JabberChess, A Mobile Chess App Speech Recognizer for Visual and Motion Impaired Users

Jackson Chen

Words: 2839

Abstract

As the mobile application marketplace has proliferated within the past few years, so has our reliance on them to perform basic day to day activities, such as GPS navigation. However, this is not the case for visually or motion impaired users, as the market for apps accessible to disabled users is fairly unsaturated. In this project, I focus on a specific category of apps that I am familiar with: chess. From my research, chess apps lack accessibility to the visual and motion impaired due to the lack of an accurate and affordable speech recognizer. This leads to the research question: Could a mobile chess application be developed to overcome voice recognition inaccuracy and costly price? This project introduces JabberChess, a speech recognizer that is both suitable and economical for mobile chess apps. It is affordable because it is built on a leading, free speech recognition toolkit. Despite using a state of the art speech recognition toolkit, the toolkit's default recognizer is virtually unusable for chess apps due to its extremely poor word recognition accuracy for chess vocabulary. In order to improve the accuracy, this project trains a new language model and a new grammar model specifically suitable for chess vocabulary and other chess related commands that are used in the new recognizer. Equipped with the newly trained grammar model and the language model, the new recognizer introduced by this project shows a significant improvement in performance in average word accuracy and sentence accuracy. The result of this project can hopefully open a new era of mobile app development: one aimed to include those with visual or motion disabilities.

1. Introduction

Automated speech recognition (ASR) technology has improved dramatically over the past decade, which has lead to a boom in speech-activated applications, especially in the mobile

market. This boom has been supported by both the business and the research community: large technology companies have been developing voice-activated personal assistants, such as Siri, Google Now, and Cortana, while researchers have been investigating the many applications of ASR, such as educational applications for visually impaired students [1]. In fact, many applications focus on improving the mobile experience for visually and motion-impaired users [2]. Despite this, the blind-accessible mobile gaming market is largely unsaturated. For example, there are no speech-activated mobile chess apps available on either the iOS or Android platforms. Searching extensively online for such a mobile chess app returns one app on iOS called Blind Chess Trainer. It is self-proclaimed as "fully accessible to blind users", yet it does not use ASR and could only read chess moves to the user [3].

This project aims to create a speech recognizer that is not only accurate in recognizing words and phrases but also affordable. A recognizer that fits these two criteria can be used in a mobile app which can be widely distributed throughout the mobile application ecosystem. Current available commercial ASR products do not satisfy these two requirements. For example, the leading commercial ASR system is owned by Nuance. Their ASR recognizer is cloud based and provides convenient software developer APIs on all major mobile platforms. They claim that their ASR recognizer "delivers the industry's highest recognition accuracy" [4]. However, it would cost more than $8 a day for a regular user (calculated based on the user playing 10 games a day) [5], while users are typically accustomed to free mobile apps [6]. On the other hand, CMU Sphinx, a state-of-art open source speech recognition toolkit [7], provides a free, trained decoder with very low recognition accuracy for chess notation and commands. This leads to the research question: How accurate is an ASR decoder trained with a special language model and a special

grammar model for chess vocabulary compared with the default decoder provided by CMU Sphinx?

A decoder trained with a language model or a grammar model should be much more accurate than free decoders, and much cheaper than commercial solutions. A speech driven chess app would improve the user mobile experience for visual or motion impaired users, as well as benefit the chess community with an influx of new players. Furthermore, this work can promote other game developers to create games that can be enjoyed by disabled users.

2. Methods

2.1 Speech Recognition Decoder Architecture

CMU Sphinx-4 is a pure Java speech recognition library that this project will use to build the recognizer as shown in Figure 1. Firstly, the feature extraction module extracts feature vectors, 39 numbers that represent the speech signal. Then, the decoder decodes these feature vectors into words with the help of the modules in the knowledge base. The acoustic model is aware of the properties of atomic acoustic units, also called "senones" [8]. The dictionary consists of mappings from words to phones. The language model is an n-gram model (in the ARPA format), which contains the statistics of word sequences. The language model is used to restrict word search in the Hidden Markov Models (HMM) sequential process. Grammars contain the JSpeech Grammar Format (JSGF) textual representation of grammars used to determine what the decoder should listen for, and thus describe the utterances a user may say. Together these modules help the decoder find the best text output (called hypotheses).

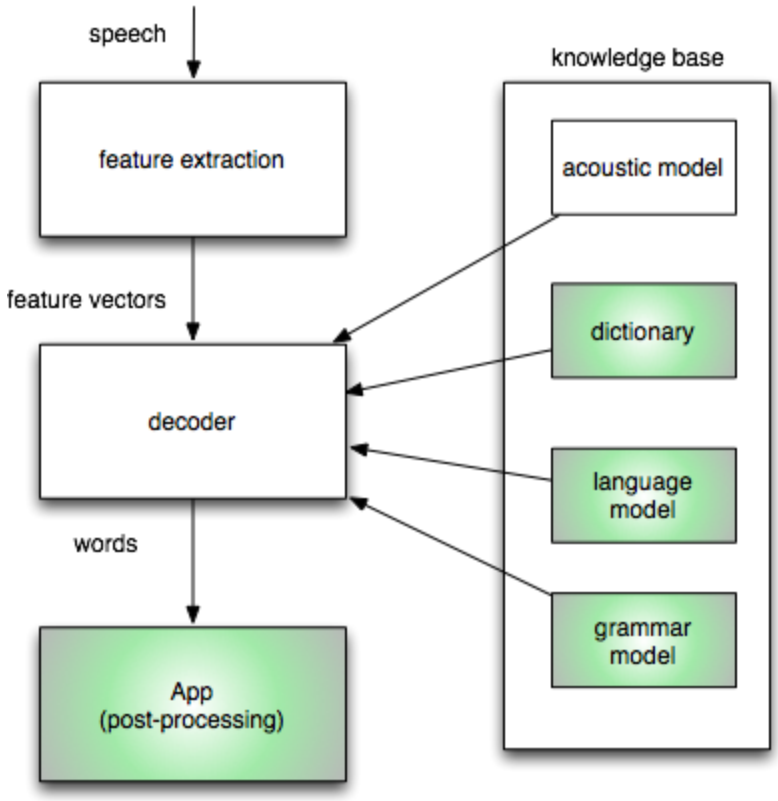

Figure 1: CMU Sphinx ASR pluggable architecture

CMU Sphinx-4 provides state-of-art acoustic modeling using mel-frequency cepstrum (MFCC), and consists of features with noise tracking and spectral subtraction for noise reduction. More importantly, however, Sphinx-4 has a flexible architecture, allowing for the replacement of its default models with customized ones that I built. The latest CMU acoustic model, "generic US English acoustic model v5.2", is used as the experimental control. As shown in Figure 1, I trained my own language model, dictionary, and grammar model to replace the default ones (see the green modules in knowledge base). With Sphinx, one can only use either a language model or a grammar model, but not both.

2.2 Speech Recognition Modeling for Chess

2.2.1 Chess Notation and Language for Mobile Apps

Algebraic notation [9] is the most popular notation used to record and describe the moves in a game of chess, where every square on the chessboard is identified with a unique letter and number pair. The vertical coordinates are called *files* and labeled **a** through **h**, from white's left to white's right. Similarly, the horizontal coordinates are called *ranks* and are numbered from **1** to **8**, starting from white's home rank. Each square of the board, then, is uniquely identified by its file letter and its rank number as shown in Figure 2. Algebraic notation is the standard in chess computer and mobile applications, thus the chess app users will speak commands using algebraic notation. Therefore, the speech recognition detector needs to recognize algebraic notation. Even though there are a total of 8 numbers (1 through 8) and 8 letters (a through h), there are thousands of combinations when linked other chess vocabulary (e.g capture, promotion, check, etc.). One move consists of full a turn by each player (white and black). A **ply** is a half-move. For example, white moved his pawn to e4:

1. e4

Then, black moved his pawn to c5:

1… c5

One chess "move" was just completed because white and black completed their turns. The complete move can be shortened to the following:

1. e4 c5

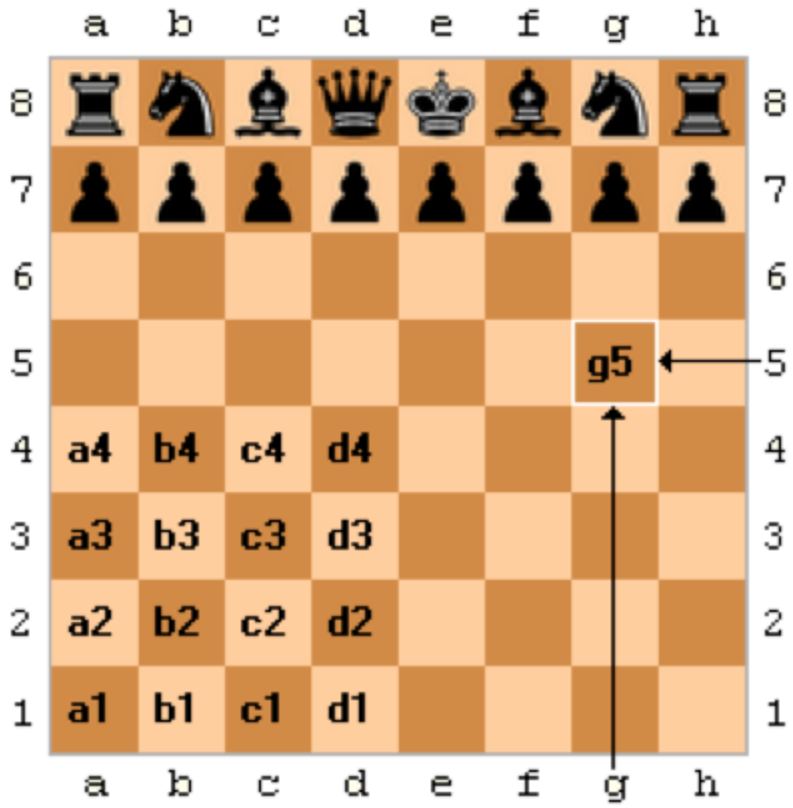

Figure 2: Chessboard and its coordinates used in chess algebraic notations

2.2.2 Chess Language Model

Since the chess language has a small set of vocabulary and a large set of word combinations, the generic English model poorly recognizes chess terminology. Thus, the language model created in this project needs to be trained with chess terms and sentences. To do so, a list of all possible chess commands is needed, for instance:

Ra1, Ra1+, Ra1#, Rxa1, … …

The above algebraic notation needs to be translated into speech recognizable format for the training program to understand:

Rook a one, Rook a one check, Rook a one checkmate, Rook captures a1, … …

Since there are 27,043 possible combinations, I wrote a program to output all possible sentences. The output of the program is a corpus file containing all 27K chess commands. I then used the CMU Lmtool [10], a statistical language modeler, to train the chess language model. The training program not only outputted the new ARPA formatted language model, but also provided the dictionary. I used the newly trained dictionary and the newly trained language model together in the decoder as shown in Figure 3.

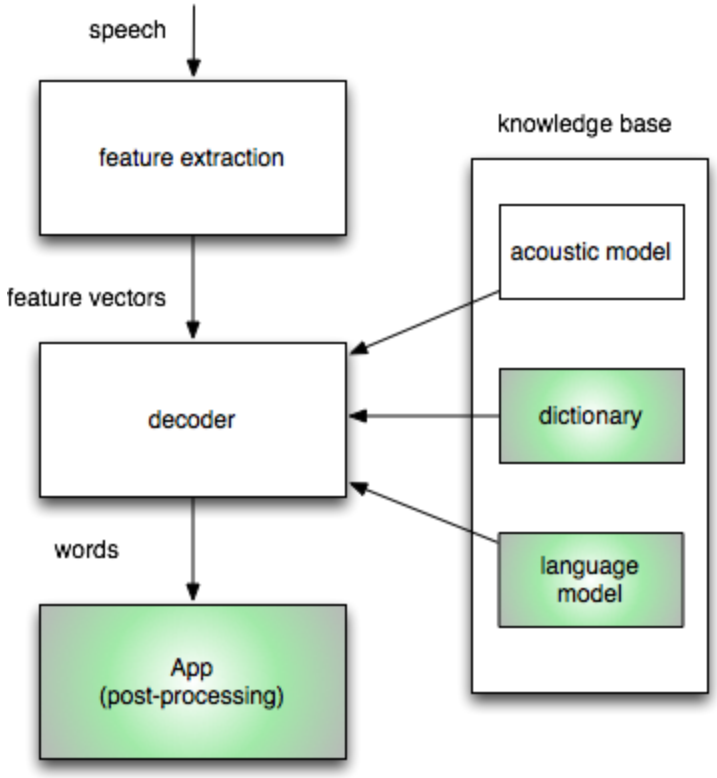

Figure 3: Decoder with newly trained language model and dictionary

2.2.3 Chess Grammar Model

An alternate way to restrict the HMM finite state search is to provide a chess grammar model. The grammar model is configured throw a JSGF [11] grammar file where possible grammatical

patterns of chess commands are specified. For example, the following line of code states that the user's chess command can be either *menu_command* or *move_command*:

public <command> = <menu_command> | <move_command>;

where a *move_command* can be "queen [<capture>] <col> <row> [<ck>] ", which means "Qc4", "Qxc4" or "Qxc4+", etc. Appendix A provides the details of the grammar model file.

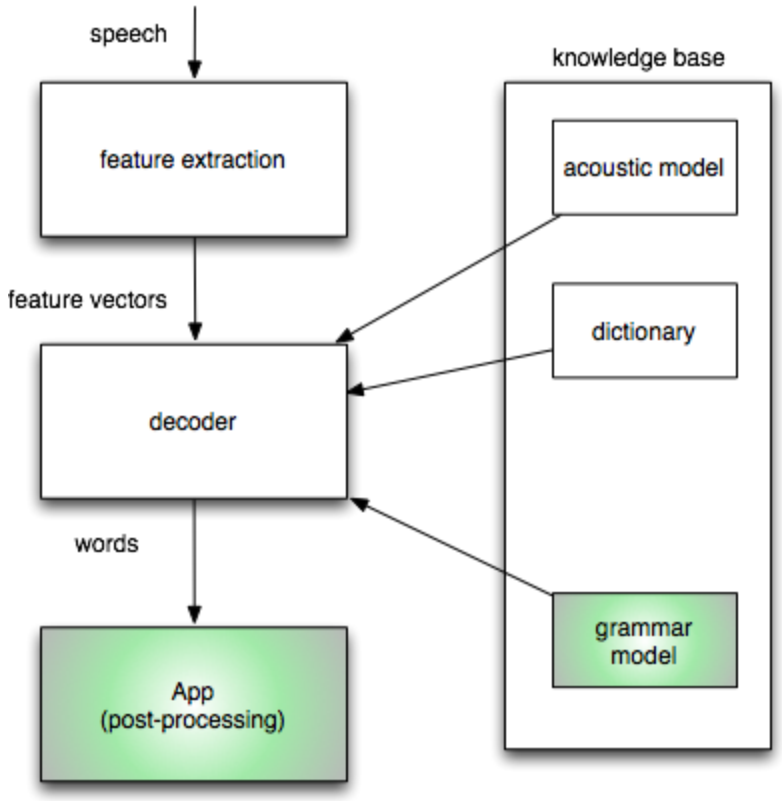

Figure 4: Decoder with the new chess grammar model

3. Results

3.1 Test Methods And Data Preparation

Due to the lack of a ready to use test chess corpus, I used chess online audio materials [12] and live recordings of people's voices to create the WAV files. Each WAV file contains one chess

game. A transcript from the chess moves in each chess game is created, and then each WAV file is fed to the decoder in order to obtain the hypothesis (to a file).

Figure 5 shows the project's first decoded chess game. Each line (sentence) is one chess ply (command), e.g. "Nf3" or "knight f three". The recognized hypothesis is then compared against the original transcript using the NIST alignment and scoring algorithm [13], which employs a word error rate (WER) and accuracy rate to measure the ASR recognition performance. I used the CMU NISTAlign package, which is an implementation of the NIST algorithm. The package can only compare one pair of strings (in my case, one sentence) at a time. It will then aggregate the final statistics in the end.

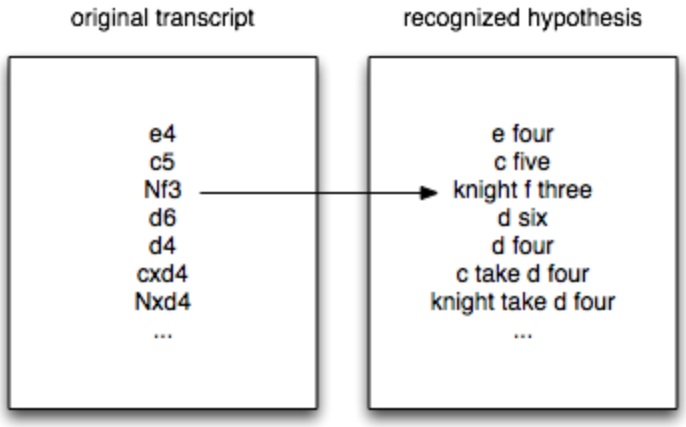

Figure 5: Sample test transcript and hypothesis

For each test audio (a chess game), I constructed two lists of strings, one for the original transcript and the other for the hypothesis. Each item on the list corresponds to one ply in the original order in the game. One issue with this input approach is when there is a mismatched line between the original transcript and hypothesis, all of the lines after the mismatched line will be compared incorrectly, as shown in Fig 5. In this case, I must replace the mismatched line with a

text string (“yyyyyy”) to ensure that there is an erroneous line in original transcript. This way, NISTAlign can count this line as incorrect and continue to compare the following lines.

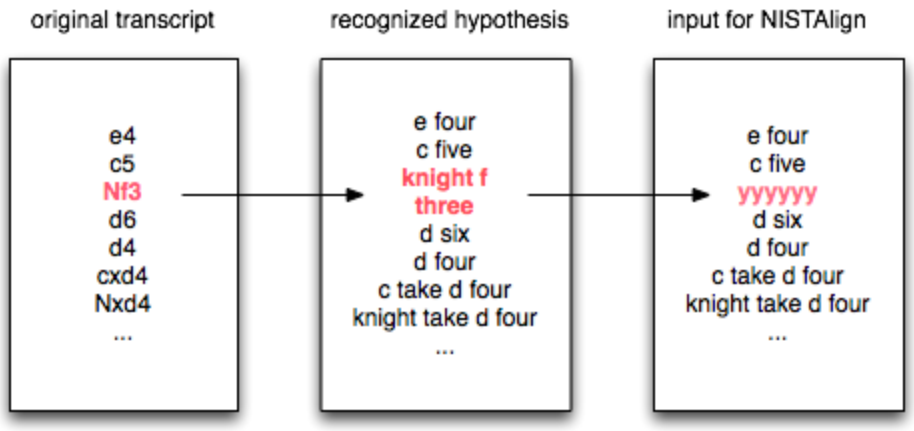

Figure 6: Replacing a mismatched line for NISTAlign comparison

I also performed a simple test on Nuance’s speech recognizer using Nuance’s sample recognizer app on the Android platform. I took one of my test scripts and dictated it using the sample recognizer app to analyze its result. However, no statistical tests were performed on this data due to the lack of quantity of data.

3.2 Data Analysis And Results

Following criteria are used in my analysis:

$$WER = (I + D + S) / N$$

where *N* is the number of words in the original transcript text. *I* words were inserted, *D* words were deleted and *S* words were substituted. Accuracy is nearly equivalent with WER, but does not count insertions:

$$Accuracy = (N - D - S) / N$$

I also measured recognition speed for each test case. Suppose the audio file was 2 minutes long and the decoding took 6 minutes. Then speed is counted as *3*RT*.

Figure 7 shows the word accuracy for the decoder using various models. Both my grammar model and my chess language model improved the word accuracy dramatically. The average accuracy for language model is 87% and grammar model 86%, in contrast with the 15% from the default model of the CMU Sphinx-4 as shown in Table 1. The difference between the newly trained language model and the grammar model is very small. Fig. 8 shows the sentence accuracy between the 3 models. The patterns are very similar to that of word accuracy. The average sentence accuracy is 68% for the language model and 66% for the grammar model.

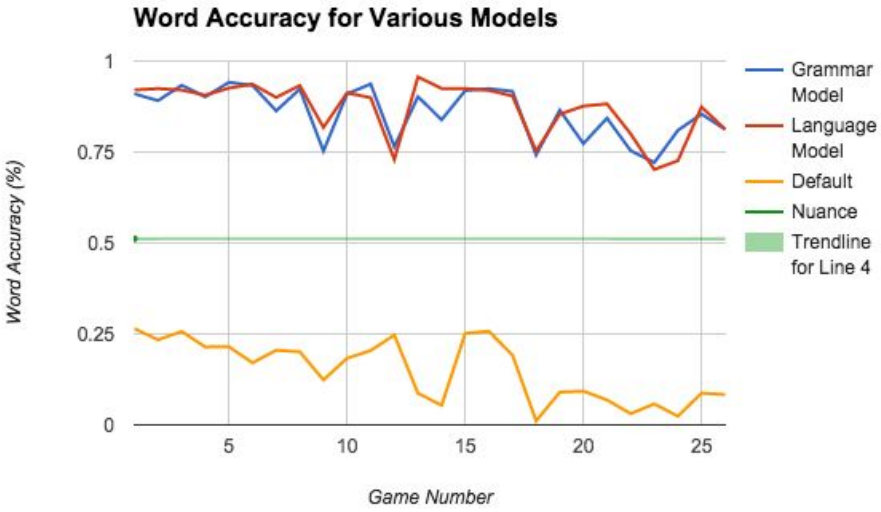

Figure 7: Word accuracy graph for various models

	Grammar Model	Language Model	Default Model
Average	0.859	0.871	0.150
Standard Deviation	0.0709	0.0736	0.0841

Table 1: Average and standard deviation for word accuracy

Figure 9 shows the decoder recognition times between models. The grammar model is twice as fast as the language model while the language model is 10 times faster than the default model.

Fig.7 and 8 also shows that Nuance's speech recognizer is below my language and grammar models in accuracy but better than the default model.

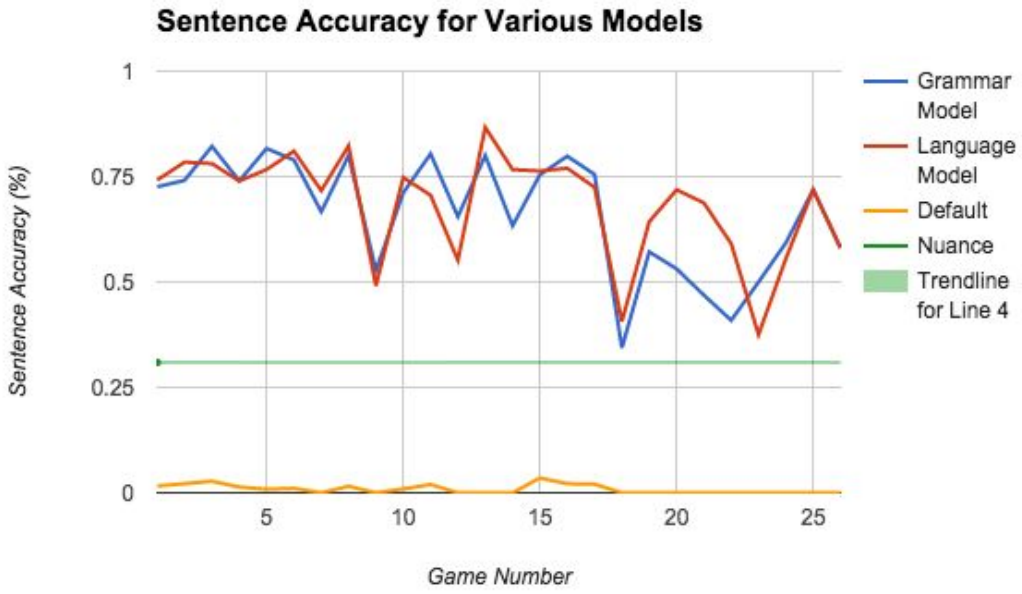

Figure 8: Sentence accuracy graph for the various models

	Grammar Model	Language Model	Default Model
Average	0.664	0.686	0.00841
Standard Deviation	0.137	0.126	0.0106

Table 2: Average and standard deviation for sentence accuracy

In addition, the One-Way Analysis of Variance (ANOVA) test was performed on the two datasets. The F_{obs} was 757.9 and 333.6 for word accuracy and sentence accuracy, respectively; these values were both greater than the F_{crit} values, meaning that the null hypothesis can be rejected and it can be concluded that there was a significant difference between the accuracies. Furthermore, the Tukey-Kramer test was performed on pairs of models for each dataset, and it was found that for both word and sentence accuracies, the grammar model and language model

were not significantly different from each other, but both were significantly different from the default model.

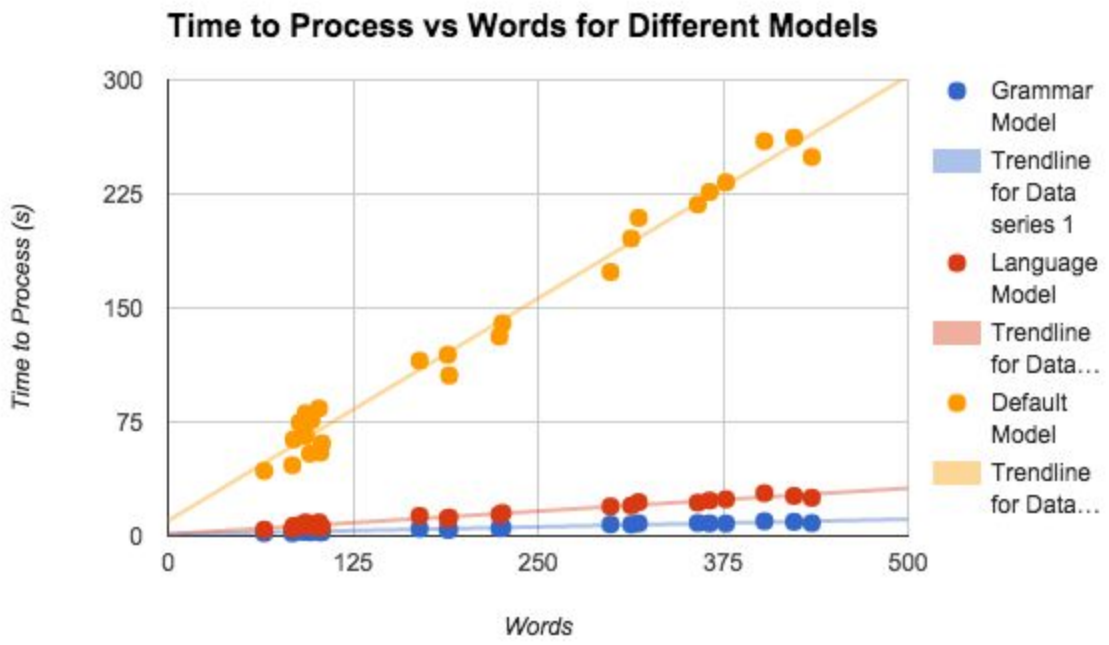

Figure 9: Decoder processing time graph for the various models

	Grammar Model	Language Model	Default Model	Words
Average	5.162692308	13.73307692	131.3457692	207.4615385
Standard Deviation	2.629695888	7.846439075	75.54561272	

Table 3: Average and standard deviation for processing times

4. Future Work

This essay presents a developed prototype of a speech activated mobile chess app on the iOS platform. It uses Stockfish [14], an open source mobile chess app, as the base app. Due to time constraints, this prototype has not been integrated with the newly trained speech recognizer as described above. However, the Speech Controller and Speech-to-Command Controller are built as shown in Fig. 10. The Speech Controller records the user's speech, saves it to a WAV file and

will send it to the app's speech recognizer once it is integrated. The speech recognizer will use the trained decoder to obtain the corresponding hypothesis and will return it to the Speech Controller. Then, the speech controller will send an event to the Speech-to-Command Controller via the iOS default event center. The Speech-to-Command Controller performs hypothesis post-processing and validation using the current game and chessboard position. It translates the user's speech into a chess game command and sends it to the Stockfish Game Controller to execute. The post-processing will, to some extent, compensate for some of the speech recognizer's mistakes since the Speech-to-Command controller may figure out the user's true intentions by using the context of the game. The end user will be able to play chess without visually looking at the chessboard and in a completely hands-free manner. Even though all components are based on open source software, the quality is still very high. In fact, Stockfish has been on the mobile market for years and has consistently been one of the strongest chess engines [15].

5. Conclusion

In this project, I introduced a low cost, high accuracy chess speech recognizer that can be used to build the first mobile chess app for visual impaired users. It is based on CMU Sphinx-4 speech recognition libraries. My results show that the new grammar model and language model both have much better accuracy compared with the default CMU Sphinx-4 model and possibly Nuance's speech recognizer. Also, the difference between the two new models is very small. However, the processing speed of the grammar model is much faster than the language model. Therefore, I recommend to use the grammar model in future mobile chess applications. For the

future, more testing data needs to be generated, especially for the Nuance commercial model. That way I can fully compare the quality of my open sourced models against the leading commercial models. In addition, a full implementation of the mobile chess application for the visual and motion impaired needs to be completed and testing to see its full impact. Overall, JabberChess is an answer to the research question by providing a cheaper solution and a much more accurate speech recognition engine; however, more testing and fine-tuning is needed for such an app to be ready to be released to the market.

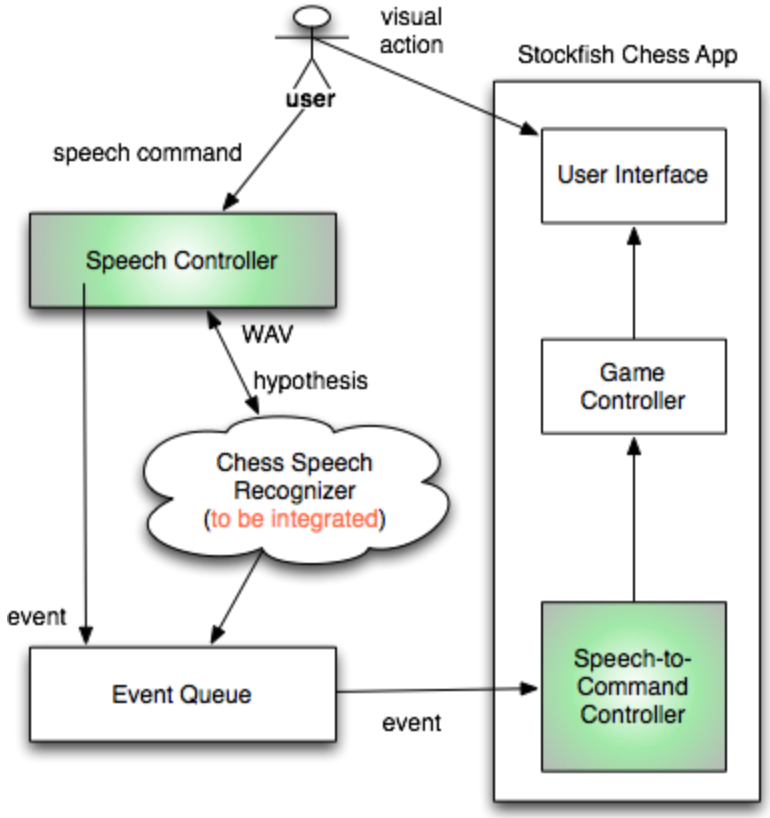

Figure 10: Speech activated iOS chess app

Appendix A: Grammar Model Configuration

A copy of my chess speech recognition grammar model file is on the following page.

```
<row> =   one   |
          two   |
          three |
          four  |
          five  |
          six   |
          seven |
          eight ;

<col> = a  |
        b  |
        c  |
        d  |
        e  |
        f  |
        g  |
        h  ;

<promote> = queen  |
            rook   |
            knight |
            bishop ;

<pair> = rook  |
         knight |
         bishop ;

<ck> = check | checkmate;

<eat> = captures | takes | capture | take;

<menu_command> =  resign                        |
                  [please] show [board] position |
                  new game                      |
                  exit [[the] program]          ;

<move_command> = queen [<eat>] <col> <row> [<ck>]            |
                 king [<eat>] <col> <row> |
                 castle | castle king side | castle queen side |
                 <pair> [<col>|<row>] [<eat>] <col> <row> [<ck>]|
                 [pawn] [<eat>] <col> <row> [<promote>] [<ck>]  ;

public <command> = <menu_command> | <move_command>;
```

6. References

[1] Lučić, B., Ostrogonac, S., Vujnović Sedlar, N., & Sečujski, M. (2015). Educational Applications for Blind and Partially Sighted Pupils Based on Speech Technologies for Serbian. *The Scientific World Journal*, *2015*, 839252. http://doi.org/10.1155/2015/839252.

[2] Zhong Y., Raman T., Burkhardt C., Biadsy F., & Bigham J (2014). *JustSpeak*: enabling universal voice control on Android. In *Proceedings of the 11th Web for All Conference* (W4A '14). ACM, New York, NY, USA, , Article 36 , 4 pages. DOI=10.1145/2596695.2596720 http://doi.acm.org/10.1145/2596695.2596720

[3] The self-proclaimed "fully accessible chess app" that does not use ASR is called "Blind Chess Trainer" by Marcel Nijman. http://wise-games.com/en/Chess.html

[4] Nuance's promotional website: http://www.nuance.com/for-business/automatic-speech-recognition/automated-ivr/index.htm

[5] Nuance's Plan and Pricing: http://dragonmobile.nuancemobiledeveloper.com/public/index.php?task=memberServices

[6] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. 2013. Why people hate your app: making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (KDD '13), Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthurusamy (Eds.). ACM, New York, NY, USA, 1276-1284. DOI=10.1145/2487575.2488202 http://doi.acm.org/10.1145/2487575.2488202

[7] CMU Sphinx project homepage: http://cmusphinx.sourceforge.net/wiki/sphinx4:webhome

[8] An overview on the senones module http://www.cs.cmu.edu/~tanja/Lectures/JRTkDoc/OldDoc/senones/sn_main.html

[9] Wikipedia page describing Chess Algebraic Notation: https://en.wikibooks.org/wiki/Chess/Algebraic_notation

[10] CMU Sphinx tool: http://www.speech.cs.cmu.edu/tools/lmtool-new.html

[11] W3 description of the JSpeech Grammar Format: http://www.w3.org/TR/jsgf/

[12] Joe Gallagher (Speaker). (2007). CD Starting Out: The King's Indian. Everyman Chess.

[13] Description of the NIST Algorithm:
http://www1.icsi.berkeley.edu/Speech/docs/sctk-1.2/sclite.htm

[14] The official Stockfish app website: https://stockfishchess.org/

[15] List of strongest chess computing engines: http://www.computerchess.org.uk/ccrl/404/