Boogle Translate

CS701 Final Project Report

Joshua Ravishankar
Middlebury College
Middlebury, VT
jravishankar@middlebury.edu

Zale Young
Middlebury College
Middlebury, VT
zyoung@middlebury.edu

ABSTRACT

With vast amounts of data becoming readily available, there have been recent leaps and bounds in the field of machine learning, particularly sequence-to-sequence learning. This has enabled the development of powerful algorithms that can perform highly accurate language translation. Despite this fact, there is a lack of software that makes use of these advancements to enable human-to-human, inter-lingual communication.

To fill this void, we have created Boogle Translate, a hybrid, mobile chat application that allows for near real time communication between two individuals operating in different languages. Our application not only supports text messaging, but also audio messaging through the use of a dynamic back-end pipeline that performs speech recognition and synthesis (along with normal parallel text translation). Our goal with this application was to consolidate existing language processing software to build a communication platform that is cheaply accessible to as many people around the world as possible.

Keywords

CS701; LaTeX; natural language processing; sequential learning models; speech translation; inter-lingual communication;

1. INTRODUCTION

In the present day, even though machine translation is not error-free, there exists highly accurate pieces of software such as Google Translate that have closed in on solving the problem [15]. Despite that, there still does not exist a widespread and seamless method for two people who don't share a common language to communicate with each other. This problem is very relevant today because, with the advent of improved transportation and increased communication due to the internet, the world has become a much smaller place. There are more people interacting across the globe than ever before. This means that there are more people communicating who do not speak the same language, giving rise to communication barriers and room for misunderstanding. It is therefore our duty as software engineers to provide technology that aids in bridging these gaps in communication.

Creating automated language processing systems necessitates the need for computers to act as intermediaries to inter-lingual human communication. This means that computers must be able to interpret human communication and also be able to communicate back in a meaningful way. Our research is not intended to directly contribute to the technical problem of Natural Language Processing (NLP). Big players such as Google already dominate the field of NLP state-of-the-art systems like Google Translate, Google Voice and Google Speech-To-Text through access . The problem is that these technologies are not designed for human-to-human communication. This creates the need for a more user-centered approach, focusing on communication, rather than improving technical translation models. It was the goal of our research, therefore, to explore an application-based method which aggregates existing state-of-the-art language and audio processing technologies in order to create a trans-lingual communication platform.

In this paper, we will begin by using the Background section to describe the problem that we are trying to address in depth and present information that is relevant to our development. We will then briefly mention similar applications that try to tackle this problem in the Related Work section. A Methods section will follow, outlining the approach we used in the undertaking of this project. After that, we present our findings in the Results section followed by a Discussion section where we explore insights gained by our results. Acknowledgements and References then follow.

2. BACKGROUND

Language is a barrier to communication. When two people don't have a known language in common, there exists no way for them to converse. Recently, because of developments in machine learning catalyzed by big data, language translation algorithms have become highly accurate. Software such as Google Translate has the ability to take text and audio in one language and unpack the meaning stored within those media in another language. This provides a very useful information dissemination tool, but doesn't make communication any less nuanced. To communicate with someone who doesn't share a common language, one must consult translation tools, usually in the form of either a living translator or an application that enables access to the aforementioned algorithms. This creates a disconnect, making inter-lingual communication feel like a game of cryptography rather than honest conversation.

With the ever-growing prevalence of mobile devices, instant messaging has become a common medium for communication. The inherently digital aspect of instant messaging allows for many different adaptations, including inter-lingual messaging. By incorporating existing speech recognition, language translation, and speech synthesis software,

we aimed to create an instant messaging platform that would enable seamless communication between users who don't share a common language, without requiring any extra effort on the user's end to translate information.

2.0.1 *Language Translation*

The field of machine translation has seen huge advancements thanks to deep learning, which is a form of machine learning that involves the use of many hidden layers between the input and output layers (hence a 'deep' network). One tool that has aided significantly in the process of machine translation is Recurrent Neural Networks (RNNs). An RNN is a special type of neural network that lends itself to processing sequential data through its use of stored state and cyclic information flow. Rather than working with each layer in the network individually, RNNs use information from previous layers as input to influence their decision making. This makes them the perfect tool to work with language processing, since language is inherently sequential; the meaning of a sentence as a whole is dependent on more than the sum of meanings carried by the individual words within it [13].

2.0.2 *Speech Recognition*

Speech recognition is the process of taking spoken language and transforming it into text. Modern speech recognition systems rely on Hidden Markov Models (HMMs), which take in audio by fragmenting inputs into small segments that can be processed as static vector data. These vectors are matched to phonemes, which are units of sound that allow words to be distinguished within languages. From here, the HMM performs statistical modeling to determine which word would be most likely associated with a given sequence of phonemes [9].

Often, there is pre-processing done on the inputted audio signals. Neural networks simplify the data by using feature transformation and dimensionality reduction, techniques which are typically used to pre-process data in machine learning. Voice activity detectors are used to cut the input signal down to just the portions in which speech is likely contained. Ambient noise adjustment is also used in the process to help increase accuracy of text classification [9].

2.0.3 *Speech Synthesis*

Speech synthesis is the process of taking text and transforming it into spoken language. In a similar fashion to speech recognition, speech synthesis systems use HMMs and neural networks to predict the most likely speech pattern associated with a given text. This is done in a backwards fashion to speech recognition, where text is pieced into words, and then the words are transformed into phonemes which are then read aloud by the computer (either through concatenating pre-recorded phonemes or synthesizing sound frequencies) [16].

3. RELATED WORK

Much of the proprietary language translation software that is available today doesn't directly aid with communication. Applications like Google Translate have high accuracy and speed, but serve only to make information in foreign languages interpretable, and would be clunky and annoying to use when trying to use it for messaging purposes.

Figure 1: iTranslate in action

Attempts to create more communication-oriented applications have made significant progress towards an interlingual messaging application, but have failed to incorporate widespread availability as a core goal. iTranslate is an iOS exclusive language translation application that focuses on learning and understanding as its goal (see figure 1). It has a feature that allows messages (specifically those done through iMessage) to be translated and saved in both the original and translated language. This creates an user experience that is similar to the one we attempted to manufacture in making our application. However, the iTranslate implementation fails to enable widespread communication. This is because the messages must be sent through iMessage to have the translation functionality, the audience for this feature is restricted to iOS users. Furthermore, by allowing this functionality only within iMessage, it limits the potential receivers of the message. Not only must both users have iOS devices, but they must also have already exchanged contact information. Rather than breaking barriers and crossing borders, this functionality only smooth communication that would have already been prevalent [7].

With Boogle Translate, we aim to reach a much wider audience and make communication the key functionality of our software. Rather than trying to make information dissemination easier, we aim to remove the dissemination process from the user's control and leave them with the same familiar messaging interface that is prominently used.

4. METHODS

Boogle Translate was created by consolidating existing proprietary software and incorporating it into a messaging application.

4.1 3-Step Pipeline

We required methods to translate text and audio mes-

sages. The problem of translating text was made trivial by the Google Translate API, which takes text in a specified input language and a target language as input, and returns the text translated into the target language as a response.

Translating audio was a slightly more nuanced process. To translate audio, we had to build a pipeline that would break down the process into tasks that were manageable by APIs that were easy to access and quick to respond. We implemented a 3-step pipeline that would first turn audio into text, then translate that text into the target language, and finally take the translated text and synthesize it into speech audio.

We implemented this pipeline using Python and several proprietary packages.

4.1.1 Speech Recognition

Speech recognition was done using Python's Speech Recognition package, which enables quick and easy access Google's Web Speech API. The package requires only a few lines to take in an audio file and an input language, and with one API call it returns a string containing the recognized text [9].

4.1.2 Text Translation

Text translation was simply handled using the Google Translate API that is available when developing with Python. All it required was input text, an input language, and a target language [14].

4.1.3 Text-to-Speech

Speech synthesis was implemented using Google Cloud Text-to-Speech API. This API uses Google's WaveNet to synthesize high-fidelity audio in a target language with various voice options [11].

4.2 Creating the Back-end

4.2.1 API

After having a working translation pipeline which takes in audio and outputs the translated audio, we needed a way to make this pipeline accessible from anywhere. Therefore we decided to expose this functionality via a back-end server.

The first aspect of our server is the API. For simplicity and flexibility, we decided to build the server in Javascript. We used Node and the Express.js framework to create an API server. Through Express.js, we exposed a /translate route which expects a POST request containing the Audio, the Input Language and the desired Output Language as form data. The API then responds with the translated audio[5].

In between the request to the API and the API's response, we needed to do two things:

1. Process the Audio, converting it to a format that the Python program can accept.

2. Run our Python Pipeline on the input and obtain it's response.

Firstly, since our python program only accepts .wav or .flac audio files, it had to be converted into one of those formats in the processing step. In order to process the audio, we used a freely available C program called ffmpeg. After the request data is received from the POST request, our Javascript code spawns a child process which runs ffmpeg, passing in the input audio as an argument. Ffmpeg then converts the audio to a FLAC file, returning it to the Node process[2].

After ffmpeg returns, the Node process spawns another child which runs the Python Pipeline, passing in the .flac audio file, the input language and the output language. This process returns the translated audio to the Node process and Express sends this audio file as a response.

Creating an API server allowed us to separate the translation service from our intended communication service. This allows flexibility in the way we create our user-facing communication service since all we would need to do is plug it in to the API server by sending a POST request[4].

4.2.2 Sockets

After connecting the frontend to our API, we noticed a significant lag. Much of this lag was due to the spawning and processing of the Python program. However, in an attempt to speed communication between client and server, we decided to use Sockets instead of HTTP requests. Communication over sockets is usually faster than HTTP communication and has less overhead sent with each packet of data. So we integrated Socket.io, a Javascript library for socket communication.

We implemented Socket.io on both the client and the server. When the server starts, it immediately begins listening for connected clients. When the client starts, it immediately connects to the server and that connection lasts for the entire time that the client app is active. The client then sends text and audio messages over that socket to the server. The server then sends the translation back to the client over the socket[8].

An interesting challenge that we encountered was sending audio over the sockets. It proved difficult to implement streaming over sockets along with Expo, the main React Native framework we used. In lieu of streaming, we decided to send the audio as a base 64 string over the socket. When the client records audio, it is saved to a file and the contents of that file are read into a base 64 encoded string. When the server receives that string, it decodes the string and writes the binary contents to a file which will be interpreted as an audio file. The same thing is done in the reverse direction when the server sends the translation back to the client.

4.3 Front-End Design

4.3.1 Application Functionality

For the front-end, we decided that a messaging platform which allowed both text and voice messages would meet our goal. This application was designed to behave just as any other messaging platform, except that a user would type or record a message in one language, and the other user would receive the message in another language. This abstracts away any translation functionality from the user, leaving the user to interface with our application just as they would with any other messaging application.

Having created the API server, we were able to think about the front-end deliverable independently of the back-end logic. We initially entertained the idea of creating both a web and a mobile application. However, since the goal of this project was to facilitate seamless communication, and given that most people these days communicate via mobile

messaging and chat platforms, we decided that a mobile application would be best suited for this goal.

4.3.2 *Development Frameworks*

After deciding on mobile, the next matter of consideration was implementation methods and details. In order for this application to be viable, it needed to be accessible by as many people as possible. Therefore, to reach both iOS and Android users, we decided to use the React Native framework, which allows developers to deploy to both of these platforms using the same fundamental Javascript code, as opposed to a version in Java for Android and one in Objective-C/Swift for iOS[6].

Additionally, we wanted to focus on creating a seamless user interface and user experience. It was therefore necessary to abstract away the implementation details of React Native which were specific to Android or iOS. Therefore, in addition to React Native, we used Expo, a set of tools and libraries which sit on top of React Native and allowed us to use the same code base for both platforms. Expo also allowed us to create a deployment environment which rapidly integrated changes to the code base making it possible to quickly test our code as we developed [1].

In order to record audio, we used the Expo Audio API. This API enabled us to create a recording, record audio and save the audio to a temporary location in the device. We then package this audio, along with the input and output language in a data object to be sent to the back-end server. The server then processes the audio and sends it back to the client side.

4.3.3 *User Interface*

We used React Native's Gifted Chat package to smoothly integrate both text and audio messaging in a familiar, approachable user interface. Gifted Chat allowed us to focus on the translation aspect of our application without worrying about how messages/chats would be stored or displayed. A Gifted Chat conversation is fundamentally held as an array of message objects with information such as sender id and photos contained within. In other words, we just needed to use our back-end to produce an array on the client side that would match the user selected language[10].

4.4 Database

We used Google Firebase to integrate a database that stores user accounts, chat rooms, and messages. We chose Google Firebase because it allowed us to integrate user accounts with other proprietary services (rather than trying to create our own) and it was simpler to merge with our existing Expo project [3].

4.4.1 *Users*

To incorporate user accounts, we used Facebook Login to collect basic information and generate unique ids, and then stored these along with a user's native language within the Firebase database. We then used Firebase for account authentication, which enabled features such as automatic login.

We used Facebook accounts to build up our user base because Facebook already has an extremely wide user base, and is well-known for connecting people worldwide. This made it seem like the natural choice to go with when developing our database.

4.4.2 *Chats*

New chat rooms are created within the database whenever a user starts a conversation with another user (that they haven't previously talked to). Chats were stored separately from user accounts, so that we would avoid creating a heavily nested database structure. This improves runtime for our application because, whenever a user's information is loaded from the database, Firebase only loads the necessary information and doesn't also grab all the chat and messages associated with that user. As of now, chats are only retrieved when a user enters a conversation, only loading the necessary information when needed.

When a new message is sent to the database, an individual chat room has its information updated. This update triggers an event on the receiving client's side, allowing them to grab the data from Firebase and update their chat UI to reflect the newly sent message.

4.4.3 *Messages*

Messages are held within the chat rooms. Each message holds the input language, the output language, the input audio/text, and the output audio/text. These input/output labels are used as a reference for the users when they load previous chats, and are initialized whenever a new chat is created between two people. This type of storage limits conversations to being two-sided, but did manage to keep the information easy to dismantle on the client side.

Every new message sent by the user to the database already includes both the versions of the text/audio.

4.5 The Codebase

The Boogle Translate code base can be accessed at the following links:

Client: https://github.com/jravishankar/BoogleTranslate

Server: https://github.com/zalecodez/TransLang

5. RESULTS

We have built BoogleTranslate, a trans-lingual messaging application for iOS and Android mobile devices. In this section we shall share the results of the application - a demonstration of how it works as well as relevant performance metrics.

5.1 Boogle Translate in Action

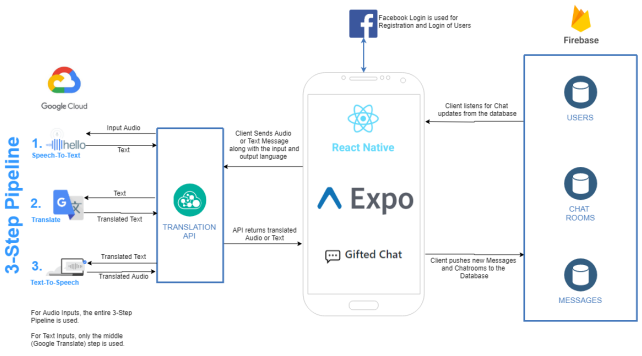

Figure 2: Diagram of app structure

The above diagram (see figure 2) shows a comprehensive model of how our application is structured. The user interacts with the front-end Expo/React-Native application. Any message the user sends will first be sent to our back-end API using sockets, and will then be returned with both input and output versions of the message. This returned message will then be sent to the database, which will update the appropriate chat. This update will then be listened in on by the receiving user, who will subsequently grab the new information from the database and update their own interface.

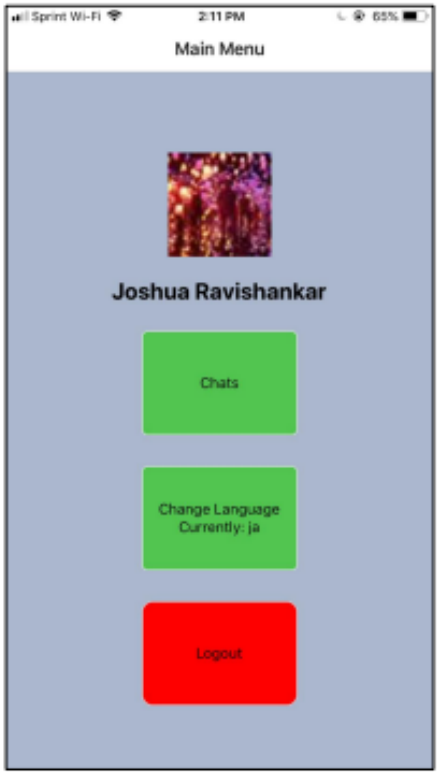

Figure 3: Home screen of Boogle Translate

Once logged in, the user sees a home screen similar to the one shown above (see figure 3). Names and photos are all provided through the initial Facebook login, which then populates our database with all the necessary user information.

When a user enters the 'Chats' screen (see figure 4), they see a list of all their previous conversation partners. At the top, there is also the option to start a new chat, which will take the user to a screen where they can select who to start a new conversation with (conversations can't be started with people who a user is already engaged with).

The screenshots in Figure 5 clearly document the main features of Boogle Translate. The two users, despite operating in two completely different languages, are able to have a conversation with audio and text messages. With this, we were able to successfully enable inter-lingual communication.

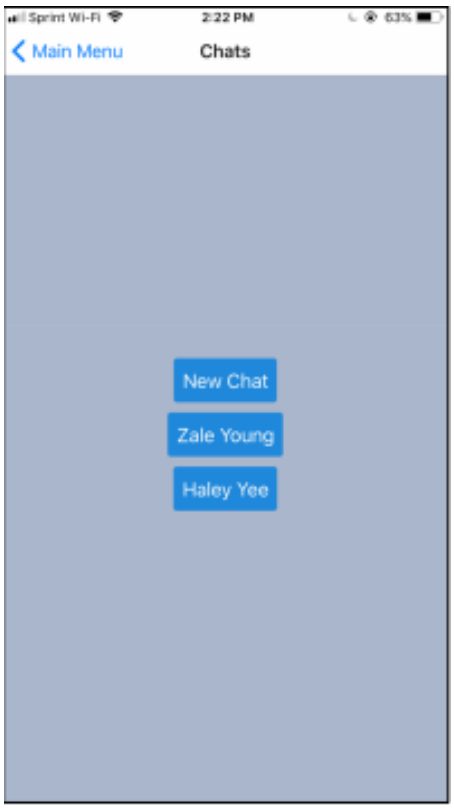

Figure 4: Chats screen of Boogle Translate

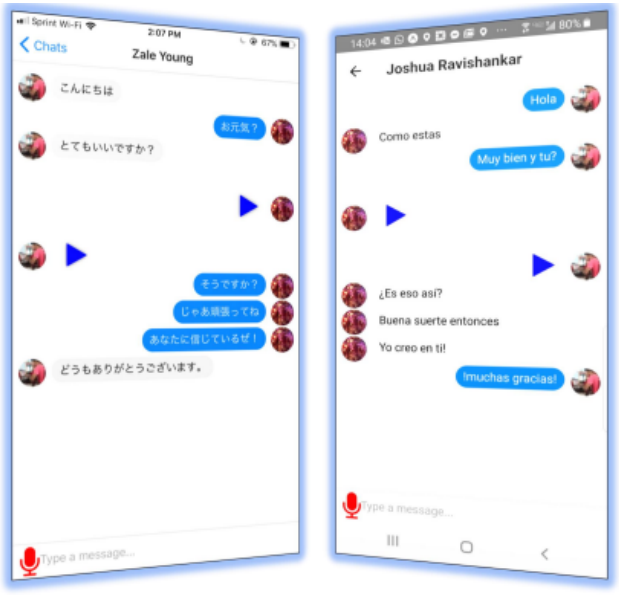

Figure 5: The same chat from two different user perspectives

5.2 Metrics

5.2.1 Accuracy

The accuracy of our translations are completely dependent on the accuracy of the software and algorithms produced by Google, since every aspect of our translation pipeline is routed through Google's APIs.

Google's access to vast amounts of data have provided a great training set for their algorithms to learn with, and with most of this information on the internet, slang and acronyms are also handled fairly well by the translation/recognition algorithms.

5.2.2 Time

Boogle Translate has a lot of latency, making the user experience not as smooth as it should be. This latency is caused by the many steps taken in the pipeline along with the fact that message communication is done through the database rather than faster methods like sockets.

5.2.3 Errors

The scope of Boogle Translate was not incredibly large, so there are very few bugs that occur within the bounds of the deliverable we set out to create. One such bug occurs when a user changes their native language. This causes the labeling of neither the input nor output language to match with the user's 'native' language, therefore causing the wrong text to display within the user's chat. Instead, the user's old 'native' language should be the display language for old messages and any new messages from the language switch on out should be in the new 'native' language.

6. DISCUSSION

Having set out to solve the problem of inter-lingual communication, we created Boogle Translate, a mobile messaging application that enables conversation between two people who don't share a common language. We were able to consolidate many of the currently available language translation software and database/application frameworks to create a functioning messaging platform. Using Boogle Translate, we have shown that language does not have to be a barrier to communication in present-day society. Though messages may not be conveyed with complete accuracy, the vital information contained in them can be passed on with the power of current translation software. As language processing software reaches peak accuracy, it is our job as software engineers to think of elegant ways to piece together this technology with existing frameworks to bring the world closer together.

Boogle Translate is in no way a completely perfect application. There are still many edge cases that haven't been fully taken care of, and the application itself is limited in scope in that it only brings pairs of people together. The application is designed in such a way that once a user's native language is selected, it shouldn't be changed in the future, though this ignores the issue of multilingual users. Though our chats and users are separate within the database, the fact that messages are still nested within chats means that the database still is not structurally sound. By using Facebook accounts, we have (rather hypocritically) restricted our user base to those people who have created or are willing to create Facebook accounts.

Our application is far from being able to serve a large audience, especially that on the scale of Facebook's user base. However, we have made significant progress towards creating a fully fleshed-out application and have provided a fundamentally-sound code base that can be developed into a real product.

6.1 Future Work

This project paves the way for future research in the field of communication oriented language translation platforms. We have demonstrated the feasibility of trans-lingual communication platforms and shown that language translation can be integrated into everyday messaging apps.

The main drawback of Boogle Translate is its high latency. Due to the length of time it takes the back-end to process audio, translate it, and return it to the application, it poses difficulties in high frequency instant messaging as well as the possibility for real time telephone calls. Therefore, this necessitates future research exploring ways to decrease that latency to near real-time operation, whether it be through changes in the structure of the pipeline or in the front-end application's communication with the pipeline.

Boogle Translate would also benefit from output audio being played in the sender's voice, so as to increase the authenticity of conversations. Research on matching voice pitch has been done, but has not made any significant progress towards developing a model to recreate any voice from audio data [12].

Further empirical research needs to take place to determine the utility of our application with regards to everyday digital conversations, which often involve colloquial vernacular and internet slang/acronyms that even the most powerful translation algorithms are not able to accurately translate. Additionally, it will prove useful to research how our application can be used to enrich communication in settings where language barriers are most prominent, such as international business transactions and language-learning environments.

7. ACKNOWLEDGMENTS

Support and guidance provided by Professor Jason Grant was greatly appreciated, and was heavily influential in the development of our application concept.

We would like to acknowledge Google for providing us with APIs that allow us to access their highly accurate speech recognition, translation, and speech synthesis algorithms.

This report template was adapted from Professor Ananya Christman.

8. REFERENCES

[1] Expo.
[2] Ffmpeg.
[3] Firebase.
[4] Heroku.
[5] Node.js web application framework.
[6] React native Âů a framework for building native apps using react.
[7] Your passport tothe world!
[8] Socket.io, May 2019.
[9] D. Amos. The ultimate guide to speech recognition with python âĂŞ real python, Jun 2018.

[10] E. Bacon. How to build a chat app with react native, Jun 2018.

[11] Google. Cloud text-to-speech - speech synthesis | cloud text-to-speech api | google cloud.

[12] H. Li, B. Ma, and C.-H. Lee. A vector space modeling approach to spoken language identification. *IEEE Transactions on Audio, Speech and Language Processing*, 15(1):271âĂŞ284, 2007.

[13] A. Misra. Using rnns for machine translation, Mar 2019.

[14] pypi. googletrans.

[15] D. Vilar, J. Xu, D. Luis Fernando, and H. Ney. Error analysis of statistical machine translation output. In *LREC*, pages 697–702, 2006.

[16] C. Woodford. How speech synthesis works, Dec 2018.