Senior Design

ENG EC 463

Test Plan

To:	Professor Pisano
From:	Arley Trujillo, Shivani Bhatia, Laura Salinas, Priya Kapadia, Steven Graham
Team:	14
Date:	2/13/18
Subject:	Vobot Second Deliverable Test Plan

Overview Description of Test to be Performed

One loop of the entire workflow has been completed:

1. The application prompts user for input by playing a sample audio file
2. The phone records what the user says for 3 seconds and saves it as an audio file
3. The audio file is sent to the SpeechAce API for analysis with relevant parameters
4. The response similarity score is received by the application in the form of a JSON object
5. The JSON response object is parsed and displayed on the screen as a graph
6. If the similarity score is greater than the threshold (50% for example), the robot displays a reward behavior such as a song

The integration of the database is yet to be completed.
In addition, the 3D printed speaker harness needs to be tested and/or modified.

1.0 Migration to CHiP Robot

Description & Goal:
While the previous version of the robot used was viable, a wider range of expressions had to be used as reward behavior for the users. In addition to more expressions and actions, the CHiP robot, which looks and acts like a dog, has a more friendly appeal and is more likely to stimulate children to adapt to the system. In addition, the MIP robot had a stability problem and could not stand without moving. It would often fall, possibly alarming the child. Transitioning to CHiP, a more stable robot, would not upset the child and be able to withstand the child's touch without falling.

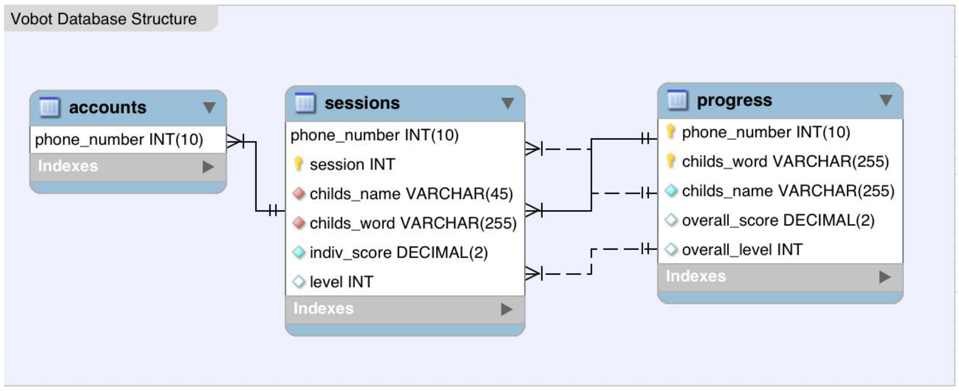
Procedure:
In talking with the client and partners at Wowwee about the goals of the project, the WowWee lab was able to secure the Android SDK for the CHiP model. To migrate the code base for the project from robot MIP to ChiP, the underlying library had to be changed, and the hardware had to be restricted to devices running Android 4.3 and above with Bluetooth Low Energy chipsets.
The WowWee SDK was attained here: https://github.com/WowWeeLabs/CHIP-Android-SDK. After downloading the zip file of the SDK, the project SDK was opened using Android Studio. The functionality that did not pertain to the Vobot project was removed, and the old code from the previous MIP Vobot app was ported in.

Verifiable Result:
The CHiP model is fully functional and integrated with the Android application. The app can connect to any CHiP robot using bluetooth with the click of a button, control its actions and run through a cycle of speech recognition with rewards according to specified progress levels.

2.0 Database

Description & Goal:
By understanding the needs of our users, and wanting to create a system that could be used for long term learning, the benefits of integrating a database for recording each users progress was realized. The goal of the database will be to serve as the central hub for parents and therapists wanting to track a child's progression. With this implementation, the power of the cloud can be leveraged to maintain a small app size instead of being weighed down by local user data. In addition to user progress, the database will have to be coupled with a layer of security to maintain and keep each user's information private.

Procedure:



A MySQL database instance was created using Amazon Web Services. Three tables, whose relationships and fields are shown in the image above, were then subsequently created for the database using MySQLWorkbench. The database is indexed into using each user's phone number as the primary key. Once the database is built, a RESTFul API was created to establish a connection to the database through a local server. When the database is integrated into the application, the RESTFul API will act as the gateway between the Android application and MySQL database. The API is then called by launching the server in Postman. Using Postman, we can enter the main server URL followed by the specific routes to add to, modify, or retrieve information from the database. In order to pull information from the database, the GET request must be selected and the appropriate route must be selected. In order to push information to the database, either a PUT or POST request must be selected and the appropriate route must be selected. The information to be pushed should be entered in a raw, JSON format within the body parameter. All information pushed to and from the database is returned in a JSON format to confirm the action taken.

Verifiable Result:
The database is currently capable of having information pushed, pulled, and modified within it. After running the server.js file, the backend system can be tested by launching Postman, and typing the appropriate requests and routes into the address bar at the top of the application. After testing routes that push, pull or modify data in the database, routes that select and display all of the information in the database can be run to verify that the backend system works. The ability to pull and push information from the database will allow Vobot to create profiles for each individual user and designate which learning data corresponds to whom.

3.0 Integrating Speechace API to Android Application

Description & Goal:
Speech recognition is essential in understanding what the users are saying. We need to monitor a user's speech progression by using Speechace API, which receives vocalizations as an input, and returns a similarity score as an output. Speechace also breaks down the input vocalizations into phonemes and syllables, and provides a similarity score on each. Handling user input in this way allows us to shape a child's vocalizations, ultimately getting them to speak a particular set of words with less difficulty. In previous deliverables we were running Speechace off the terminal. Now, it is integrated into the Android application so that we can send Speechace requests from the phone.

Before making the decision to fully integrate Speechace with the Android application we ran multiple tests with samples of children's vocalizations provided by the client (*see Figures 1-5 in appendix)*. The samples of audio were not necessarily of good quality so the similarity score of subjective human scores versus Speechace scores tends to vary quite a bit. However, in the environment that this system would be used, ie: with children in their homes or at the therapist office, we would indicate that a noise free environment is preferred for accurate scores to be derived.

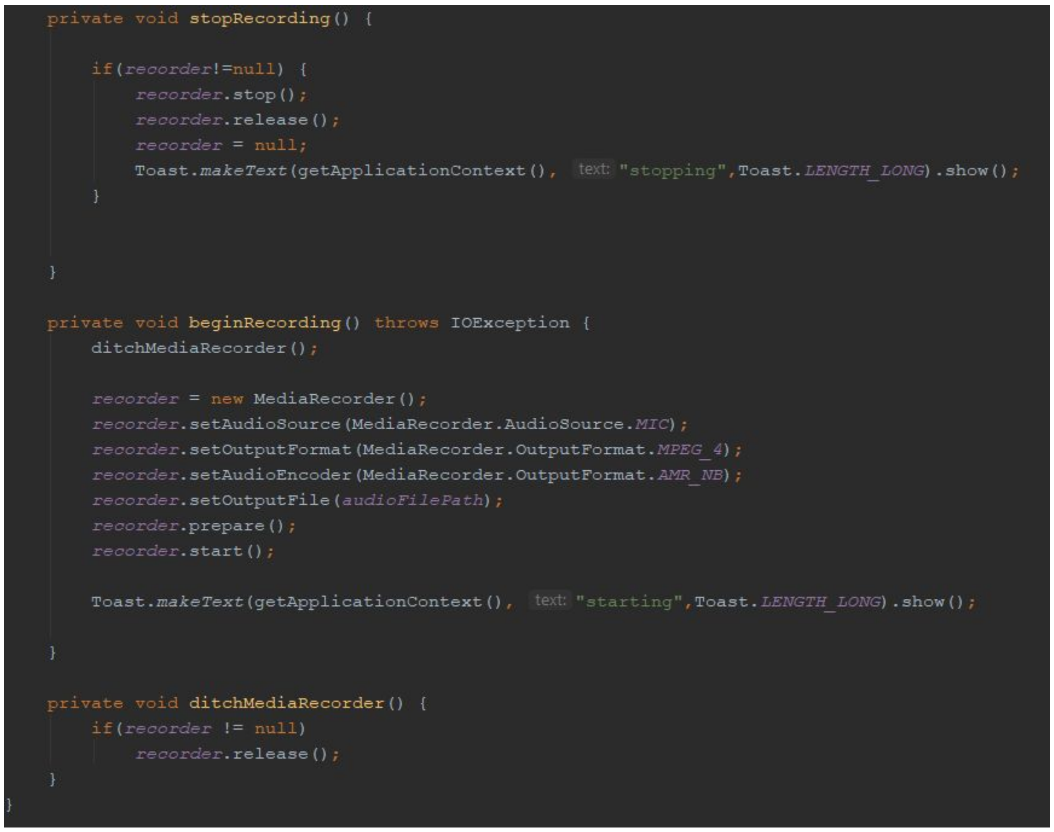
Procedure:
To implement a successful call to the Speechace API, the API documentation was used: https://documenter.getpostman.com/view/2675436/speechace-api-calls/6tW8nSr
The sample requests were opened using a software called Postman. Using Postman, a Java query was formulated using OKHttp. The headers and body were dissected to provide the necessary information required to create a request. The header included the SpeechAce key, the user id, and the dialect. The body included the audio file that needed to be analyzed, and a text field noting the word or phrase to be scored against.

Verifiable Result:
Once the user finishes speaking, their audio is saved and formatted, and the API call is made which will print the similarity score on the appropriate screen in the app. By capturing the users vocalizations, they attain a score indicating how close the recorded word is to the model word within 1 second. The comparison will allow us not only to give feedback on speech progress, but also serves as the foundation for the module which rewards users based on their progress.

4.0 Saving Audio Recordings

Description & Goal:
In order to understand how close vocalizations are to model words we are incorporating the Speechace API. This API requires a recording of the vocalization in order to analyze and return a similarity score. Speechace then breaks down the word by phoneme to generate a result of how accurate the vocalization is to the model word. Before being able to integrate Speechace API with the Android application we first needed to be able to capture the audio spoken and save it locally.

Procedure:



```
    private void stopRecording() {

        if(recorder!=null) {
            recorder.stop();
            recorder.release();
            recorder = null;
            Toast.makeText(getApplicationContext(),  text: "stopping",Toast.LENGTH_LONG).show();
        }


    }

    private void beginRecording() throws IOException {
        ditchMediaRecorder();

        recorder = new MediaRecorder();
        recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
        recorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
        recorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);
        recorder.setOutputFile(audioFilePath);
        recorder.prepare();
        recorder.start();

        Toast.makeText(getApplicationContext(),  text: "starting",Toast.LENGTH_LONG).show();

    }

    private void ditchMediaRecorder() {
        if(recorder != null)
            recorder.release();
    }
}
```

A media recorder is initiated with certain parameters: the audiosource (currently the phone's mic), the output form (MPEG.4), the audio encoder, and the location where we want to save the file (which is on the SD Card). After the recording is over, which ends after 5 seconds, the audio file is saved over the previous file. The media recorder is released once the recording has stopped and isn't prepped again until we want to record again. The child is prompted to speak by the speaker on the phone, where a pre recorded audio file is played. Once the audio file is over, the recording begins.

Currently, we manually record the model vocalization ourselves to be played aloud to the child during sessions. These recordings are saved locally in the resources folder under our source set. Since the client denotes nearly 150 words in his grant proposal document, moving forward with the current system is neither ideal nor practical long-term. We intend to utilize a different service that SpeechAce offers in which we may generate an audio file given a word as input to resolve the memory-intensive setup currently in place.

Verifiable Result:
Once the user is prompted to start speaking, any audio captured by the recording device is saved locally and passed to Speechace. It is s then passed to Speechace for analysis and phoneme breakdown.

5.0 Reward Action Behaviors

Description & Goal:
The core idea behind this project is encouraging nonverbal autistic children to begin speaking in a safe and comforting environment. In order to allow for progression across word discovery, our ability to provide positive reinforcement is crucial to the motivation of our users. Due to the clinical experience our client has shared with us, we know that positive reinforcement must be delivered in a staggered form such that a child is not moving on to future levels by trial and error vocalizations. This process is called progressive shaping of successive approximations and is implemented via a levelling system.

Procedure:
Level 1: CHiP will dance for each vocalization, and after 5 successful vocalizations (quality score ≥ 1) will progress to Level 2 alongside rewarding the child. In Level 1, the child has unlimited attempts to vocalize successfully, however if too much time is spent on this level then CHiP will switch words.

Level 2: CHiP will now dance for more improved vocalizations (quality score ≥ 10), and levels beyond the first are alloted 5 attempts total. If an attempts quality score ≥ 10, the child progresses to Level 3 and is rewarded, however if the child fails to cross the threshold score in the 5 attempts returns to Level 1.

As the levels increase in number, the threshold quality score to advance increases as well. The levelling system does abide by a few rules:

- CHiP will model the word after each level change
- If the quality score ≥ 90 on an attempt, CHiP will reward with a dance and switch words
- The quality score ϵ [1,100]
- Sessions are 30 minutes long

Verifiable Result:
CHiP is currently able to give positive reward feedback by performing songs. Initially upon first attempt vocalizations CHiP will always return a reward for any utterance made, then upon the remaining four attempts CHiP only returns a reward behavior if 90% accuracy is met.

6.0 3D Printed Speaker Harness

Description & Goal:
The goal for this component is to create a sturdy, reliable harness for CHiP to wear which will hold the PRED Smart Cube Speaker. As children interact with the robot, it is important that the instructions to start practicing their vocalizations appear to be coming from the robot itself. The seamlessness with which CHiP plays with the child is a major aspect in the scope of our project. The more fun and user friendly the lesson learning is for the user, the more willing they will be to use the system. This harness was designed and customized to fit the CHiP robot using SolidWorks studio.

Procedure:
By measuring the dimensions of the CHiP robot and brainstorming ideas for how we wanted to attach a speaker, we could begin to model a variety of options through SolidWorks for rapid prototyping on a 3D printer. Depending on which model of the harness is chosen, setup is different, but very simple. The PRED Smart Cube sits inside the rectangular holder regardless of the model. The clip on harness simply slips onto the back of CHiP and is secured in place by design. The collar model has two points of attachment, through CHiP's neck and through the belly.

Verifiable Result:
With the harness in place we can control where the audio instructions come from during lessons. Having the speaker secured on CHiP's back it remains out of the way of the children, but still maintains its functionality and maintains the illusion that CHiP itself is the instructor. We have three more models waiting to finish printing this week. Figures 6 and 7 are SolidWorks drawings of these models.

Appendix

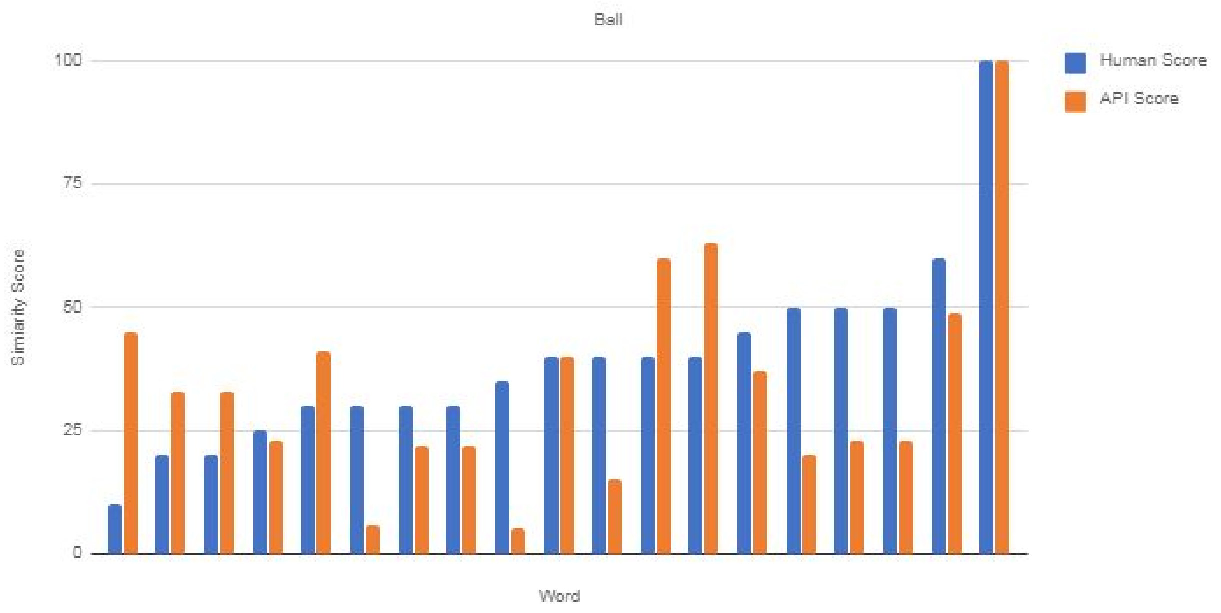


Figure 1

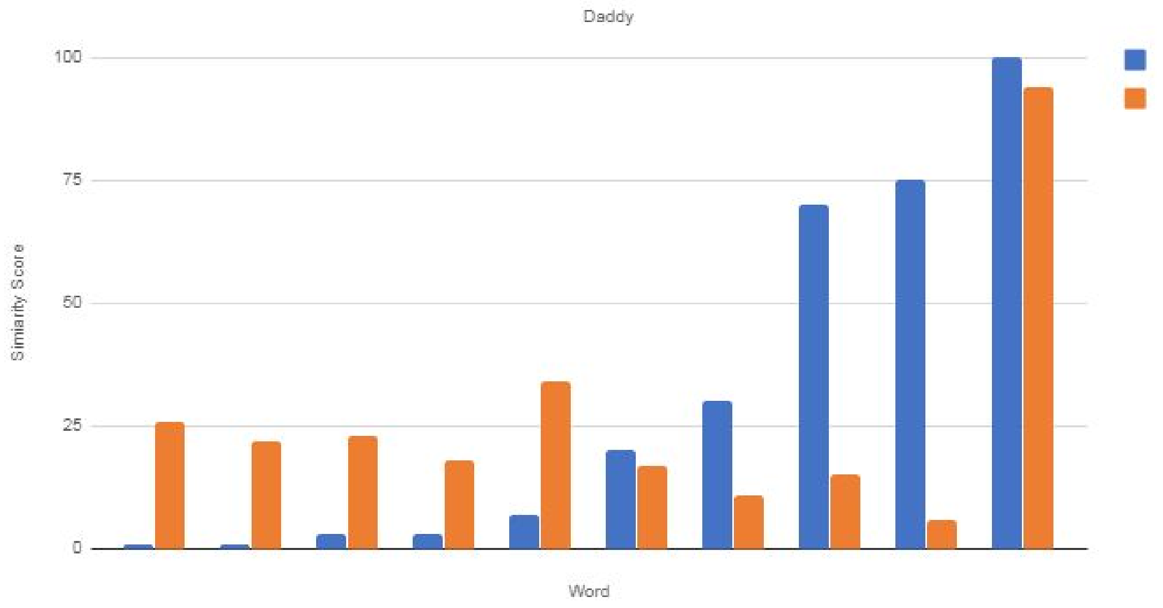


Figure 2

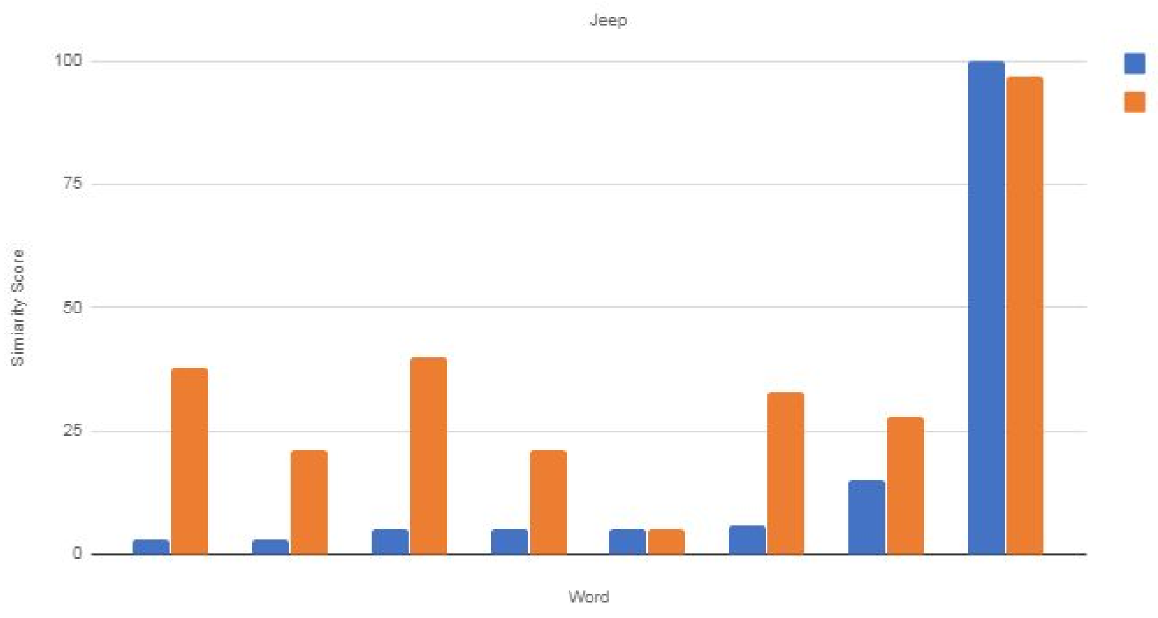


Figure 3

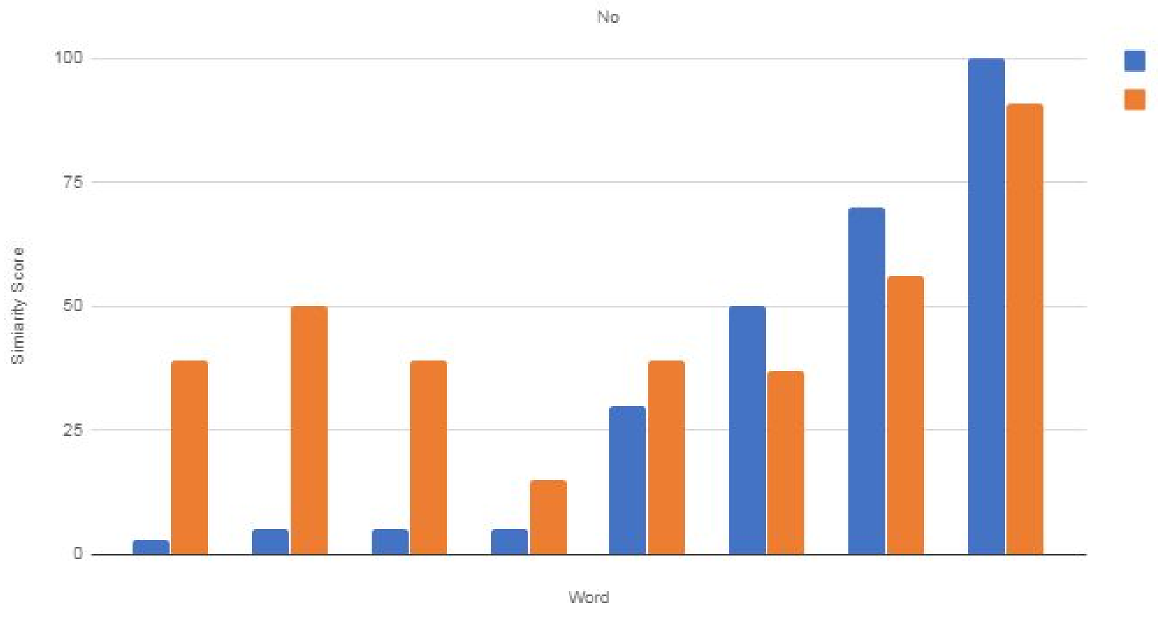


Figure 4

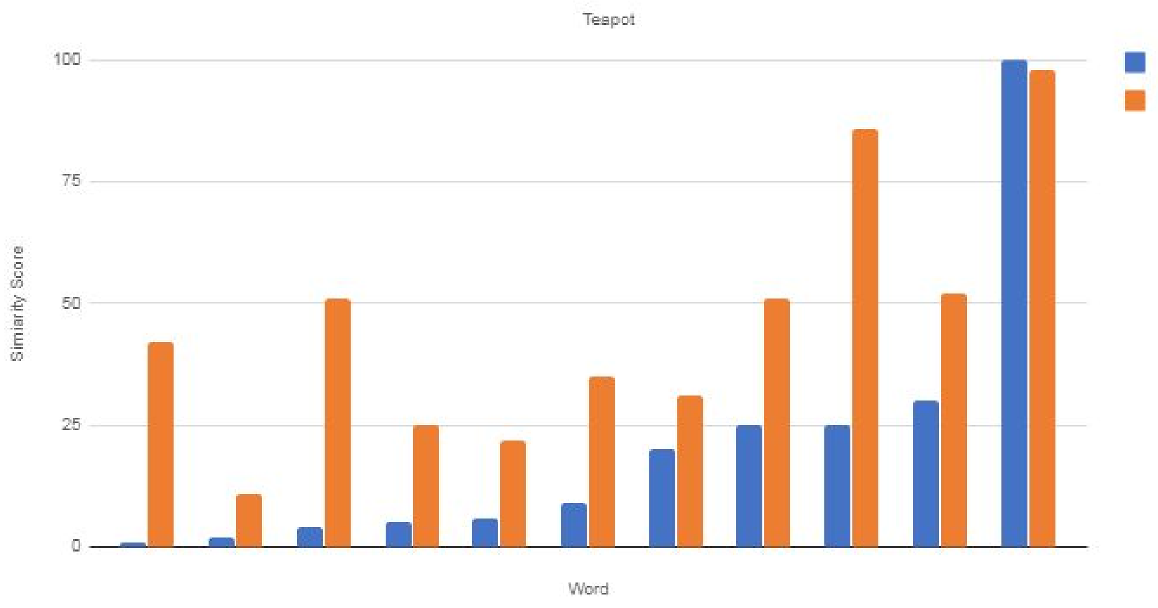


Figure 5