New Content Functionality for an Automated Oral Reading Fluency Tutor

A Project Report

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Meghan Day, B.A.

Graduate Program in Computer Science and Engineering

The Ohio State University

2017

Master's Examination Committee:

Eric Fosler-Lussier, Advisor

Wei Xu

Table of Contents

Page

List of Tables iv

List of Figures v

1. Introduction 1

    1.1 Reading RACES 1
    1.2 Organization of this Report 2

2. Design Requirements 3

    2.1 Users 3
    2.2 Use Cases 4
    2.3 High Level Requirements 4
        2.3.1 Read the Story to the Student 5
        2.3.2 Differentiate Between Correct and Incorrect Pronunciations 5

3. Design Decisions 6

    3.1 Language 6
    3.2 User Interface 6
    3.3 System Flow 6
    3.4 Text Normalization 8
    3.5 Tokenization 9
    3.6 Out-Of-Vocabulary Words 9

4. Pronunciation Prediction 11

    4.1 Related Work 11
    4.2 Data 12

4.3 Grapheme Model 13
4.4 Acoustic Model 15
4.5 Combined Model 17
4.6 Sounds-Like Spellings 20

5. Problems and Resolutions 21

5.1 Lexical Stress 21
5.2 Speech Synthesis of Baseform 22
5.3 Acoustic model 23

6. Summary and Future Work 25

References 27

List of Tables

Table		Page
3.1	NSW Transformations	8
4.1	Word Accuracy of Grapheme Model, Reading Races, varying n	15
4.2	Word Accuracy of Grapheme Model, $n = 5$	15
4.3	Word Accuracy of Acoustic and Grapheme Models, $n = 1$	17
4.4	Word Accuracy of Combined Model, FSG size = 40, $n = 5$	18
4.5	Word Accuracy on Reading Races, Varying λ	19
4.6	Word Accuracy of Interpolation Model, $\lambda = 0.7$	19
4.7	Sounds-Like Spellings	20
5.1	Lexical Stress Levels	21
5.2	Lexical Stress Results	22

List of Figures

Figure **Page**

3.1 System Flow Diagram 7

4.1 Grapheme Flow Diagram 14

4.2 PocketSphinx Speech Recognition Model 16

Chapter 1: Introduction

In this work I present additional functionality for Reading RACES, an oral reading fluency tutor. Specifically, I outline a modular system that enables the commercialization of the tutor by allowing educators to add new content. This system is mobile-compatible, fully supports all reading interventions, and eliminates the need for a specialized content developer role.

1.1 Reading RACES

Reading RACES is an oral reading fluency tutor designed to improve the ability of elementary students to read quickly and accurately. The program provides individualized attention to students while requiring minimal intervention from teachers, parents, and administrators. It guides students through a number of reading interventions: cold reads, Read to Me, Read Along, Listen to Me, and maze comprehension tests. Cold reads assess the student's performance on previously unseen stories to track the student's improvement in reading fluency over time [2]. During the Read to Me task, the student listens to the story while silently reading along with the text displayed on the screen. During the Read Along task, the student reads out loud along with the audio of the story. During the Listen to Me task, the student reads into a microphone, and the system tracks the student's progress through the story,

listening for hesitations and errors and providing extra practice and prompting as needed. The maze comprehension test evaluates the student's reading comprehension of the story. This test is comprised of the story text with every sixth or seventh word replaced with three multiple choice options. The student must use their understanding of the story they just read to select the best choice to complete the passage. For further details of the Reading RACES program, see [9].

The original research system is now being rewritten by a commercialization partner with the aim of making a market-ready mobile application to be deployed in schools. For this endeavor to be successful, teachers must be able to add new content to the system - either original stories or existing children's literature. To that end, I seek to build a mobile-compatible system that takes the text of a story and extracts all the necessary system components required for the Reading RACES interventions.

1.2 Organization of this Report

The remainder of this paper is organized as follows. Chapter 2 outlines the primary user roles, use cases, and high level requirements for this work. Chapter 3 describes the implementation choices along with some discussion of those choices and what alternatives were considered. Chapter 4 details the most interesting aspect of this work: how to automatically determine the pronunciation of out-of-vocabulary words. Methodology for data collection, system testing, and results are also presented. Chapter 5 discusses the challenges that surfaced during this work and their resolutions. Chapter 6 summarizes the project work and suggests improvements and potential areas for future study.

Chapter 2: Design Requirements

2.1 Users

The original Reading RACES design outlined four users: students, researchers, content developers, and teachers [9]. This work does not affect the student or researcher roles, but it does simplify the content developer role such that it can be subsumed by the teacher role.

Currently, content developers are responsible for creating and entering stories into the system. This occasionally requires encoding sounds-like pronunciations for out-of-vocabulary words and heteronyms. Content developers also select practice words, write the maze comprehension questions, and create audio recordings for the stories at various reading speeds.

With the changes in this work, content developers still need to select practice words and write the maze comprehension questions but no longer need to create the audio recordings. Except in rare instances, content developers no longer need to encode sounds-like pronunciations for words. These changes should allow teachers to fill the role of content developers by reducing the requirements of the content developer role to basic computer literacy skills and the ability to create test questions.

2.2 Use Cases

The system in this work has only one use case: a teacher adding a new story to the system. The user provides the story as a text file. One by one the system displays new vocabulary found in the story along with suggested pronunciations. To identify the correct pronunciation from a short list of suggestions generated by the system, the user can select a suggestion and hear it pronounced by the system's speech synthesis engine. Finally, the user enters the practice words and maze comprehension questions. As these last two tasks are unchanged and straight-forward, they will not be addressed further.

2.3 High Level Requirements

To perform all Reading RACES interventions for a student, the system must store the full text of a story for display on-screen, read the story to the student, and differentiate between correct and incorrect pronunciations for all words in the story. Storing and displaying the story text is trivial, but the other two functions pose more interesting problems and will be discussed below. The system should also support mobile devices as many school systems can more easily provide tablets for their students than the more expensive laptop or desktop systems.

The dependencies of the system include the story text and recordings of the out-of-vocabulary words - either pre-existing files or recorded by the user during the content creation process.

2.3.1 Read the Story to the Student

Reading the story to the student can be done in two ways: via a recording of a human reading the story or by synthesizing the text of the story. The current system uses audio recordings, but this approach requires that the content developer create multiple recordings at various speeds according to the student's reading ability. For this reason, synthesizing the text is preferable, but this requires knowing the correct pronunciations of all words.

2.3.2 Differentiate Between Correct and Incorrect Pronunciations

Differentiating between correct and incorrect pronunciations also requires knowing the correct pronunciation for all words in each story. Word pronunciation are stored as phonetic spellings, or baseforms, in a phonetic dictionary. The phonetic dictionary included in the speech recognition system is extensive, but out-of-vocabulary words will inevitably appear. This work attempts to predict the phonetic spelling of out-of-vocabulary words without the user needing any specialized knowledge of linguistics or phonetics.

Chapter 3: Design Decisions

3.1 Language

As the project is being reimplemented for commercialization, there are few hard requirements that would limit the choice of programming language for this content authoring module. Python was chosen for a number of reasons: it is portable due to being an interpreted language, it is easily readable and ideal for prototyping work due to its expressiveness and dynamic typing, and there are a wide range of scientific and natural language processing toolkits available in Python, which will prove useful.

3.2 User Interface

The user interface is implemented as a simple command line system sufficient for a proof-of-concept. This functionality is contained within a modular component that can easily be swapped out with a new implementation should a graphic user interface be desired.

3.3 System Flow

Figure 3.1 shows the system flow. Non-standard word transformation, tokenization, and out-of-vocabulary identification will be discussed in the sections that follow.

The pronunciation prediction models and creative misspellings will be discussed in Chapter 4.

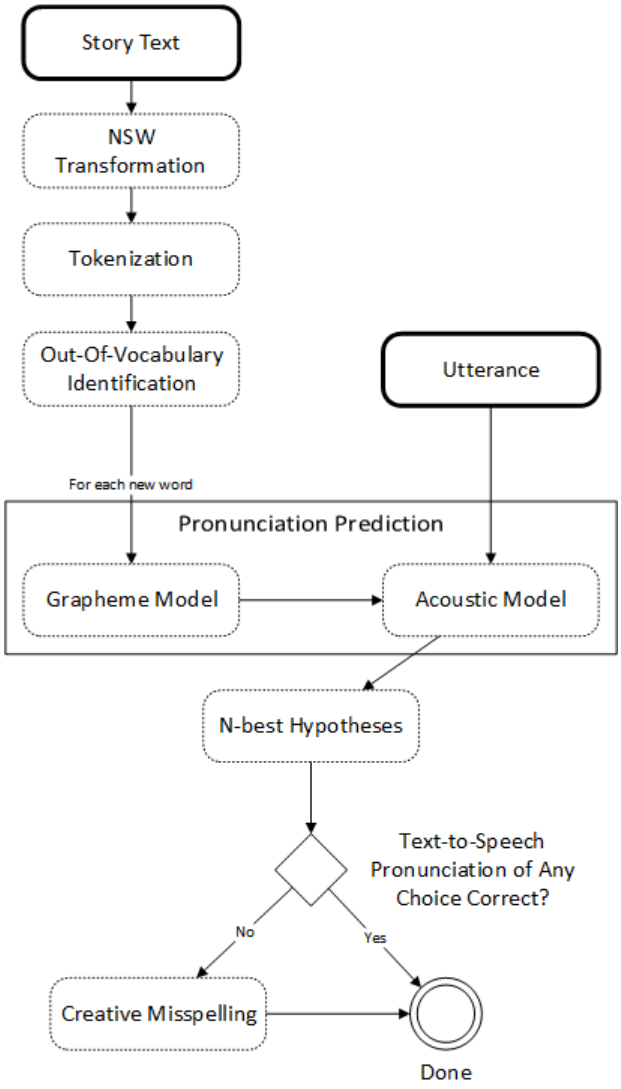

Figure 3.1: System Flow Diagram

3.4 Text Normalization

Speech synthesis requires that the system recognize and transform non-standard words (NSWs) into their full form. NSWs include numbers, dates, times, currency amounts, abbreviations, and acronyms. See Table 3.1 for examples of NSWs and their transformations.

NSW Transformations		
Numbers	1023 1.13	one thousand twenty three one point one three
Dates	1/1/13 2-3-07	January first twenty thirteen February third two thousand seven
Currency Amounts	$12.98 50¢	twelve dollars and ninety eight cents fifty cents
Times	10:05 AM 13:48:02	ten o-five A M thirteen hours forty eight minutes and two seconds
Acronyms	NASA DMV	N AA S AH D IY EH M V IY
Abbreviations	appt sgt	appointment sergeant

Table 3.1: NSW Transformations

Transforming numbers, dates, times, and currency amounts is done using the Lextools project developed for the CLSP Summer Workshop at Johns Hopkins University [1]. Lextools was developed to address a lack of a systematic approach to text normalization and significantly out-performed similar existing tools [13].

Pronunciations of many common acronyms are already defined in the phonetic dictionary, and new ones can be added using the approach for new vocabulary discussed in Chapter 4. Abbreviations, however, will not be handled in this work and must be spelled out in full by the user.

3.5 Tokenization

The text is parsed using the WhitespaceTokenizer class provided by the Natural Language Toolkit (NLTK), a Python framework for natural language processing. WhitespaceTokenizer leaves leading and trailing punctuation attached to word tokens, including periods at the end of sentences and opening and closing quotation marks around speech. These characters are removed by stripping non-alphanumeric characters from the beginning and end of tokens. The final step in tokenization is an uppercase transform as some of the models to be discussed in later sections are case-sensitive.

3.6 Out-Of-Vocabulary Words

The final step is handling out-of-vocabulary words. The system scans through the tokenized story text to find any words that are not defined in the phonetic dictionary and generates a number of suggested pronunciations for each word. The number of suggestions is customizable with a default of 10. The method for generating suggestions will be discussed thoroughly in Chapter 4.

Users can identify the correct pronunciation from the list by selecting each option in turn and hearing it pronounced by a speech synthesis system. The Festival Speech Synthesis System from the University of Edinburgh is used to generate this speech.

Each suggestion is also labelled with a sounds-like spelling, e.g. the word builder with pronunciation 'B IH L D ER' is labelled 'bilder'. The method for generating sounds-like spellings will also be discussed in Chapter 4 as it builds on the techniques used for pronunciation prediction. If the correct pronunciation is not among the list of suggestions, users can enter their own sounds-like spelling to generate the correct pronunciation.

Once the pronunciations of all out-of-vocabulary words have been determined, the system will use speech synthesis to read the entire story to the user. At this time, any improperly identified heteronyms can be corrected by choosing from a list of known pronunciations or by adding a new pronunciation through the pronunciation prediction method.

Chapter 4: Pronunciation Prediction

This chapter discusses the most challenging piece of this work: how to determine the correct phonetic spelling of out-of-vocabulary words. As the user will have no specialized knowledge of linguistics, this must be done using available information, such as the spelling of the word. The user can also be expected to know how to pronounce the word even if they are unable to annotate this pronunciation using phonemes [7]. Thus, a solution must be found that uses only these two pieces of information.

4.1 Related Work

The problem of pronunciation prediction has been studied reasonably well, as least for the English language. Many approaches have used only the spelling of the word to determine its pronunciation. For general English vocabulary, Pagel et al. achieved 62.79% word accuracy and 87.84% phone accuracy using letter-to-sound rules [12]. For the harder problem of proper nouns, which tend to deviate from typical letter to sound rules, Deshmukh et al. achieved word accuracy of 36.05% using a Boltzmann machine [3]. Ngan et al. improved on this in [10], achieving 54.5% word accuracy.

Results of up to 97.1% phone accuracy have been achieved on general vocabulary by Lucassen and Mercer in [8] by using both the spelling and a sample utterance.

This approach takes advantage of the ability of spelling-based or grapheme models to correctly predict consonant phonemes and the ability of acoustic models to correctly predict vowel phonemes.

A similar approach to [8] is taken here with the aim of achieving similar results by building a grapheme model and combining it with an acoustic model to do pronunciation prediction.

4.2 Data

To evaluate this approach, a set of word spellings, pronunciations, and utterances is needed. The CMUdict phonetic dictionary is used for the word spellings and pronunciations [14]. The latest version of CMUdict contains almost 134,000 entries of American English words with their phonetic pronunciations. The pronunciations use the ARPAbet phone set, which consists of 39 phonemes designed for representing American English.

Word recordings are available from a number of resources, largely as language-learning aids for non-native English speakers. A script was written to search Google, Wiktionary, and the Shtooka Project [4] for the words contained in CMUDict and label them with the corresponding pronunciations. Exactly 31,373 recordings were acquired this way; these recordings will be referred to as the full web recordings. The 2,573 recordings for homographs and words with multiple pronunciations were manually labelled with the correct pronunciations. These are referred to as the duplicate web recordings and they are inherently more challenging than the full web recordings. The remaining recordings were assumed to be pronounced as expected, although some of them are likely to be incorrectly labelled. These recordings are also

not representative of typical out-of-vocabulary words, as a majority of them are common English words. The speech recognition engine used by Reading RACES contains pronunciations for most common English words, so typical out-of-vocabulary words are more likely to be proper nouns, new words and slang, and and non-English words.

To supplement the web recordings with a more representative set of out-of-vocabulary words, a custom dataset was built using words from the existing Reading RACES stories. These stories contain 2,106 unique vocabulary words, 49 of which are not contained within CMUdict. Most of the new words are proper nouns, newly-coined words, such as 'swag', and non-English words, such as 'djembe'. These words represent a more realistic selection of the types of out-of-vocabulary words that would be contained in new stories. I annotated the phonetic pronunciations for these words and created recordings for each. These recordings are referred to as the Reading RACES dataset.

Train and development sets for building and tuning the models are created by removing the web recording words from CMUdict and dividing the remaining words into a 60-40 train-development split. The web recordings, web duplicate recordings, and Reading RACES recordings are used as three separate test sets.

4.3 Grapheme Model

The grapheme-based model is built using Phonetisaurus, a WFST-based system for grapheme-to-phoneme conversion [11]. The dictionary entries were aligned using a many-to-many alignment method provided by Phonetisaurus, which iterates over all possible alignments of the graphemes comprising the word's spelling to the phonemes contained in the word's pronunciation. This process uses epsilon insertions to support

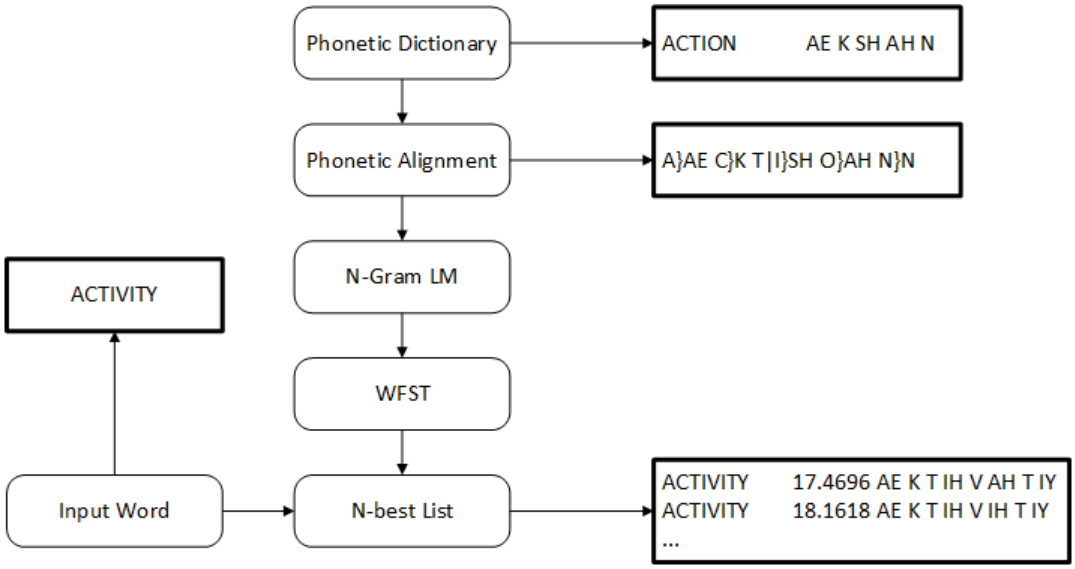

Figure 4.1: Grapheme Flow Diagram

both graphemes that generate multiple phonemes and digraphs that generate a single phoneme.

A trigram language model with Good-Turing discounting is created from the resulting sequences of aligned grapheme-phoneme pairs using the SRI Language Modeling toolkit [6]. A WFSA is built using the the language model and then the WFSA is converted into a WFST by splitting the grapheme-phoneme pairs into input and output arc labels.

To generate pronunciations for a new word, the word is represented as a WFSA with each arc between sequential states labelled with the corresponding grapheme. This WFSA is composed with the WFST created from the language model, and an n-best list of pronunciations can be extracted by finding the n shortest paths through the resulting graph. This model flow is shown in Figure 4.1.

n	Accuracy (%)
1	40.81
5	48.30
10	54.99
40	61.22

Table 4.1: Word Accuracy of Grapheme Model, Reading Races, varying n

Dataset	Accuracy (%)
Web - All	77.73
Web - Duplicate	48.30
Reading Races	61.22

Table 4.2: Word Accuracy of Grapheme Model, $n = 5$

One factor to be considered is how many options a user can effectively sort through. As Table 4.1 shows, accuracy increases as the number of hypotheses evaluated grows, but a list of 40 options is too cumbersome for a user. An output $n = 5$ is used for the rest of this work as it is a reasonable compromise between accuracy and manageability for a user to sort through. Table 4.2 shows the accuracy of the grapheme model on all three test sets with this value of n.

4.4 Acoustic Model

The acoustic model is built using PocketSphinx, a member of Carnegie Mellon's Sphinx family of speech recognition engines that is optimized for mobile performance [15]. PocketSphinx determines the most likely phone sequence $\hat{W}$ given an acoustic observation O by finding the one with the maximum relative probability as such:

$$\begin{aligned}\hat{W} &= \arg\max_{W} P(W|O) \\ &= \arg\max_{W} \frac{P(O|W)P(W)}{P(O)} \\ &= \arg\max_{W} P(O|W)P(W)\end{aligned} \tag{4.1}$$

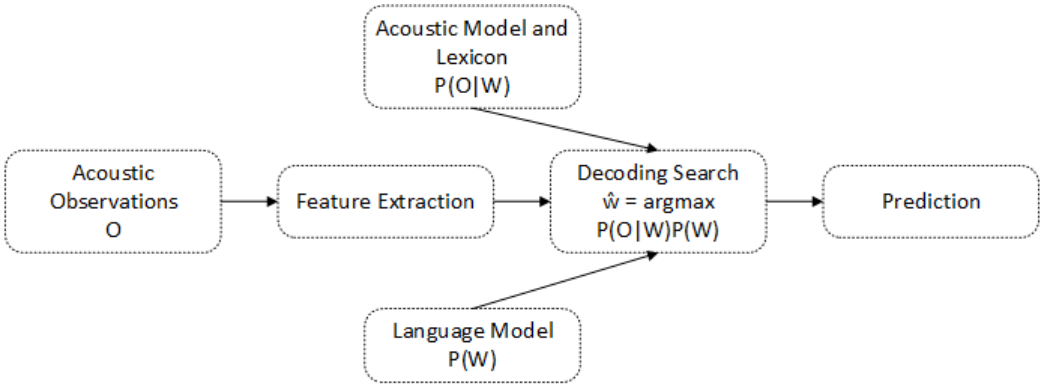

Figure 4.2: PocketSphinx Speech Recognition Model

Figure 4.2 shows the components of the speech recognition system used to calculate these probabilities. For phone recognition, $P(O|W)$ is obtained from the acoustic model and a phone dictionary. The acoustic model used here is the default PocketSphinx English model, a robust speaker-independent model trained on 1600 utterances from multiple speakers. The phone dictionary contains a mapping of all the phones in the phone set used to train the acoustic model with each phone mapped to itself. The probability of a phone sequence, $P(W)$, is obtained from a phonetic language model that is provided by PocketSphinx, along with the acoustic model. The acoustic observations, O, are extracted from the audio signal by dividing the signal into frames and calculating the power per frame along with 13 Mel-frequency cepstral coefficients, deltas, and delta-deltas.

Dataset	Acoustic Accuracy (%)	Grapheme Accuracy (%)
Web - All	5.73	51.77
Web - Duplicate	5.40	20.59
Reading Races	0.00	40.81

Table 4.3: Word Accuracy of Acoustic and Grapheme Models, $n = 1$

PocketSphinx finds the most likely phone sequences via Viterbi search with four supported search modes: n-gram, grammar, all-phone, and keyword. The all-phone mode is used for the acoustic model as it allows for phone-level hypotheses rather than word-level hypotheses as generated by the n-gram or keyword search modes. Unfortunately, this mode is not compatible with producing n-best lists, so the acoustic model will be limited when used independently of the grapheme model. Table 4.3 shows the performance of the acoustic model along with the results for grapheme model with output $n = 1$ for comparison.

4.5 Combined Model

To combine the grapheme and acoustic models, the PocketSphinx decoder search mode is changed from all-phone to grammar. The grammar mode takes an input regular grammar and supports generating n-best lists. In this mode, the grammar is used in conjunction with the language model to determine $P(W)$. Here, the grammar is constructed from the n-best list produced by the grapheme model with all hypotheses being represented as equally-weighted rules. Then the acoustic model is run as described in Section 4.4. This effectively re-ranks a large n-best list from Phonetisaurus and outputs a shorter list that can be provided to the user. Table 4.4

shows the results of this model on the three datasets with a grammar containing 40 pronunciations and a final output size of 5.

Dataset	Accuracy (%)
Web - All	70.16
Web - Duplicate	49.81
Reading Races	53.06

Table 4.4: Word Accuracy of Combined Model, FSG size = 40, $n = 5$

To improve this model, a second re-ranking is obtained by linearly interpolating the scores for each hypothesis as produced by the grapheme and acoustic models. The acoustic score s_c represents the acoustic log likelihood of the predicted phone sequence. The original grapheme score w_g represents the path weight of the hypothesis through the composed WFST. To convert the grapheme scores to probabilities in log likehood space, a new score $s_{g,i}$ for the i^{th} hypothesis is calculated as:

$$s_{g,i} = log_{1.001}\left[\frac{(w_{g,i})^{-1}}{\sum_{j=1}^{n}(w_{g,j})^{-1}}\right] \tag{4.2}$$

These scores are combined to produce an output score s_o:

$$s_o = \lambda * s_g + (1 - \lambda) * s_c \tag{4.3}$$

To determine an optimal value for λ, Reading Races was used as a development set, producing the results shown in Figure 4.5. This dataset is prone to overfitting due to its size, so λ was set to a value of 0.7 rather than either of the true maximas at $\lambda \in [0.4, 0.55]$ or $\lambda = 0.9$. The results for the interpolation-based combined model using this value of λ are shown in Table 4.6.

Lambda	Accuracy (%)
0.10	51.02
0.15	55.10
0.20	59.18
0.25	59.18
0.30	59.18
0.35	59.18
0.40	61.22
0.45	61.22
0.50	61.22
0.55	61.22
0.60	59.18
0.65	59.18
0.70	59.18
0.75	59.18
0.80	59.18
0.85	59.18
0.90	61.22

Table 4.5: Word Accuracy on Reading Races, Varying λ

Dataset	Accuracy (%)
Web - All	82.70
Web - Combined	55.88

Table 4.6: Word Accuracy of Interpolation Model, $\lambda = 0.7$

4.6 Sounds-Like Spellings

As mentioned in Section 3.6, the system generates sounds-like spellings for each hypothesis. This helps users more easily identify the correct pronunciation from a list and trains users to enter their own sounds-like spellings that will be predictably pronounced by the system. Generating sounds-like spellings is done by inverting the WFST discussed in Section 4.3 and composing it with a WFSA built from the phonetic spelling. The sounds-like spelling is the shortest path through the resulting graph. Examples of sounds-like spellings generated by this model are shown in Table 4.7 along with the corresponding pronunciations.

True Spelling	Pronunciation	Sounds-Like Spelling
aunt	AE N T	ant
aunt	AO N T	aunt
builder	B IH L D ER	bilder
carrying	K AE R IY IH NG	carying
enormous	IH N AO R M AH S	enormus
lilies	L IH L IY Z	lilli's
pets	P EH T S	pets
poured	P AO R D	pord
shouldn't	SH UH D AH N T	shoodent
younger	Y AH N G ER	yunger

Table 4.7: Sounds-Like Spellings

Chapter 5: Problems and Resolutions

A number of challenges presented themselves during the implementation of this work. In this chapter, these challenges and their solutions will be discussed.

5.1 Lexical Stress

The first issue that presented itself was a mismatch in phone sets between CMUdict and PocketSphinx. Both platforms use ARPAbet, but CMUdict marks vowel phones with lexical stress levels and the PocketSphinx acoustic model does not. Lexical stress levels indicate the level of stress given to vowels and can vary between 0 and 2, as shown in Table 5.1. The lexical stress can help differentiate words which have similar phones but different stress patterns. For example, the noun object is pronounced 'AH0 B JH EH1 K T' while the verb object is pronounced 'AA1 B JK EH0 K T'. However, as the default acoustic model that ships with PocketSphinx does not differentiate between lexical stress levels, the stress levels had to be removed.

Level	Description
0	No stress
1	Primary stress
2	Secondary stress

Table 5.1: Lexical Stress Levels

The question then became at which stage the lexical stress markers should be removed. Including the additional information in the training data of the grapheme model might help increase the accuracy. To determine this, three versions of the grapheme model were built. The first version was trained on data containing all three levels of lexical stress and the stress markers were only removed when comparing to the desired results. The second version collapsed the primary and secondary stress levels in the input training data, keeping the unstressed markers unchanged until testing. The third version collapsed all lexical stress levels in both the input training and test data. As Table 5.2 shows, there is not a significant difference between the three versions of the model, so all lexical stress levels are collapsed prior to training the grapheme model.

	Collapse none	Collapse 1 and 2	Collapse all
N = 5	0.860	0.860	0.856

Table 5.2: Lexical Stress Results

5.2 Speech Synthesis of Baseform

Another challenge that presented itself was how to instruct Festival to pronounce a baseform once a phonetic spelling is generated. Festival allows users to specify pronunciations via the Sable markup language. However, the options for this tag allow the user to either provide the pronunciation via the IPA phone set, specify the language of origin, or enter a substitution similar to the sounds-like pronunciations discussed in section 4.6. Festival does not currently support the IPA specification,

and the language of origin is of limited usefulness for this problem. Entering the sounds-like spelling of the word using the substitution option consistently fails to produce the correct pronunciation.

Festival does allow the user to add a pronunciation directly to the current lexicon. The lexicon is composed of three major parts: the addenda, the compiled lexicon, and the unknown word method. The addenda is a mapping of unique word and part-of-speech tuples to pronunciations, and it is searched before the other components of the lexicon. The compiled section is a full listing of words and their pronunciations in the lexicon. The unknown word method handles guessing the pronunciation of out-of-vocabulary words, which typically employs letter to sound rules. Adding words to the compiled lexicon requires re-compiling, but adding entries to the addenda can be done on the fly so this is the approach used here.

5.3 Acoustic model

Originally, I hoped to build my own acoustic model that could be customized for this problem and achieve higher accuracy by using an existing set of story recordings for the original stories. For the 52 existing stories, there are a total of 120 recording files including multiple recordings per story at different reading speeds. These recordings add up to nearly 6 hours of audio data all from the same speaker.

CMU has produced an acoustic model training tool, sphinxtrain, that allows a user to create an acoustic model compatible with PocketSphinx. Training an acoustic model using sphinxtrain requires a phonetic dictionary, phone set definition, language model, filler dictionary, and training and test sets of recording files and transcriptions. A phonetic dictionary and phone set definition were already available, and a language

model was trained using SRILM using the stories' text. The filler dictionary was empty as the recordings were clean with no stammering or filler words. The recording transcriptions already existed in the form of the story text, so the recordings and transcriptions were randomly sorted into training and test sets and the model was trained.

Unfortunately, the results were disappointing even on a test set with the same speaker used for the training set. This is largely due to an insufficient amount of training data. PocketSphinx's recommendations for building speaker-independent dictation-capable models are at least 50 hours of recordings from at least 200 speakers, much more than was available here [15]. Because of these data limitations, the default PocketSphinx model was used instead.

Chapter 6: Summary and Future Work

In this work, a system was built to allow new content to be added to Reading RACES in order to support the commercialization goal of the project. This involved moving towards a mobile platform and reducing the existing user roles by allowing teachers to perform tasks previously done by specialized content developers. The system framework was built, including modules for text normalization, tokenization, and tagging of out-of-vocabulary words. A model for predicting the pronunciation of out-of-vocabulary words was developed with 82.70% word accuracy on general English vocabulary and 59.18% accuracy on representative new words. A number of outstanding issues and interesting problems remain in this work. These include improvement of the acoustic data and predictive models and automation of other content authoring tasks.

It could be worthwhile to re-examine this approach using a set of word recordings of higher quality. The Reading RACES recordings are fairly noisy with audible keyboard sounds in the audio, and closer inspection of the acoustic hypotheses for individual recordings reveals that some phones in the hypotheses correspond to the sound of keystrokes rather than speech. While there are a number of high-quality

annotated speech corpora that are freely available, there are none that provide annotated recordings at a single word basis. The creation of such a resource would enable better evaluation of this and other approaches to pronunciation prediction.

Additionally, PocketSphinx was chosen for this work as it is a speech recognition engine optimized for mobile. It does not represent the state of the art in speech recognition, nor is it designed for phoneme recognition. A comparison of open-source speech recognition models found the Kaldi system to have the lowest word error rates on several populare speech corpora [5]. Even within the Sphinx family of speech recognizers, Sphinx4 is a more robust and configurable option that might better fit this task.

Another interesting problem for future investigation would be the automatic generation of other content authoring tasks such as selection of practice words and creation of maze reading comprehension questions. The system built here requires that the content developer manually create and enter the questions, but an automated approach could be developed to reasonably handle these tasks.

References

[1] A. Black, S. Chen, S. Kumar, M. Ostendorf, C. Richards, and R. Sproat. Lextools. http://festvox.org/nsw/, 1999.

[2] M. R. Council, G. Cartledge, D. Green, M. Barber, and R. Gardner. Reducing risk through a supplementary reading intervention: A case study of first- and second-grade urban students. *Behavioral Disorders*, 41(4):241–257, Aug 2016.

[3] N. Deshmukh, J. Ngan, J. Hamaker, and J. Picone. An advanced system to generate pronunciations of proper nouns. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-97)*, volume 2, pages 1467–1470, 1997.

[4] G. Ducrue and J. Franck. Shtooka Project. http://www.shtooka.net/.

[5] C. Gaida, P. Lange, R. Petrick, P. Proba, A. Malatawy, and D. Suendermann-Oeft. Comparing open-source speech recognition toolkits. *Technical report, Project OASIS*, 2014.

[6] SRI International. SRI Language Modeling Toolkit. http://www.speech.sri.com/projects/srilm/, 2016.

[7] J. Lucassen. Discovering phonemic base forms automatically: an information theoretic approach. Master's thesis, Massachusetts Institute of Technology, 1983.

[8] J. Lucassen and R. Mercer. An information theoretic approach to the automatic determination of phonemic baseforms. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '84)*, volume 9, pages 304–307, 1984.

[9] T. D. Lynch. The design and development of reading races. *The Ohio State University*, Jan 2016.

[10] J. Ngan, A. Ganapathiraju, and J. Picone. Improved surname pronunciations using decision trees. In *ICSLP*, 1998.

[11] J. Novak. Phonetisaurus G2P. https://github.com/AdolfVonKleist/Phonetisaurus.

[12] V. Pagel, K. Lenzo, and A. Black. Letter to sound rules for accented lexicon compression. In *International Conference on Spoken Language Processing (ICSLP-98)*, 1998.

[13] R. Sproat, A. W. Black, S. Chen, S. Kumar, M. Ostendorf, and C. Richards. Normalization of non-standard words. *Computer speech & language*, 15(3):287–333, 2001.

[14] Carnegie Mellon University. CMU Pronouncing Dictionary, version 0.7b. http://www.speech.cs.cmu.edu/cgi-bin/cmudict, Nov 2014.

[15] Carnegie Mellon University. PocketSphinx Speech Recognition Engine. https://github.com/cmusphinx/pocketsphinx, 2016.