Speaker Diarization: A path to offline solutions

Team Bravo

Milestone 4 – Research Paper

Software Engineering Project

SWEN 670

July 29, 2021

Version 1.0

1. Introduction

Speaker Diarization is the system from which audio files are analyzed to identify differing inputs from multiple sources. These sources of sound can come from multiple speakers, music, or background noise and pollute an audio file with extraneous sounds. Diarization aims to identify and separate each sound profile from an audio file to then improve quality or extract speech from the desired speaker (Tranter, 2006). Diarization has most commonly been an offline system in PC's or devices that can handle higher levels of computation. However, due to the personalization and additional security in cell phones, speaker diarization has been optimized by moving its calculations to online cloud-based services. This provides the means for a high-quality and secure speaker diarization service without using a lot of space on your local storage. The online systems that provide speaker diarization have then expanded their services to cover multiple audio file analyses, such as speech-to-text and speech-to-intent, so that they can be a single provider for multiple services. While these solutions help optimize space, security issues are always a concern. To be able to overcome these concerns, a local, lightweight system is an ideal solution for speaker diarization.

The aim of this paper is to understand the current available options for offline speech diarization services. This will be done by reviewing the methods for speaker diarization and then analyzing three current online solutions, Azure, Watson and Google Speech to text. By analyzing all the online solutions, we can compare them to offline solutions, Kaldi, Alize, and pyAudioAnalysis, and understand their differences. This will then provide context to the offline solutions, formulate ideal scenarios to use them, and then explain how to integrate them into mobile devices. The research indicates that while an offline speaker diarization solution for mobile devices is available, due to its high processing needs, the solution sacrifices accuracy or

efficiency. If these deficiencies are acceptable for the application, there are many steps that need occur before full integration can occur.

2. System of Speaker Diarization

There are many different algorithms used to be able to achieve speaker diarization. In the past, the most popular method was to graphically analyze the sound waves that were retrieved, compare them to known models and then segment and separate soundwaves, finally identifying a single speaker, or source of sound, from others (Ramon, 2019). These complex algorithms were used until AI was able to compute and process the audio files faster and more efficiently through spectral clustering and neural networks (IBM, 2020). This solution proved to increase accuracy of translated speech and speaker identification.

Both algorithms are effective in speaker diarization as each one is used in many of the modern solutions. These solutions, however, are heavyweight and require a lot of processing power and space. A problem that modern computing has solved.

3. Online Solutions

By using cloud storage and cloud computing, many services have been able to provide effective APIs that accept audio files and provide a response containing the text that each speaker provides and a confidence of speech identification accuracy. Some of the top-of-the-line speech recognition and speech to text solutions are Watson, Azure and Google speech-to-text. These solutions use AI and state of the art processing methods to be able to provide their users speech to text services. Even with the similarities of processing, each solution provides unique features to assist developers for different tasks.

Azure's solution is a robust system that provides a confidence level for speaker identification and the text that is understood from the speech (Azure, 2021).

Azure API:

```
curl --location --request POST 'INSERT_ENDPOINT_HERE/speaker/verification/v2.0/text-dependent/profiles/INSERT_PROFILE_ID_HERE/verify' \ --header 'Ocp-Apim-Subscription-Key: INSERT_SUBSCRIPTION_KEY_HERE' \ --header 'Content-Type: audio/wav' \ --data-binary 'INSERT_FILE_PATH_HERE'
```

Azure Response:

```
{ "recognitionResult": "Accept", "score": 1.0}
```

Azure's solution is specific by identifying a unique speaker compared to a profile created on its system. Azure addresses the security of its users by not allowing profiles to be created with identifiers of the user and by requiring a subscription tag to access the profile. Azure assigns every profile a unique ID that has no correlation to the user. By doing this Azure removes identification and security weaknesses from the identification process.

IBM boasts its constantly improving AI neural networks and the constant improvement of their diarization solution. Watson's API is unique as it not only identifies speaker, but also provides options for alternative transcripts for what is understood:

```
curl -X POST -u "apikey:{apikey}" --header "Content-Type: audio/flac" --data-binary @{path_to_file}audio-file.flac "{url}/v1/recognize?timestamps=true&max_alternatives=3"
```

Response:

```
{ "results":
      [ { "alternatives":
            [ { "timestamps":
                  [ ["several":, 1.0, 1.51],
```

```
                ["tornadoes":, 1.51, 2.15],
                ["touch":, 2.15, 2.5], . . . ] },
            { "confidence": 0.96
                "transcript": "several tornadoes touch down as a line of
            severe thunderstorms swept through Colorado on Sunday " },
            { "transcript": "several tornadoes touched down as a line of
            severe thunderstorms swept through Colorado on Sunday " },
            { "transcript": "several tornadoes touch down as a line of
      severe thunderstorms swept through Colorado and Sunday " } ],
  "final": true } ],
 "result_index": 0 }
```

Google Speech-To-Text API is an online solution that separates each speaker and presents the transcript for each speaker. While it does not specify a speaker like the other systems, it can separate the speech from every speaker that is identified:

```
curl -s -H "Content-Type: application/json" \
    -H "Authorization: Bearer $(gcloud auth application-default print-access-
token)" \
    https://speech.googleapis.com/v1p1beta1/speech:recognize \
    --data '{
    "config": {
        "encoding": "LINEAR16",
        "languageCode": "en-US",
        "enableSpeakerDiarization": true,
        "diarizationSpeakerCount": 2,
        "model": "phone_call"
    },
    "audio": {
        "uri": "gs://cloud-samples-tests/speech/commercial_mono.wav"
    }
}' > speaker-diarization.txt
```

Response:

```
{
  "results": [
    {
      "alternatives": [
        {
```

```
          "transcript": "hi I'd like to buy a Chromecast and I was wondering
whether you could help me with that certainly which color would you like we have
blue black and red uh let's go with the black one would you like the new
Chromecast Ultra model or the regular Chrome Cast regular Chromecast is fine
thank you okay sure we like to ship it regular or Express Express please terrific
it's on the way thank you thank you very much bye",
          "confidence": 0.92142606,
          "words": [
            {
              "startTime": "0s",
              "endTime": "1.100s",
              "word": "hi",
              "speakerTag": 2
            },
            {
              "startTime": "1.100s",
              "endTime": "2s",
              "word": "I'd",
              "speakerTag": 2
            },
            {
              "startTime": "2s",
              "endTime": "2s",
              "word": "like",
              "speakerTag": 2
            },
            {
              "startTime": "2s",
              "endTime": "2.100s",
              "word": "to",
              "speakerTag": 2
            },
            ...
            {
              "startTime": "6.500s",
              "endTime": "6.900s",
              "word": "certainly",
              "speakerTag": 1
            },
            {
              "startTime": "6.900s",
              "endTime": "7.300s",
              "word": "which",
              "speakerTag": 1
            },
            {
              "startTime": "7.300s",
              "endTime": "7.500s",
```

```
            "word": "color",
            "speakerTag": 1
          },
          ...
        ]
      }
    ],
    "languageCode": "en-us"
  }
 ]
}
```


These online solutions provide robust services that prioritize security. By providing an API and hosting their solutions online, each has removed the weight of the solution from needing to host on the device. Each API is capable of streaming audio files and approach diarization in unique ways to give applications a variety of features in user verification. By analyzing these online solutions, we can view offline solutions and see how they compare, and how to best to utilize them.

4. Offline Solutions

Due to the complexity of the processing, speech diarization solutions have been stored on devices with a lot of threading and processing capabilities. To make the transition to mobile devices has been difficult due to the amount of storage and processing that can be taken to accommodate the speech diarization. Some possible solutions are python-based solutions that have been developed using pyAudioAnalysis library. PyAudioAnalysis is an open-source library that uses python to complete various algorithms to achieve the speech diarization. When tested using 32 different scripts of various lengths, the pyAudioAnalysis only has an accuracy of 84% under ideal conditions (Mane, 2020). This also affects the further breakdown of speaker identification because each solution is responsible for their own processing beyond graphical audio breakdown.

Another option for offline speaker diarization is Kaldi. In 2009, a John's Hopkins University team sought to create the state-of-the-art speech diarization system, Kaldi (Khudanpur, 2016). Kaldi has gained multiple contributors and is a constantly growing system. Kaldi can do speech to text and diarization in multiple languages and is constantly being updated by its many contributors. It is a free and open-source system that also provides packaging for android devices.

Kaldi also acknowledges its drawbacks of being a heavyweight system and suggests users to limit their system integration to lightweight models and usage. While it provides an Android compatible package, there is no IOS package. It is an accurate system but lacks in efficiency in processing. While research and development are still ongoing, at its current state, Kaldi, and applications that use it, have drawbacks that can affect the average mobile device user.

Finally, Alize is the third offline library that assists in offline speech diarization that is attempting to build plugins for mobile device use. Alize is a powerful tool that is written and C++ and then connected to Java through an API. It is used in pc speech diarization for desktop applications. It has built its system to allow the developer to define more of its speech parameters than other libraries. By following the gaussian model of vector modeling, it allows the user to define the audio files of the speaker, an audio file of background noise to expect, and then cleans and compares a third audio file sent to library. Like Kaldi, Alize has put effort into making an Android specific library. By removing a lot of the features and giving the developer the main parts of the library, it allows for a more compact solution to use for testing in Android.

There has been research done on this library, providing an architecture for speech diarization using this library and having an 82% success rate (Sharafeddin, 2015). The research shows that an IOS library is more than possible because of the native code being written in C++. Out of the

already built solutions, this library is the most diverse and capable of being added to multiple systems. However, due to the developer needing to provide their own audio files, it is also the one that requires the most set up.

Each solution also doesn't address the need for translation. The hurdle of getting it onto a phone has been complete, but if the application that is being developed is not written in the same language as the diarization solution, then more code and effort would need to go into translation or creating a plugin. Unlike an API call that is possible in most languages, a system of communication between application and solution would need to be created.

5. Implementation of Offline Solutions

To use the pyAudioAnalysis library in an application, developers would have to create their own diarization systems from the analysis provided from pyAudioAnalysis. The library can process audio files to graphical arrays and even complete speech to text, but speaker identification and graphical analysis must be processed on the side of the developer. The developed system could then be wrapped and integrated into an application but would need to create two separate systems for both Android and IOS. A flutter package or application class could be developed to interact with pyAudioAnalysis using starflut, a public package published on dart.pub in February 2021. Starflut allows flutter to interact with scripting languages such as Python to be used on both Android and iOS devices (Starflut, 2021). Another plugin that could be used that appears to be more stable but is limited to Android only is Chaquopy. Using Chaquopy the python code is accessed using wrapper classes interacting with the Chaquopy SDK (Chaquopy, 2021). In the flutter application this would be done by setting the configuration of the plugin to include a python.pip object which points to your python dependencies. To implement the python code

using starflut, the .py files could then be included as assets within an assets folder and loaded with the loadLibrary() method within starflut. Starflut has been under development since July 2018, however using their example application as a reference along with the pyAudioAnalysis documentation, the estimate for development of a plugin or custom class to be used in a flutter application would be a minimum of four months. This would be a similar timeline for Chaquopy, and it would be limited to the Android platform only.

The best system of reference on how to integrate Kaldi is the toolkit OpenVINO (OpenVINO, 2021). OpenVINO integrates with Kaldi and optimizes the accuracy by assisting in removing excess layers of sound in audio files. OpenVINO works in conjunction with Kaldi, requiring both to be setup on the device, and is optimized when using workhorse Intel processors (OpenVINO, 2021). OpenVINO processes the audio file, sends it to the Kaldi system, and then uses command line analysis to retrieve the output and present it on their UI. In a similar fashion, a mobile device library would need to be able to complete the command line analysis to retrieve the speakers and text processing, and then create a process to identify specific speakers. This system must then be packaged and added to the application. The developer Nick Fisher has a good start to a plugin started in github at the following URL: kaldi_asr_plugin. This could be adapted into a plugin to be run locally along with the flutter project files. It relies primarily on a OnlineKaldiDecoder() class which creates an instance of the Kaldi decoder using a directory to a log file, and the sample frequency of the input audio. The plugin has its first commits in December 2020 and is still currently being updated (Flutter Kaldi ASR Plugin, 2021). Based on this information, a reasonable amount of time for a similar plugin to be developed, tested and deployed in an application would be approximately six months to full implementation. This could be reduced by

an increased number of developers working on the plugin and depending on how much of the existing plugin can be used as a reference point.

An Alize integration would require integrating with the Alize for Android solution developed by Alize Speaker Recognition. Interactions with the Alize library would require an interface with their Java libraries. One way this could be accomplished is by writhing a MethodChannel within

the Flutter client-side code which can communicate with the platform using asynchronous calls (Mwiti, 2020). A CHANNEL object by the same name would have to be declared in the Java code in order for the communication to be successful (Mwiti, 2020). Inside the Android files, the MainActivity file would need a declaration to explicitly use java code such as:

flutter create -i objc -a java native_code

After this, once the MethodChannel has been initialized, the classes and methods in the Java files can be accessed by the MethodChannel. For example, a method call could be performed with a line similar to the following:

await platformMethodChannel.invokeMethod('powerManage');

This would run the powerManage method from the Java classes as long as both MethodChannels in the Flutter and Java code have been set up correctly and have the same name (Mwiti, 2020). Using these MethodChannels the Java methods for Alize can be accessed by the flutter application. This could be implemented with a custom package developed for the project, or with a handler class. This is by far the simplest integration and could be completed within a matter of three to four months to fully implement and test. However, it would only be compatible with Android devices.

6. Conclusion

Speaker diarization is a complex system of algorithms, segmenting audio files to be able to layer separate speakers. While the algorithms have very recently been optimized, it is because of AI that the accuracy can be improved through neural networks and spectral clustering. By moving the computing from workhorse computers to online services, applications are now allowed to be able to create lightweight solutions that still have the capability of diarization. Mobile phones currently do not have the local processing power to handle the large calculations required to have accurate speech diarization, and provide space for other applications and storage needs. If these issues are not a concern, it is possible to create systems of diarization through some examples like pyAudioAnalysis, Kaldi, and Alize. PyAudioAnalysis library can be integrated into a unique python solution that is then wrapped to work on either IOS or Android. This solution is only as accurate as the computations that are done in the developed solution. Kaldi does not formally have an IOS package, but for android, it can be used through command line analysis from the application and application side speaker identification. Finally, Alize has the most versatility as an offline solution. Alize comes with a toolkit for android and easier to implement into IOS because of its C++ native language. It also requires a lot of set up to improve accuracy and like all the other toolkits, requires further development if the application is not written in the same language as the diarization solution.To improve security and reduce streaming needs a possible partial offline and online solution can be looked at to see benefits and impact on mobile devices.

References

Azure. (2021). Speaker recognition quickstart - speech service - azure cognitive services. Speaker Recognition quickstart - Speech service - Azure Cognitive Services | Microsoft Docs. https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/get-started-speaker-recognition?tabs=script&pivots=programming-language-curl.

Chaquopy, (2021, April 16). chaquopy: 0.0.9 documentation. Retrieved from https://pub.dev/packages/chaquopy

Flutter Kaldi ASR Plugin, (2021, July 13). flutter_kaldi_asr_plugin documentation. Retrieved https://github.com/nmfisher/flutter_kaldi_asr_plugin

Giannakopoulos, T. (2021, July). tyiannak/pyAudioAnalysis: Python audio Analysis Library: Feature EXTRACTION, CLASSIFICATION, segmentation and applications. GitHub. https://github.com/tyiannak/pyAudioAnalysis.

Google. (n.d.). Separating different speakers in an audio recording. Google. https://cloud.google.com/speech-to-text/docs/multiple-voices.

IBM. (2021, April 13). Audio transmission and timeouts. cloud.ibm.com. Audio transmission and timeouts.

Khudanpur, S., Povey, D., & Trmal, J. (2016, June 13). Building Speech Recognition Systems with the Kaldi Toolkit . www.clsp.jhu.edu. https://www.clsp.jhu.edu/wp-content/uploads/2016/06/Building-Speech-Recognition-Systems-with-the-Kaldi-Toolkit.pdf.

Mane, A., Bhopale, J., Motghare, R., & Chimurkar, P. (2020). An overview of speaker recognition and implementation of speaker diarization with transcription. International Journal of Computer Applications, 175(31), 1–6. https://doi.org/10.5120/ijca2020920867

Mwiti, D., (2020, February 5). Writing Native Java Code in Flutter for Android. Retrieved from https://heartbeat.fritz.ai/writing-native-java-code-in-flutter-for-android-6a35ada7e915

OpenVINO. (2021). OpenVINO™ Toolkit Overview. docs.openvinotoolkit.org. https://docs.openvinotoolkit.org/latest/index.html.

Ramon, Y. (2021). YoavRamon/awesome-kaldi: This is a list of FEATURES, SCRIPTS, blogs and resources for better using KALDI (HTTP://KALDI-ASR.ORG/). GitHub. https://github.com/YoavRamon/awesome-kaldi.

Starflut, (2021, February 5). starflut 1.0.0 documentation. Retrieved from https://pub.dev/packages/starflut

Sharafeddin, H., Sharafeddin, M., & Akkary, H. (2015). Voice verification system for mobile devices based On alize/lia_ral. Proceedings of the International Conference on Pattern Recognition Applications and Methods. https://doi.org/10.5220/0005218902480255

Tranter, S. E., & Reynolds, D. A. (2006). An overview of automatic speaker diarization systems. IEEE Transactions on Audio, Speech and Language Processing, 14(5), 1557–1565. https://doi.org/10.1109/tasl.2006.878256