speech-to-text library developed by OpenAI. In this approach, we've incorporated a technique for identifying specific keywords that indicate the speaker's location. Through a dedicated algorithm, we extract these keywords and decipher their associated values, which play a role in pinpointing where the speaker is located. What sets our system apart is its complete offline functionality. This signifies that the entire process, ranging from capturing the voice recording, extracting the keywords, to detecting their corresponding values, operates independently without relying on external internet and other network connections.

Speech-2-Text Models

Speech-to-Text (STT) technology allows you to turn any audio content into written text. It is also called Automatic Speech Recognition (ASR), or computer speech recognition. Speech-to-Text is based on acoustic modelling and language modelling. Note that it is commonly confused with voice recognition, but it focuses on the translation of speech from a verbal format to a text one whereas voice recognition just seeks to identify an individual user's voice.

Speech-to-Text APIs offer versatile applications across various industries. They find significant utility in call centres, enabling the analysis of customer data to detect trends. In banking, these APIs enhance customer communications, ensuring security and efficiency. Automation benefits from STT, enabling tasks like appointment scheduling and order tracking. Governance and security sectors employ STT for customer identification and verification. In healthcare and media, STT drives voice-driven medical reports, form filling, and content conversion into searchable text across TV, radio, and social networks. In the table below, a list of some popular speech-2-text APIs/services are mentioned.

Table 1. Popular Text-2-Speech Model.

Company Name	API Name	Release Year	Pricing	Supported Languages	Accuracy/WER
Assembly AI	AssemblyAI S2T	2020	Paid	9	16.8 (WER)
AWS	AWS Transcribe	2017	Paid	75	18.42% (WER)
Wit.ai	Wit Speech	2015	Paid	132	—
Microsoft	Microsoft Azure Speech	2018	Paid	147	9.2% (WER)
Vosk	VoskAPI	2020	Free	20	63.4% Acc.

Whisper	OpenAI	2022	Free	99	11.4 % / 4.2 % (best model-EN)
IBM	IBM Watson S2T	2017	Paid	13	11.3%-36.4%
Google	Google S2T	2017	Paid	119	16% (WER)

I have chosen Whisper by OpenAI for text-to-speech work due to its compelling combination of factors that make it a strong choice in the field. Whisper, released in 2022, offers a substantial library of 99 supported languages, which is one of the widest languages supports among the options listed. Moreover, its accuracy, with the best model achieving an impressive 11.4% Word Error Rate (WER) and an even more impressive 4.2% WER on English text-to-speech, makes it one of the most accurate options available. Notably, Whisper is free to use, making it an accessible choice for individuals and businesses looking for a cost-effective solution without compromising on quality. These factors, the wide language support, high accuracy rates, and cost-effectiveness, justify using Whisper for text-to-speech work for keyword detection. In addition to its extensive language support and impressive accuracy rates, another key factor that influenced my choice of Whisper for text-to-speech work is its efficiency in terms of resource utilization. The Whisper base model stands out for its minimal memory and computational resource requirements, all while delivering a remarkable 4.2% Word Error Rate (WER). This means that it not only offers top-tier accuracy but does so without imposing a heavy burden on hardware or cloud resources, making it an efficient and practical choice for a wide range of applications, from small-scale projects to large-scale deployments. Whisper's ability to strike a balance between accuracy and resource efficiency further solidifies its position as a compelling choice for text-to-speech tasks.

OpenAI Whisper

Whisper represents an automated speech recognition (ASR) system that has undergone training using 680,000 hours of diverse and multilingual supervised data garnered from online sources. The utilization of this extensive and varied dataset results in enhanced resistance against accents, ambient disturbances, and specialized terminology. Additionally, this facilitates the conversion of speech into text across numerous languages, along with subsequent translation into English. The models and inference code are being made available through open-source to establish a base for creating practical applications and to encourage continued exploration in the domain of resilient speech analysis. Whisper's design is a straightforward end-to-end technique, realized as an encoder-decoder Transformer model. The initial audio is divided into segments of 30 seconds each, transformed into a log-Mel spectrogram, and fed into an encoder. Concurrently, a decoder is educated to foresee the related textual description. This procedure includes distinct tokens that instruct the unified model to execute various functions like recognizing languages, marking timestamps for phrases, transcribing multilingual speech, and translating speeches to English.

Whisper's training used a broad dataset without focusing on any single one, causing it to not outperform specialized models on the highly competitive LibriSpeech benchmark. However, when examining Whisper's performance on various datasets without prior tuning, it proves notably robust, displaying a 50% reduction in errors compared to those models. Around 33% of Whisper's audio data is in languages other than English. It's assigned the task of either transcribing in its original language or translating into English. This technique is particularly successful in mastering speech-to-text translation, surpassing the state-of-the-art supervised method on CoVoST2-to-English translation without specific training.

Whisper serves as a versatile speech recognition system. It's trained on an extensive range of audio samples and functions as a multitasking entity, capable of executing tasks like recognizing speech in multiple languages, translating spoken words, and identifying languages. The training employs a Transformer model specialized in converting sequences, tackling diverse speech-related duties like recognizing speech in different languages, translating spoken content, identifying spoken language, and detecting voice activity. These roles are collectively encoded as a series of forecasted tokens for the decoder, the training architecture of Whisper is shown in Figure 1. This approach enables one model to replace multiple steps of a conventional speech processing setup. The multi-task training technique involves distinct tokens used for indicating tasks or acting as goals for classification.

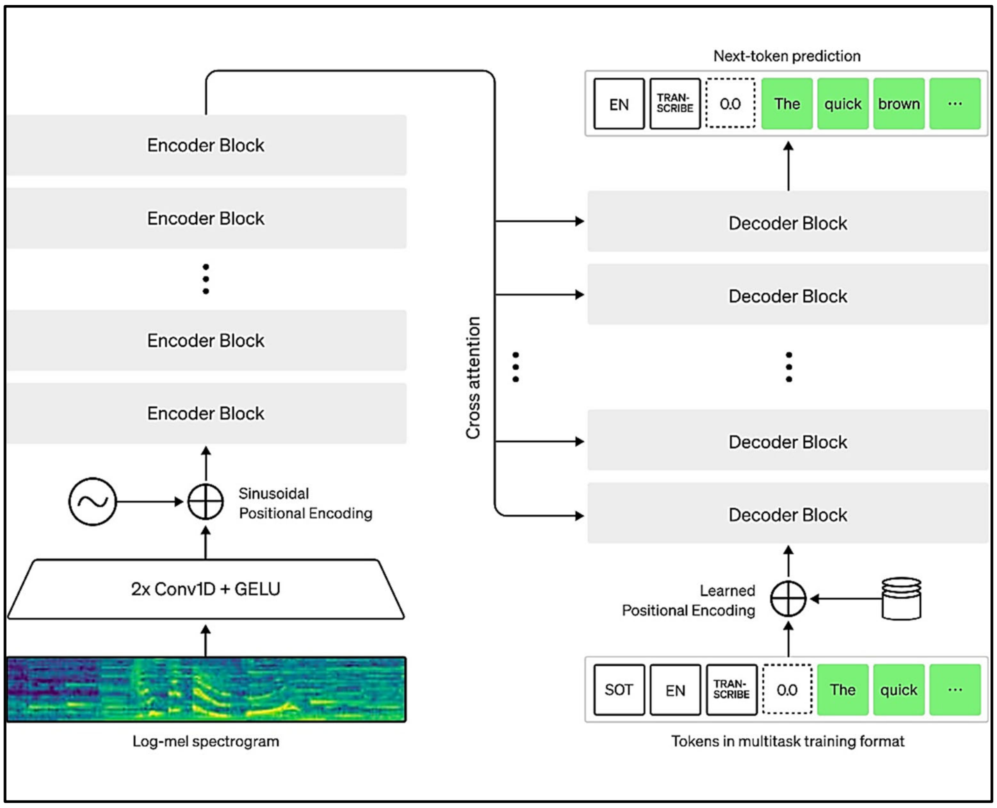


Figure 1. Base architecture of OpenAI's Whisper.

Input audio is split into 30-second chunks, converted into a log-Mel spectrogram, and then passed into an encoder. A decoder is trained to predict the corresponding text caption, intermixed with special tokens that direct the single model to perform tasks such as language identification, phrase-level timestamps, multilingual speech transcription, and to-English speech translation.

There are five model sizes, four with English-only versions, offering speed and accuracy trade-offs. Below are the names of the available models and their approximate memory requirements and relative speed.

Table 2. Various Whisper's Model.

Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	tiny.en	tiny	~1 GB	~32x
base	74 M	base.en	base	~1 GB	~16x
small	244 M	small.en	small	~2 GB	~6x
medium	769 M	medium.en	medium	~5 GB	~2x
large	1550 M	N/A	large	~10 GB	1x

System Overview

We only used the Whisper base model because higher models require expensive computational hardware that was not practical at the time of our experiment and because we seek to extract location from the text, which typically doesn't encompass a large range of variety. Designing the location keyword and its value extraction, designing the audio conversion to.wav/mp3 for continuous audio recording, and finally designing the transcription of the recorded using model and passing it to get ingested in the interface, are the three phases of high-level implementation, the system overview is visualized in Figure 2 below.

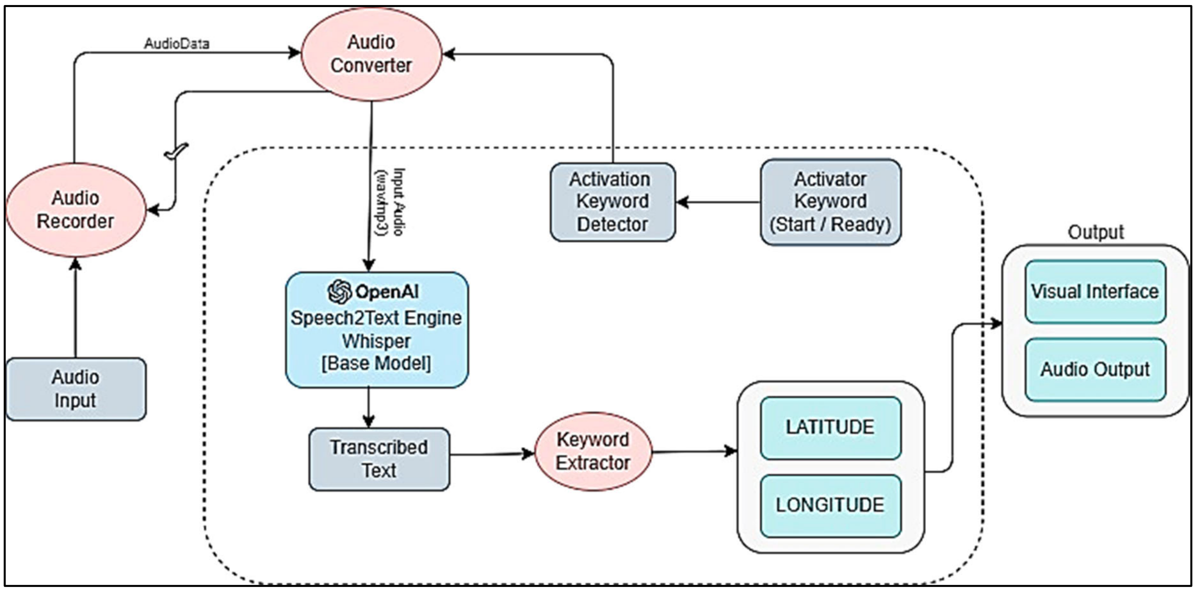


Figure 2. System Overview.

Implementation

The system functions via a two-stage sequence launched by a recognition keyword, similar to verbal cues such as "Start" or "Ready." This keyword acts as the system's trigger, resembling the way devices like Alexa or Google Home are activated. Upon triggering, the system comes to life. For instance, if the keyword "start" is detected, it signifies the user's intent to initiate the system's audio recording for location detection, starting the microphone to capture spoken content. For this purpose, the SpeechRecognition library is employed, which facilitates audio recording from the microphone. The library establishes a Recognizer object designed for speech recognition and integrates the microphone as the primary audio source. During this phase, a status message is displayed, signifying the commencement of recording. Subsequently, the system utilizes the listen() method within the Recognizer object to capture audio inputs from the microphone. Once recording is complete, the resultant audio is encapsulated within an AudioData object.

To align the recorded audio with the requirements of the Whisper model, which expects data in byte or np.array formats, a conversion function has been implemented. This function translates the AudioData object into either .wav or mp3 formats. The converted audio file is then channelled into the transcriber model, facilitating its transcription in the text format which is further processed.

The audio output undergoes processing through the conversion model, specifically the base model of Whisper developed by OpenAI. This formidable model boasts an impressive array of over 70 million parameters, meticulously trained across a staggering 680,000 hours of diverse multilingual audio data. This extensive training equips the model with unparalleled proficiency in accurately capturing spoken words, irrespective of nuances like word accent or other linguistic intricacies.

Initially, we collected transcribed text derived from a pretrained Whisper model, by sending the received audio from the convertor function we discussed above. Subsequently, this text is then sent to a specialized location keyword detection and extractor function named extract_lat_long_from_text. The extracted latitude and longitude values were then seamlessly presented on the Streamlit interface, offering users an insightful visualization.

The core of the implementation exist in in the main() function. Starting with the Streamlit interface initiation titled "Latitude and Longitude Detection," accompanied by a descriptive sub header, users are prompted to record their geographic coordinates. Using the whisper.load_model() function, the Whisper model's base version is instantiated. Within the Streamlit interface, distinct sections for latitude and longitude outputs are initialized. The sr.Microphone() context manager establishes the microphone source, enabling the ongoing capture of user input. Within an infinite loop, the code alertly awaits user input from the microphone. The identification of the activation keyword "start" within user input serves as a initialization of the system. This prompts the system to audibly signal initiation through a startup tone, subsequently commencing the recording of audio data which is converted to a suitable format via a conversion function.

The extract_lat_long_from_text() function is then used to extract the transcription from the MP3 audio by utilizing the Whisper model's capabilities. Latitude and longitude data are precisely determined from the transcribed text by the systematic extraction method. The retrieved geographic information is displayed on the Streamlit interface, which has a stronger emphasis on user-friendly visualization.

When there are problems with voice recognition or service outages, understandable error messages are provided right away. The culmination of this dynamic implementation, which combines audio recording, complex speech recognition, and exact position extraction, is an easy user interface that makes it simple to get geographic data using speech input.

Sound Conversion Algorithm:

The function converted(audio) takes an audio object as input and performs a series of operations aimed at converting the audio format, providing the outcome as a result:

1. Utilize the pydub library's AudioSegment.from_wav() function. The initial audio input is converted into the WAV format using this function, which creates an audio_segment object. Notably, the io.BytesIO() function is employed to handle and process the audio data produced from the input object, which facilitates this transformation.
2. Use the audio_segment object's native export() method. With this technique, the audio is preserved by having it stored as an MP3 file" The output format is wisely specified by setting the format argument to "mp3," which is a directive.
3. Allow for the modification of the audio variable, requiring that it be updated to reflect the location of the just created MP3 audio file ("check.mp3").
4. The method finally completes its work by supplying an enhanced audio variable that includes the path to the freshly converted MP3 audio file. The transformed audio content can then be accessed quickly thanks to this route.

Location Extraction Algorithm:

The following steps make up the algorithm for extracting latitude and longitude from text: First, the text input is used to launch the method extract_lat_long_from_text(). The text is then broken up into its component words, which are then kept in a list named words. Latitude and longitude are then set up as two empty variables to contain the corresponding data. The program examines the digit status of each word in the list, removes the period, and determines whether it is a number. The method continues if the input is a number; else, it moves on to the following word. If latitude is empty, the word is designated as the value for latitude inside the validation; if latitude already has a value, the word is marked as the value for longitude inside the validation; and the loop is terminated using the break statement. After correctly extracting the latitude and longitude data from the given text, the algorithm concludes. It is crucial to notice that this technique assumes that the text comprises latitude and longitude values separated by words, with latitude coming before longitude. If an accurate longitude or latitude cannot be determined, the corresponding variables are left empty. The step by step implementation of the extraction algorithm is given below.

ALGORITHM: `extract_lat_long_from_text` functioning

```
1. Start the function extract_lat_long_from_text(text) with text as input.
2. Split the input text into individual words using the split() function, and store the result in the
   words list.
3. Initialize an empty string variable latitude to store the latitude value and another empty string
   variable longitude to store the longitude value.
4. For each word in the words list, do the following:
     a) Check if the current word is a number by removing one period (.) from it using
        replace('.', '', 1) and then checking if the result is a digit using the isdigit() function.
     b) If the word is indeed a number, proceed to the next step; otherwise, skip this iteration.
5. Within the if block, check if the latitude variable is empty:
     a) If latitude is empty, assign the current word as the value of latitude.
     b) If latitude is already assigned, assign the current word as the value of longitude.
6. Exit the loop using the break statement after assigning the longitude value.
7. The function ends, having captured the latitude and longitude values from the input text.
```

Accuracy

Precise evaluations of speech-to-text model performance are crucial for their trustworthiness and usefulness. Word Error Rate (WER) and similar methods are essential for this evaluation. WER checks how well predicted words match the actual words, flagging errors and inconsistencies. This metric is vital for perfecting models by pinpointing common errors and areas to enhance. Accurate assessments not only improve transcription quality but also broaden their application, benefiting various domains. This extends from aiding the hearing-impaired with accessibility services to streamlining professional tasks like automated transcription, ultimately fostering smoother communication and easier access to information, and has eventually helped us in our keyword detection task.

Algorithm for Word Error Rate (WER)

Word Error Rate is a common metric for comparing Audio Speech Recognition. It compares a reference with a hypothesis.

$$Word\ Error\ Rate\ =\ (S\ +\ I\ +\ D)\ /\ N$$

where,

S — is the number of substitutions,
D — is the number of deletions,
I — is the number of insertions and
N — is the number of words in the reference
REFERENCE: Some of words
HYPOTHESIS: Sum of words
"Substitution" is happening in this case. 'Some' is substituted by 'Sum'.

WER calculation is based on Levenshtein distance at the word level. Algorithm for WER is as follows:

doi:10.20944/preprints202310.0690.v2

Algorithm 1: Calculation of WER with Levenshtein distance function

```
1   function WER(Reference r, Hypothesis h)
2       int [|r| + 1 |h| + 1] D
    *Initialization
3       for (i = 0; i <= |r|; i++) do
4           for (j = 0; j <= |h|; j++) do
5               if i == 0 then
6                   D[0][j] ← j
7               else if j == 0 then
8                   D[i][0] ← i
9               end if
10          end for
11      end for
12  for (i = 1; i <= |r|; i++)                                  *Calculation Part
13          for (j = 1; j <= |h|; j++) do
14              if r[i - 1] == h[j - 1] then
15                  D[i][j] ← D[i - 1][j - 1]
16              else
17                  sub ← D[i - 1][j - 1] + 1
18                  ins ← D[i][j - 1] + 1
19                  del ← D[i - 1][j] + 1
20                  D[i][j] ← min(sub, ins, del)
21              end if
22          end for
23  end for
```

We conducted a series of 30 iterations to assess the performance of our developed keyword detection application. In these experiments, we had Word Error Rate (WER) of 4.2 as we utilized whisper's base model on English language (since our application only supports English language). The resulting metrics were as follows: an F1 score of 0.905, a precision score of 0.92, and a recall score of 0.89. These values were obtained by assuming an overall accuracy target of approximately 91.3%, and the confusion matrix of our experiment is shown in Figure 3. To calculate accuracy, we employed the formula that considers both precision and recall. Our comprehensive evaluation provides insights into the robustness of our system, with an emphasis on balancing precision and recall. These results demonstrate that our application excels in identifying keywords related to location within speech, with high precision and recall rates. This is a promising indication of our system's effectiveness in capturing location-related information accurately from spoken language. Our commitment to fine-tuning and optimization has allowed us to achieve these promising results, paving the way for further improvements and real-world applications of our speech recognition technology.

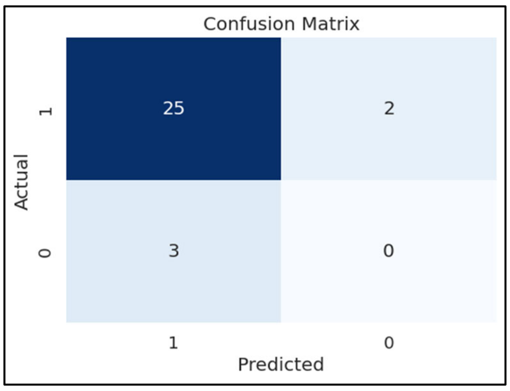


Figure 3. Starting screen of the system.

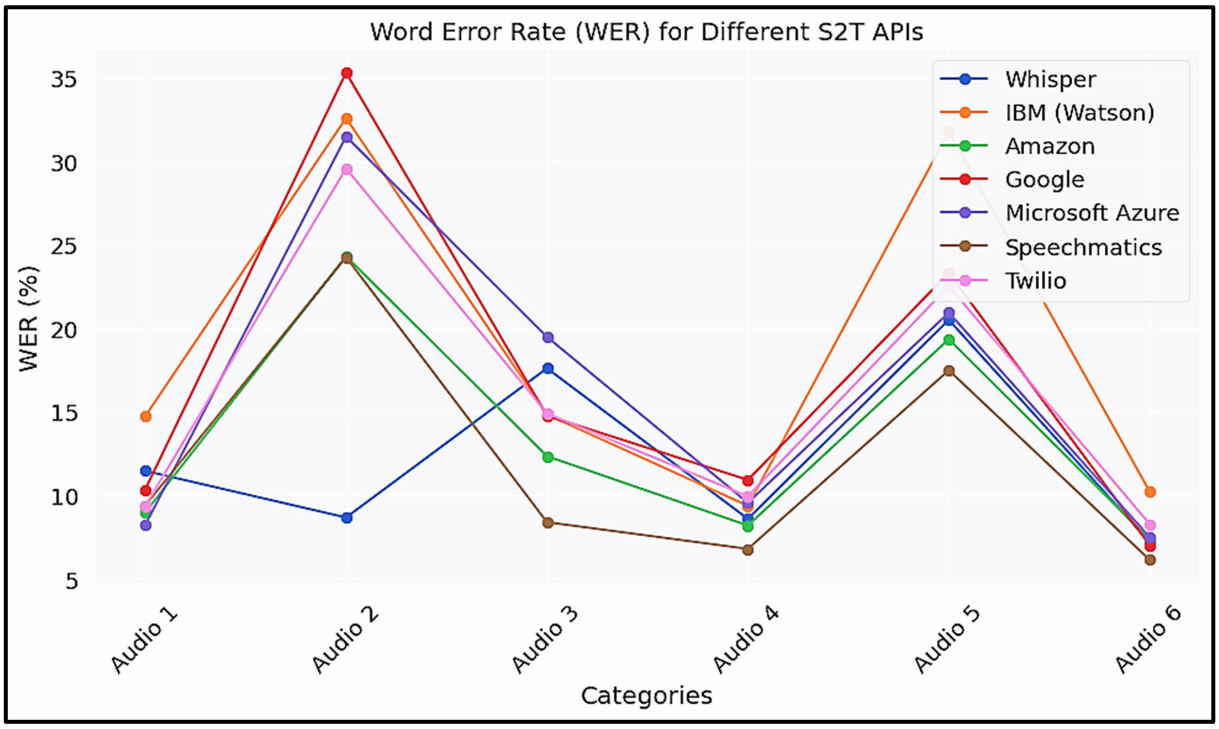
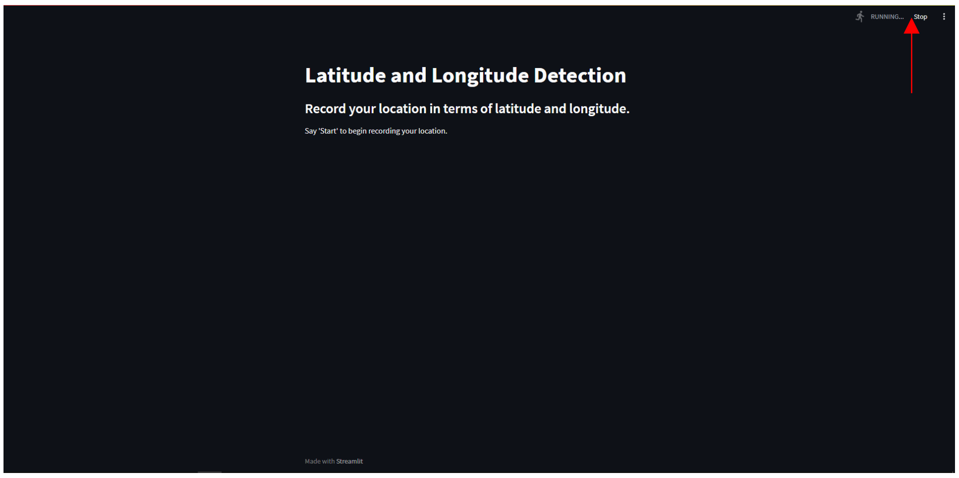


Figure 4. WER Analysis of various S2T models on six audio data.

Interface Illustration



Latitude and Longitude Detection

Record your location in terms of latitude and longitude.

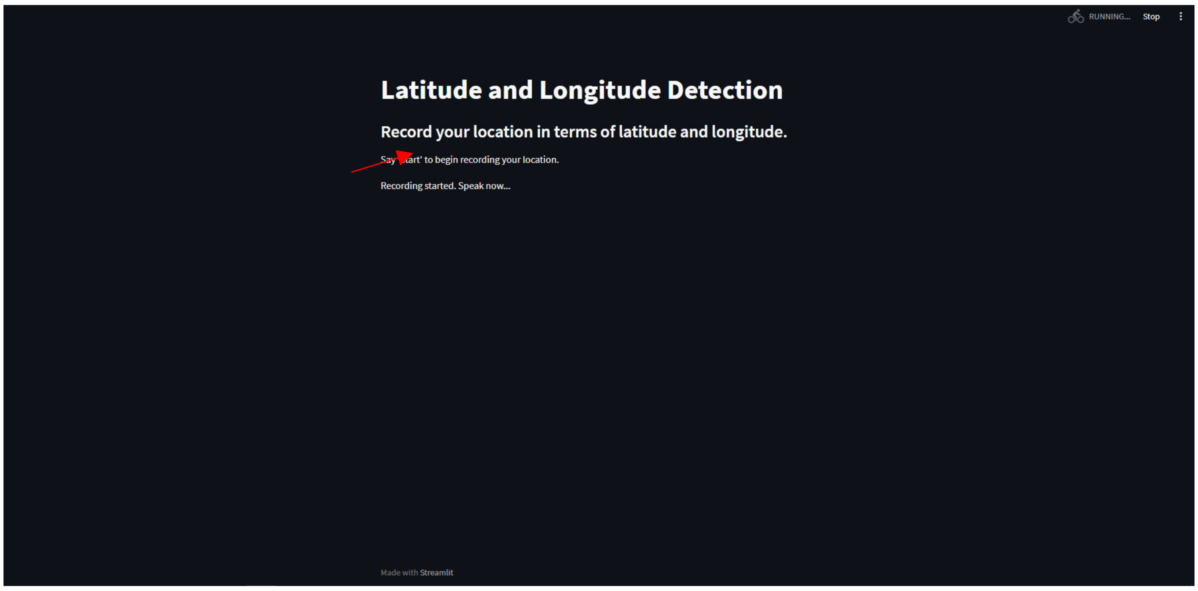
Say 'Start' to begin recording your location.

Made with Streamlit

Figure 5. Starting screen of the system.

doi:10.20944/preprints202310.0690.v2

The starting interface of the produced system is shown in the image above, along with the header and a prompt asking the user to utter the designated activation keyword in order to wake the system. Additionally, we can see the RUNNING state in the screen's upper right corner (depicted by a red arrow), which is a result of the infinite loop we set up and is for the system's continuous recording so that it would start if the user says the activation keyword.



Latitude and Longitude Detection

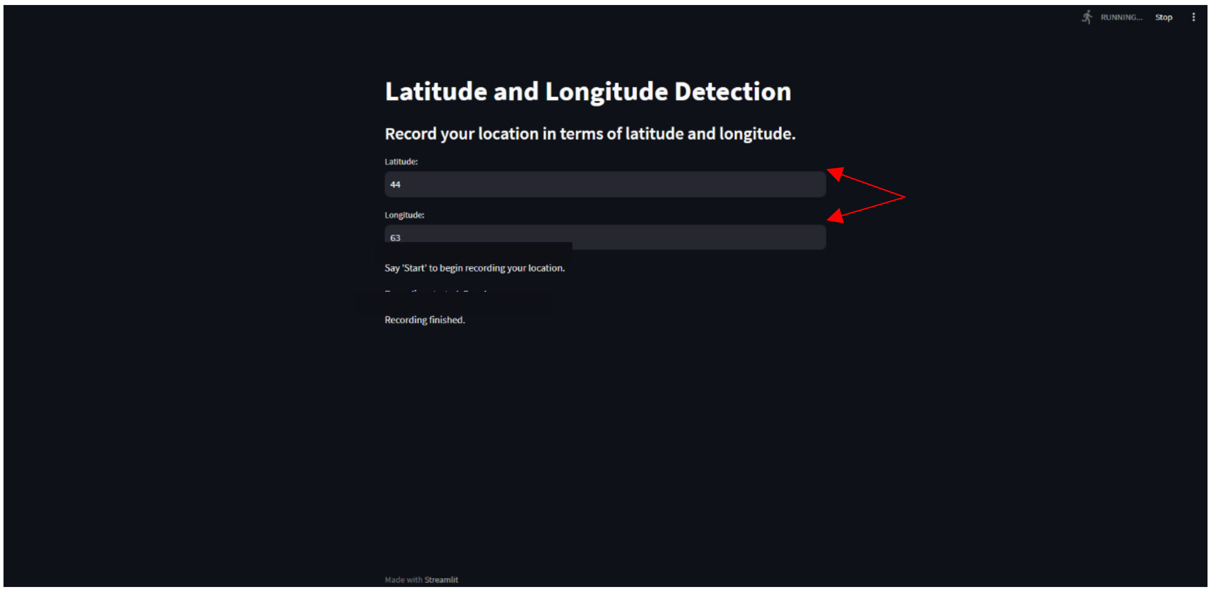
Record your location in terms of latitude and longitude.

Say 'Start' to begin recording your location.

Recording started. Speak now...

Figure 6. Activation Keyword Detected.

In the above screenshot, as we can see that the activation keyword is detected, and a message stating that recording has started, and prompting the user to speak.



Latitude and Longitude Detection

Record your location in terms of latitude and longitude.

Latitude: 44

Longitude: 63

Say 'Start' to begin recording your location.

Recording finished.

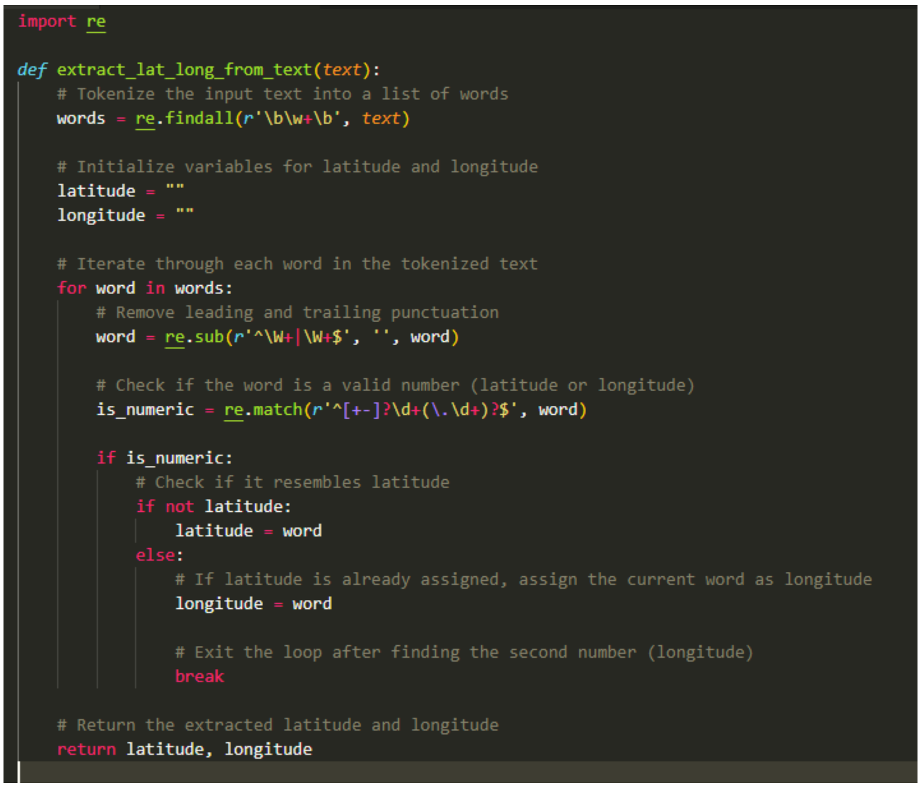
Figure 7. Activation Keyword Detected.

Finally, we can see that the audio has been successfully converted into text. According to the algorithms we have developed, the words related to the user's location—in this case, Latitude and Longitude—have also been identified. The values associated with those keywords are displayed on the screen, and once more, the system has begun to voice the recording until something is said.

Conclusion

This project introduces an innovative system that addresses the critical challenge of accurate location determination, particularly in offline and resource-limited scenarios. By employing advanced offline speech recognition techniques, the system can transcribe spoken terms with remarkable accuracy, achieving an over 91.3% accuracy rate and a 4.2% word error rate. The utilization of the OpenAI Whisper model, along with sophisticated audio conversion and transcription algorithms, empowers the system to identify specific keywords indicative of geographic coordinates. This breakthrough holds immense value, especially for military personnel operating offline, emergency response teams, exploratory missions, and outdoor enthusiasts. By providing a seamless and user-friendly interface, this technology bridges the gap between speech input and precise geographic information, enhancing accessibility and awareness across various domains.

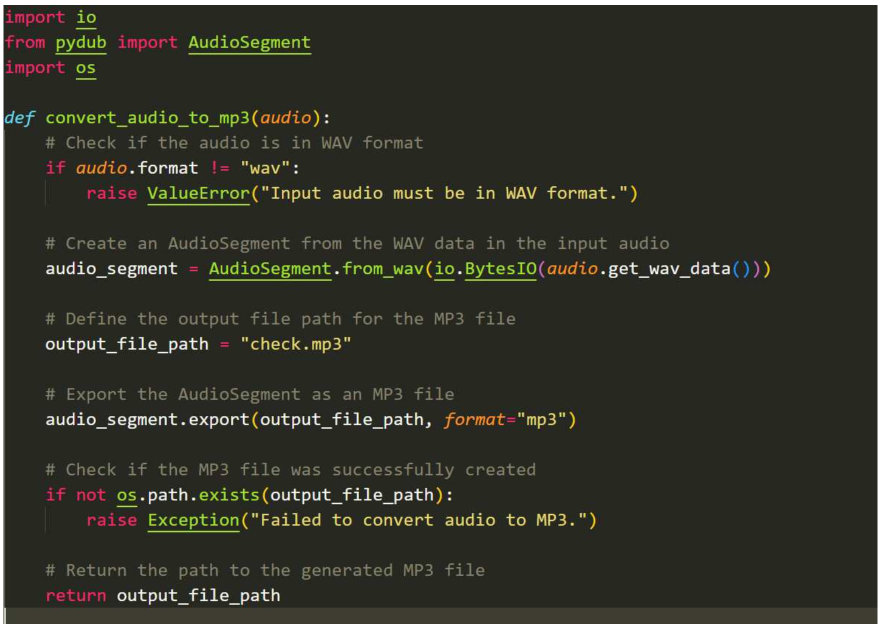
Key Code Snippets



```python
import re

def extract_lat_long_from_text(text):
    # Tokenize the input text into a list of words
    words = re.findall(r'\b\w+\b', text)

    # Initialize variables for latitude and longitude
    latitude = ""
    longitude = ""

    # Iterate through each word in the tokenized text
    for word in words:
        # Remove leading and trailing punctuation
        word = re.sub(r'^\W+|\W+$', '', word)

        # Check if the word is a valid number (latitude or longitude)
        is_numeric = re.match(r'^[+-]?\d+(\.\d+)?$', word)

        if is_numeric:
            # Check if it resembles latitude
            if not latitude:
                latitude = word
            else:
                # If latitude is already assigned, assign the current word as longitude
                longitude = word

                # Exit the loop after finding the second number (longitude)
                break

    # Return the extracted latitude and longitude
    return latitude, longitude
```

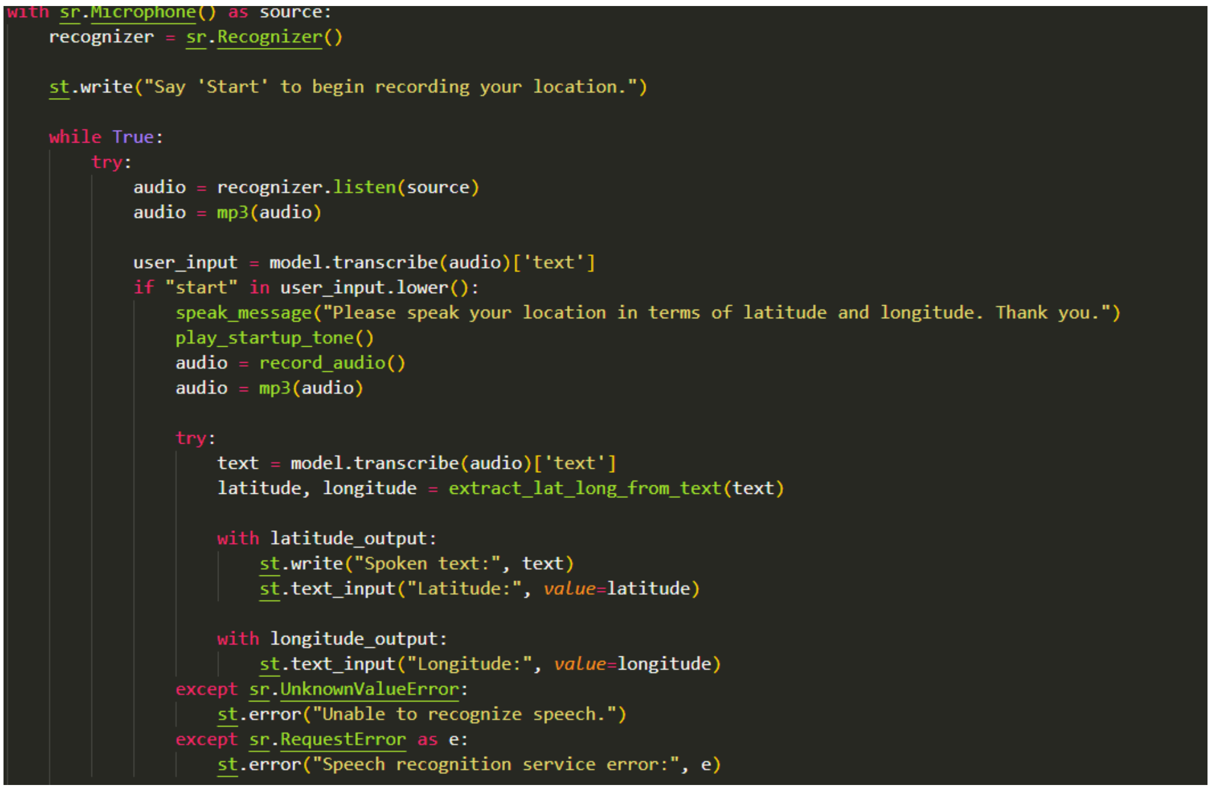
Figure 8. Keyword Extraction Algorithm.

doi:10.20944/preprints202310.0690.v2



```python
import io
from pydub import AudioSegment
import os

def convert_audio_to_mp3(audio):
    # Check if the audio is in WAV format
    if audio.format != "wav":
        raise ValueError("Input audio must be in WAV format.")

    # Create an AudioSegment from the WAV data in the input audio
    audio_segment = AudioSegment.from_wav(io.BytesIO(audio.get_wav_data()))

    # Define the output file path for the MP3 file
    output_file_path = "check.mp3"

    # Export the AudioSegment as an MP3 file
    audio_segment.export(output_file_path, format="mp3")

    # Check if the MP3 file was successfully created
    if not os.path.exists(output_file_path):
        raise Exception("Failed to convert audio to MP3.")

    # Return the path to the generated MP3 file
    return output_file_path
```

Figure 9. Sound Conversion Algorithm.



```python
with sr.Microphone() as source:
    recognizer = sr.Recognizer()

    st.write("Say 'Start' to begin recording your location.")

    while True:
        try:
            audio = recognizer.listen(source)
            audio = mp3(audio)

            user_input = model.transcribe(audio)['text']
            if "start" in user_input.lower():
                speak_message("Please speak your location in terms of latitude and longitude. Thank you.")
                play_startup_tone()
                audio = record_audio()
                audio = mp3(audio)

                try:
                    text = model.transcribe(audio)['text']
                    latitude, longitude = extract_lat_long_from_text(text)

                    with latitude_output:
                        st.write("Spoken text:", text)
                        st.text_input("Latitude:", value=latitude)

                    with longitude_output:
                        st.text_input("Longitude:", value=longitude)
                except sr.UnknownValueError:
                    st.error("Unable to recognize speech.")
                except sr.RequestError as e:
                    st.error("Speech recognition service error:", e)
```

Figure 10. Sound to text conversion, keyword extraction.