Document to speech conversion

KUNAL DILIP MANSUKHANI[1], RUCHI MANIKRAO DHORE[1], SANFER SAMSON NORONHA[1], AND STUTI SANJAY JANI[1]

[1] *kmansukhani@scu.edu*
[2] *rdhore@scu.edu*
[3] *snoronha@scu.edu*
[4] *sjani@scu.edu*

Compiled December 8, 2022

The written word provides access to a vast universe of knowledge, fantasy, and creativity. However, those with sight problems and learning difficulties could find it challenging to read digitized books and documents. Moreover, some individuals choose listening to text-based material rather than reading it. By giving written material a voice, a text to speech solution widens the audience for digital information. We have developed a straightforward web application that leverages Amazon Web Services's Machine learning Services to turn text into speech. The program has an user interface which accepts a document and translates it into audio files that can be played from a web browser.

1. INTRODUCTION

Due to the high cost of maintaining the infrastructure, huge servers, specialist hardware (such as GPUs), and dedicated operations teams that are needed for speech recognition, it is difficult to have a personal setup running for such an operation. Unfortunately, current Speech recording engines are also too heavy to operate inside a Lambda function. Such an issue can be resolved with computing by using a web application using an Amazon S3 bucket that will receive a user-provided document as an input and use Amazon Polly to turn it into an audio file before storing it in the Amazon S3 bucket and making it available to the user as a .mp3 file. This is different from other regular speech processing applications since this gives the user easy change and deployment capabilities. For simple deployments and consistent results across environments, we have containerized both the front end and the back end. This is the key idea of project, event-driven architecture that converts text from scanned documents into speech. Using a cell phone or other portable device, the document-to-speech technology instantly turns scanned books or papers into speech. The benefits of the solution include increased learning potential for young readers as well as user mobility and flexibility. Another reason is warnings and dosage instructions are frequently printed in small, difficult-to-read typefaces. To prevent potential harm, we might use this method to snap a photo, translate it to voice, and listen to the instructions. This paper describes the architecture, implementation and results of such conversion of document to speech. The application has a straightforward user interface and accepts documents in a few languages before translating it into audio files that can be played from a web browser. One can use any sort of text, but we'll utilize a sample pdf file. For instance, you can read books or news articles while you're driving or riding a bike, or recipes while you're preparing a meal.

2. BACKGROUND AND RELATED WORK

The article on the Amazon Web Services blog [1] discusses how to build text-to-speech applications using Amazon Polly, a service that uses advanced deep learning technologies to synthesize speech that sounds like a human voice. The article provides an overview of the features and capabilities of Amazon Polly, including the ability to create lifelike voices in multiple languages, customize speech output with different styles and emotions, and integrate speech synthesis into a wide range of applications and devices. The article also includes step-by-step instructions for using Amazon Polly to build a simple text-to-speech application, as well as tips for optimizing speech output and managing costs.

The research paper published by Kumari, A et.al [2] presents a portable text-to-speech converter for visually impaired individuals. The device combines optical character recognition (OCR) and text-to-speech synthesis (TTS) technology to convert text from printed or electronic sources into speech in real-time. The device is small and lightweight, making it easy for visually impaired individuals to carry and use. The paper discusses the design and implementation of the device, as well as the performance and accuracy of the OCR and TTS components. The authors also compare the device to other portable text-to-speech converters and discuss its potential applications and benefits for visually impaired individuals.

The research paper published by Zhang, X [3] discusses a method for improving the accuracy and speed of optical character recognition (OCR) systems. The proposed method uses a combination of image preprocessing and machine learning

algorithms to automatically identify and correct errors in OCR output. The authors evaluate the performance of the proposed method using a dataset of scanned documents and compare it to existing OCR error correction methods. The results show that the proposed method significantly improves the accuracy and speed of OCR systems, making them more reliable and efficient for a wide range of applications.

The article on the Amazon Web Services blog [4] discusses how AI-powered speech recognition technology can be used to create interactive learning experiences for children and adults. The article describes how Amazon Transcribe, a cloud-based speech recognition service, can be used to transcribe and analyze spoken language in real-time, enabling the creation of educational apps and games that respond to spoken input. The article also includes examples of how Amazon Transcribe has been used in educational settings, including a game that teaches children about animal sounds and a voice-enabled vocabulary trainer for adults. The article concludes by highlighting the potential of speech recognition technology to enhance the effectiveness and accessibility of learning experiences.

For most of these works, there wasn't a scalable and easily deployable functionality to demonstrate a text to speech application. Some used hardware implementations (IAAS) which demonstrated a limited scope of reach, for others using software implementation (PAAS), the application was not scalable and easily deployable. Hence, we have created a containerized application to which components can be added later if further improvements, like the ones mentioned at the end of this report, are needed. The solution quickly sets up document uploading and turning scanned documents into voice. For the backend to continue transmitting real-time progress updates to the user, we employ a listener that detects if there is a document recently uploaded. For this we have a single lambda function. Amazon Textract, Amazon Translate and Amazon Polly are started using a AWS Lambda function. Hence this project is a natural progression from the already present text to speech applications.

Virtual Machine is being used in the form of Amazon EC2 instances. It enables capacity scalability in accordance with computational needs. A public cloud is reliable, and the application is robust to failure, it also gives system administrators and developers a solution from failure situations. Nowadays, cloud computing, and notably infrastructure as a service, is widely used by businesses. Public cloud instances enable capacity scalability in accordance with computational needs. It has solid uptime and the application is robust to failure, hence Amazon EC2 gives system administrators and developers isolation from failure situations.

Here, we are using a FaaS solution called Amazon Lambda. It executes our code on a high-availability infrastructure and manages the server and operating system upkeep, capacity provisioning, and automated scaling of the compute resources. Since it's a Function as a service, this application can be managed without the complexity of creating and maintaining the infrastructure typically associated with creating and releasing an application. We are using an architecture which extends the same reason above, to distance the complexity of the architecture from the user.

AWS CloudWatch enabled us to monitor execution of an Amazon Lambda function and use event data to take automated actions and reduce time of resolution. This freed up important resources and allowed us to focus on building applications and business value.

A conscious design decision, especially given the current growth in demand for serverless database designs and one of the reasons for choosing a public cloud like Amazon Web Services was that it can provide performance at scale. DynamoDB being an Amazon service itself can utilize a variety of other AWS services, which will undoubtedly set itself apart from an expanding number of database providers. To keep track of documents converted, the application employs a Amazon DynamoDB table.

We create an IAM role for the lambda function before we start writing these functions. Which AWS services (APIs) the functions can communicate with are specified by the role.

To store our files, Amazon S3 offers web interfaces. Our data files, including picture files like JPG, PNG, and pdf files, are kept in buckets as objects. Buckets are used to hold items in place. Amazon S3 buckets store items, such as data files, in the same way that regular household buckets store water. One can easily retrieve our data and have it protected by using S3.

Thus, this project can be a stepping-stone Implementing a document-to-speech system on other platforms, such as telephony systems, ATMs, video games, and any other platforms where text-to-speech technology would be a benefit and boost functionality, this would be another area where this can be implemented. User authentication can be added, support for more human languages, and a queue for delaying incoming requests.

Beyond the limitations of traditional optical character recognition (OCR), Textract employs machine learning technology to automatically detect, comprehend, and extract data from forms and tables, and in our case the document. We also went one step further to use the Amazon Translate API to first convert our document to three desired languages and then use Polly to convert the translated text to audio file.

3. APPROACH

The objective of the proposed work is to convert text from an input document into a human-like audio file. The conversion takes place in several steps. The developed system follows a client-server architecture. This architecture allows client(s) to request and receive service from a server. The architecture of the solution is designed keeping in mind the current trends in cloud technologies and the ability to use services provided by the cloud platforms.

The high-level architecture of the end-to-end system is shown in Fig.1 which completely binds the front-end of the system interaction with the back-end and AWS cloud.

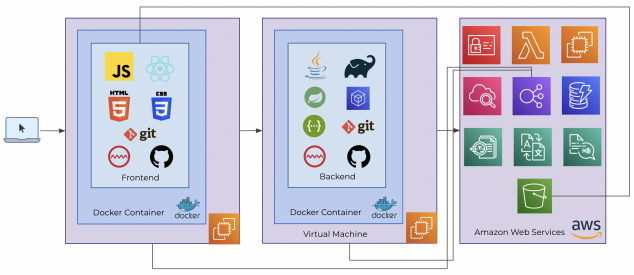

Fig. 1. System Architecture

The front-end web application provides two main functionalities to the user. The first feature enables the user to upload the desired source document and allows to play a converted audio file on the fly as a stream from AWS S3 bucket. To enhance the user experience, the user has a choice to play the audio in

three different languages namely English(US), Hindi and Chinese languages. The second feature provides an option to the user to download the audio file for selected language. The offline functionality provided by the download option enhances the user experience and allows for anytime speech listening. The dynamic front-end web application has been written using a JavaScript library ReactJS and Axios is used as HTTP client. Both the web application and the back-end has been containerized using Docker running on two different Amazon EC2 instances hence making the application more scalable. This also comes with the advantage of easy deployment and eliminates initial hardware cost. Fig.2 shows overall view of Amazon EC2 dashboard.

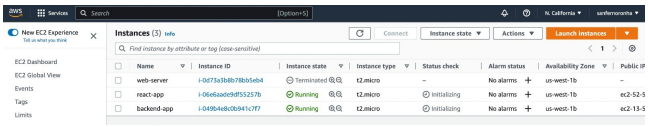

Fig. 2. Amazon EC2 Dashboard

The communication between front-end and back-end is completed by utilizing HTTP API endpoints. The back-end application provides three HTTP API endpoints to provide three different functionalities to the front-end. The first functionality involves uploading the front-end-received input document to a source S3 bucket. The second functionality involves determining whether a specific input document is present in the source S3 bucket. Downloading an audio file from the final S3 bucket is the third functionality. The back-end application has been written using Java along with Spring Boot web framework. Fig. 3 demonstrates the class diagram for the same.

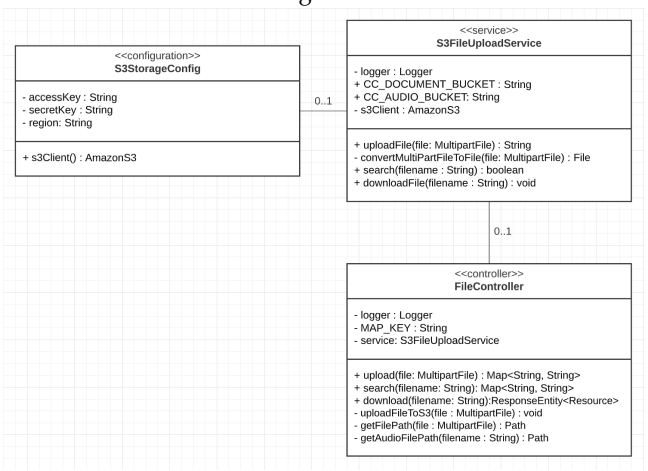

Fig. 3. Class Diagram

Amazon Web Services offers scalable, reliable and inexpensive cloud computing services. These services are mainly provided in areas like Analytics, Compute, Database, Developer tools, Machine learning, Management and Governance, Networking and Content Delivery, Serverless etc. AWS provides diverse artificial intelligence services which can be used to extract text, translate text from one language to another and convert text to human-like speech voice.

Amazon Identity and Access Management (IAM) provisions role access to AWS Lambda. Policies with full access privilege for AWS services Lambda, S3, Textract, Translate, Polly, DynamoDB are attached to this role. Fig.4 shows overall view of AWS IAM Console.

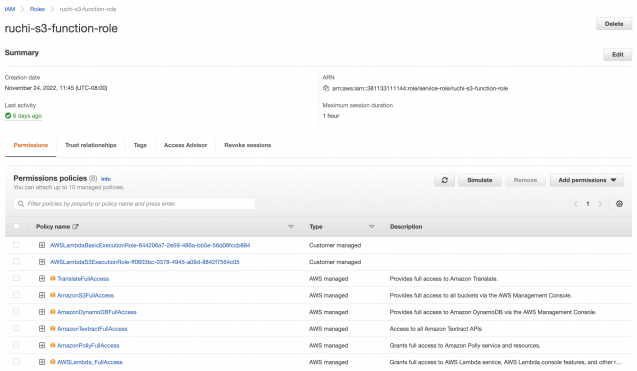

Fig. 4. AWS IAM Console

The ideal choice to store diverse files like PDF, JPG, PNG, MPEG is Amazon Simple Storage Service (Amazon S3). Amazon S3 bucket is a container to store objects. Two separate buckets, one for source 'cc-documents-bucket' and one for destination 'cc-audio-bucket' are created. Fig.5 shows overall view of S3 buckets.

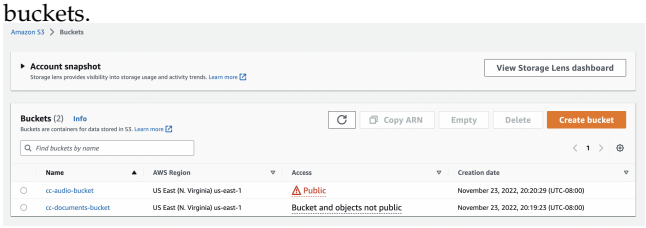

Fig. 5. AWS S3 buckets

Amazon Lambda is a serverless, event-driven compute service that lets us run code for any type of application or backend service without managing servers. It helped accomplish the required calls and processing using clients like Textract, Translate, Polly and DynamoDB. An input trigger was set up on this Lambda on source S3 bucket.

Amazon Textract was leveraged to extract text blocks from an input document. The service supports four document extensions as of now - PDF, JPG, PNG and TIFF. It helps to extract printed text, forms and tables in English, German, French, Spanish, Italian and Portuguese languages. For the current solution, PDF, JPG and PNG file extensions are supported which consists of text in English.

Language Translation is offered through Amazon Translate. The extracted text blocks are translated to Hindi and Chinese.

Amazon Polly generates human-like speech from audio response for any given text. For English, the voice ID used was Joanna. For Hindi, the voice ID used was Kajal and for Chinese (Mandarin), the voice ID used was Zhiyu.

A table called Logs is created in Amazon DynamoDB to contain conversion logs, which show which source file was converted to which destination file.

AWS Lambda execution logs are uploaded to AWS CloudWatch in order to speed-up the development process. It offers real-time execution logs, which makes it easier to identify mistakes right away.

At some point, when the application needs to be scaled up and there are more incoming requests, a load balancing mechanism is needed. Amazon Elastic Load Balancer serves this purpose by

balancing the load between multiple backend containers. Software Architecture describes the fundamental structures of a software system. This is shown in Fig.6 Software Architecture

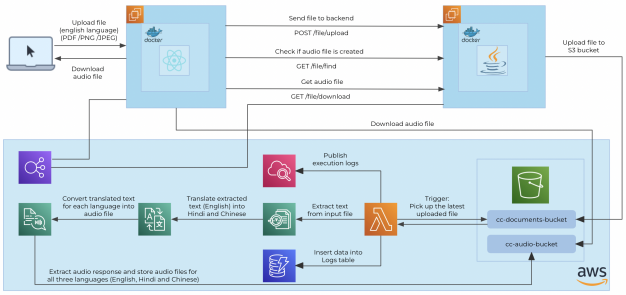

Fig. 6. Software Architecture

The overall process is carried out as follows:
Step 1: Upload input document of choice (PDF/JPG/PNG) using upload input box

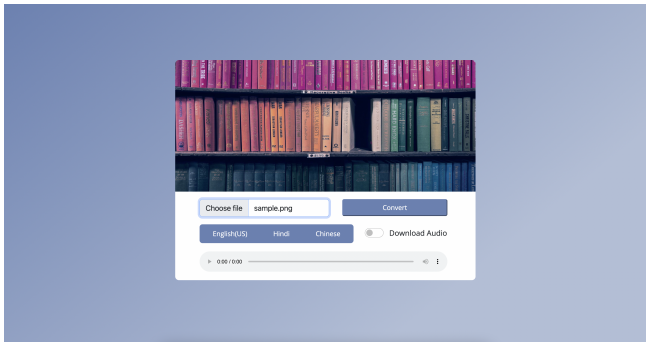

Fig. 7. User Interface

Step 2: Click on Convert button which will call HTTP API POST endpoint /file/upload responsible to upload the document to source Amazon S3 bucket 'cc-documents-bucket'
Step 3: Trigger on Amazon Lambda function detects upload event and gets the object from the event
Step 4: Text blocks is extracted from the input object using Textract service
Step 5: Text blocks are concatenated to form a single text block
Step 6: Text block is translated to Hindi language using Amazon Translate service and stored in hindi text block
Step 7: Text block is translated to Chinese language using Amazon Translate service and stored in chinese text block
Step 8: English text block is converted to audio English (US) using Amazon Polly
Step 9: Hindi text block is converted to audio Hindi using Amazon Polly
Step 10: Chinese text block is converted to audio Chinese (Mandarin) using Amazon Polly
Step 11: Store English (US) audio response as file in destination Amazon S3 bucket 'cc-audio-bucket'
Step 12: Store Hindi audio response as file in destination Amazon S3 bucket 'cc-audio-bucket'
Step 13: Store Chinese audio response as file in destination Amazon S3 bucket 'cc-audio-bucket'
Step 14: Store file conversion logs in Amazon DynamoDB
Step 15: User can click check/uncheck download switch icon
Step 16: User can select the target language for audio file among English (US), Hindi and Chinese (Mandarin)
Step 17: Check if the audio file for selected language is generated in destination Amazon S3 bucket 'cc-audio-bucket' using HTTP API GET Endpoint /file/find/
Step 18: If file is present, stream the audio file from the destination Amazon S3 bucket in the audio player on the frontend

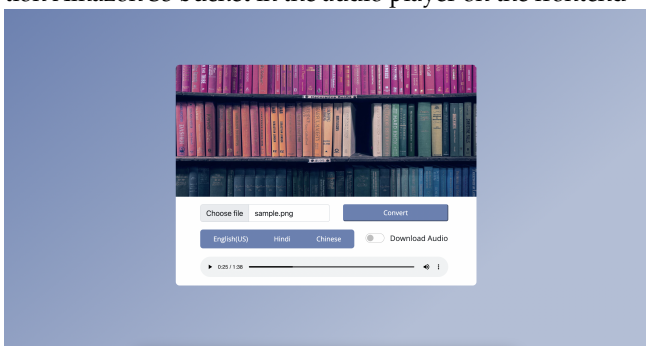

Fig. 8. Play Audio

Step 19: If download switch is on, call the HTTP API GET Endpoint /file/download to get the audio file from the destination S3 bucket and store the audio file in the downloads folder

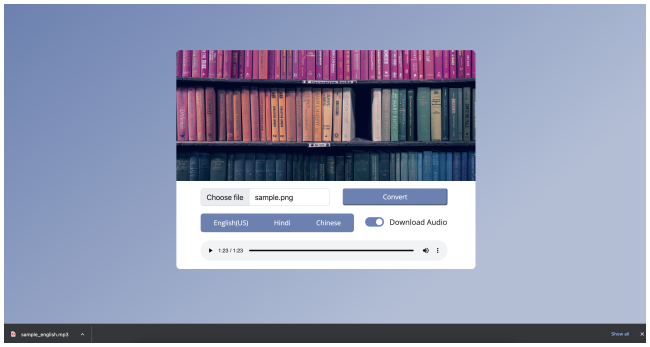

Fig. 9. Download functionality

Swagger 2 was used for API documentation maintenance. To catch and fix coding issues at early stages, sonarlint plugins were utilized in both front-end and back-end IDE. Fig. 10 demonstrates the Swagger UI.

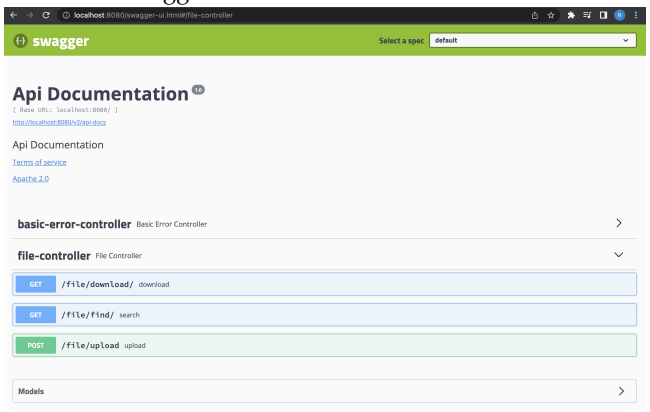

Fig. 10. Swagger UI

4. OUTCOME

Expected outcome for this project was successfully converting text to speech using amazon web service's machine learning

services to aid people with sight problems or learning difficulties. This model has achieved the conversion to speech successfully.
Runtime details: Both the front-end and back-end codebase is dockerized and pushed to dockerhub.

```
sudo docker build .
sudo docker tag <id> <dockerhubID>/<repo>:<tag>
sudo docker push <dockerhubID>/<repo>:<tag>
```

In order to run our project, we boot a 2 EC2 instances and pull our dockerhub containers onto the remote instances. Next, we run our docker containers in detatched mode so that the process stays active in the background.

```
sudo docker run –d –p 80:30000
<dockerhubID>/<repo>:<tag>
```

The projects are also configurable to run in non-detatched mode for the purpose of debugging. Here are the observations for runtime performance of our EC2 instances.
Server Performance:

Compute	Value
Family	t2 micro
vCPU	1
Memory (GiB)	1.0
Memory per vCPU (GiB)	1.0
Clock Speed (GHz)	3
Memory (GiB)	1.0
Average Response time	190ms

Table 1. Server Performance

Lambda Performance:

Compute	Value
Type	png
Duration	12716.58 ms
Billed Duration	12717 ms
Memory Size	128 MB Max
Memory Used	87 MB
Init Duration	510.06 ms

Table 2. Lambda Performance - png

Compute	Value
Type	pdf
Duration	12690.92 ms
Billed Duration	12691 ms
Memory Size	128 MB Max
Memory Used	89 MB

Table 3. Lambda Performance - png

Compute	Value
Type	jpg
Duration	12255.73 ms
Billed Duration	12256 ms
Memory Size	128 MB Max
Memory Used	90 MB

Table 4. Lambda Performance - png

Endpoint response times: POST /file/upload - 800 ms
GET /file/find/filename - 500 ms
GET /file/download/filename - 900 ms

Memory: React in dev mode (npm build) compiled to around 4MB including node modules. The Spring boot JAR file comes down to around 20 MB. Once the user creates audio files,they will be saved on S3. The size for the audio files is respective to the user. We have added various functionalities to the project to make it more user friendly like : download feature to save audio file on local machine, easy user interface to allow user to select language, slider functionality to traverse the audio file.The audio files can be accessed through the S3 bucket as well at the backend.The engaging and user-friendly UI will ensure that the user is able to convert text to speech efficiently.
Cloudwatch metrics

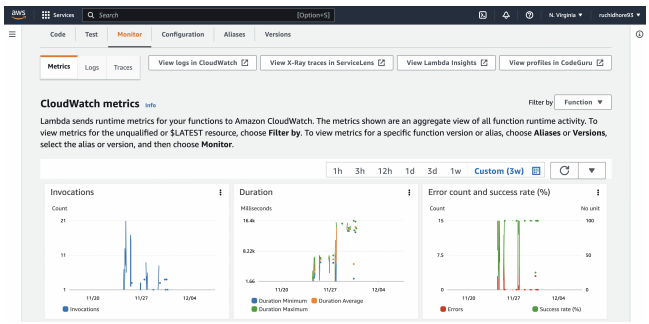

Fig. 11. CloudWatch Metrics

5. ANALYSIS AND FUTURE WORK

The model currently supports only a limited number of languages as input and the output is available only in three languages. This creates saturation for users. Currently, the facilities to store and access converted audios are also not provided to the users.
Future advancements in proposed work are expected. The addition of user authentication features like login and logout of the user would be the first potential future task. A feature would be enabled which will allow the user to store and access previous converted audio files. It is expected to overcome the limitation of input languages. The output range of languages could be increased from 3 to 36 languages as AWS provides support for 36 languages. Addition of Continuous Integration/Continuous Deployment (CI/CD) pipeline can bring further enhancements in the work. The project here is based in AWS, similarly it can be applied to various cloud providers like Google Cloud, Azure using Terraform.
Moreover, the entire code-base can be packaged into reusable

code and be used in a lot of applications and projects related to education, medicine, tourism, etc. It can also be turned in a public API project where developers can make use of the document to multi-lingual text to speech feature for their own projects.

6. CONCLUSION

In this project, we demonstrated a solution which is a document-to-speech conversion application by using two virtual machines handling two containers (frontend and backend) that are exchanging HTTP requests with each other. The idea behind containerizing is that each machine can run independent of the operating system it is run on. Since a container encapsulates all the dependencies of the application, it can run on virtually any operating system.
One way that AWS helped with deploying a document-to-speech conversion app is through the use of Amazon Elastic Compute Cloud (EC2) instances. EC2 allowed us to launch virtual servers on which we can installed and ran our app, providing the computational power and storage necessary to handle the demands of our document-to-speech app. Additionally, we used Amazon Textract, Lambda and Polly for generating the speech output, and Amazon S3 for storing and serving the resulting audio files.

Some of the primary limitations of our work include:
1. I live in Las Vegas, which is pronounced differently from the way it is written. vs. This speech is being streamed live from Las Vegas.
2. Normalization of text. Defining acronyms, abbreviations, and units: St., which can be expanded to mean either street or saint.
3. Deja vu, legitimate names like François Hollande, slang like LOL, etc. are examples of foreign words.
4. Audio conversion of words succeeded by exclamation point are not converted properly into native language
5. Language limitation, this app will work with languages supported by AWS Polly.

7. REFERENCES

[1] Amazon Web Services (AWS) (n.d.). AI-powered speech recognition creating interactive learning experiences for children and adults. AWS Public Sector Blog. Retrieved from https://aws.amazon.com/blogs/publicsector/ai-powered-speech-recognition-creating-interactive-learning-experiences-children-adults/

[2] Kumari, A., and A. K. Singh. 2017. "Enhanced Portable Text-To-Speech Converter for Visually Impaired." ResearchGate. doi:10.13140/RG.2.2.14507.86874.

[3] Zhang, X., Tang, X. (2018). Improving the accuracy and speed of optical character recognition systems using machine learning. IEEE Transactions on Pattern Analysis and Machine Intelligence, 40(9), 2165-2177. https://doi.org/10.1109/TPAMI.2018.2813596

[4] Amazon Web Services (AWS) (n.d.). Build your own text-to-speech applications with Amazon Polly. AWS Machine Learning Blog. Retrieved from https://aws.amazon.com/blogs/machine-learning/build-your-own-text-to-speech-applications-with-amazon-polly/

Repository Link https://github.com/ruchidhore12/document-to-speech-conversion Commit ID : 89c9cabda335c8799615fc0889c15d75a7c9ff7a