Tabular Priority Control Design:
An Automated Transcription Comparison Framework for Phonetics

by

Ishaan Mishra

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 1, 2021

Keywords: Phonetics, Algorithms

Approved by

Dr. Cheryl D. Seals, Chair, Professor of Computer Science and Software Engineering
Dr. Dallin J. Bailey, Assistant Professor of Speech, Language, and Hearing Sciences
Dr. Marisha S. Atkins, Assistant Professor of Speech, Language, and Hearing Sciences

ABSTRACT

Phonetics is the study and systematic classification of speech sounds. It is a discipline of linguistics that deals with the science of sound production, transmission, and reception by humans. Phonetic transcription is the visual depiction of speech sounds utilizing written symbols, IPA (International Phonetic Alphabet) being the most common standard. There have been several computational tools in the past for comparing phonetic transcriptions, and some are either difficult to access and integrate into research, or the learning curve is too high for regular users. This thesis describes the development of an online computational tool, called APTct (Automated Phonetic Transcription comparison tool), with unique features, including a comparison visualization module for optimal alignments and scoring operations. The main highlight of this research is creating a unique tabular priority control system designed to control the path taken by the algorithm while performing phonetic transcription comparisons. This system is very versatile in its implementation, and it applies to several other tabular-style and dynamic programming-based algorithms.

APTct underwent rigorous analysis for precise implementation of phonetic and algorithmic principles. Functional testing of results showed a very high degree of accuracy compared with results obtained from an expert's hand scoring of phonetic transcription comparisons. APTct is an innovative and validated web-based tool, freely accessible to the public. It is available online at the following URL: https://aptct.auburn.edu.

ACKNOWLEDGMENTS

I am extremely grateful to Dr. Cheryl Seals for her immeasurable support and expert guidance during my master's degree. I would not have been able to make it this far without her. I would also like to thank Dr. Dallin Bailey and Dr. Marisha Speights Atkins for their incredible cooperation and their profound trust in me throughout the process. A huge shout out to Abigail Bennett for her help with testing.

I am greatly indebted to my mother for being my voice of reason and my father for his endless love. Lastly, I would like to thank me for all the hard work and for never giving up.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

From the time humanity was born on Earth, social communication has undoubtedly been the most crucial skill for human beings. To ensure their survival, they had to collaborate continuously, which would not have been possible without a communication modality. Communication forms the basis for the evolution of language conveyed through speech, writing, and gesture. Our thoughts as human beings are converted into speech by default whenever interaction is required. This communication via sounds is made possible via a complex orchestration of lips, tongues, and other vocal components of our mouth.

Linguistics is a branch of science that deals with studying and analyzing language and the factors affecting it [1]. Linguistics comprises various branches like Phonology, Morphology, Semantics, Pragmatics, etc. One branch, in particular, Phonetics, analyzes how sounds are pronounced and recognized by human beings. It deals with the classification of speech sounds. These speech sounds denote the written form as phonetic symbols and organized into a format known as the International Phonetic Alphabet, or IPA [2]. IPA is thus a systematic representation of speech sounds in textual format and demonstrates how something is pronounced. Also, IPA is not associated with any language, like English or French, for instance. It applies to all languages since it represents pronunciations of speech sounds. Thus, all spoken sounds, regardless of the language, can be transcribed into a written form via the application of phonetics.

Phonetic transcription is the process of visually representing speech sounds via symbols (IPA is the most common standard). It has various use cases. In linguistics, for example, linguists

(for their comprehension) can record a person's verbal expression for research analysis and other purposes. This written document can be preserved and shared with others. In speech-language pathology, clinicians or speech-language pathologists can observe and catalog irregularities in a patient's speech production if it is disordered [3]. These transcriptions are created by converting an individual's speech into phonetic symbols and code, with IPA being the most preferred format.

There can be several reasons for these transcriptions to be compared with each other. For example, a clinician can measure the difference between previous and current records of a patient's speech development to track their progress over time. Phonetic transcription comparison also finds uses in linguistics to discover semantic associations between languages [4], syntactical relationships, etc. Furthermore, phonetic courses in universities and institutions can also benefit from phonetic transcription comparisons and comparing a student's transcription to an expert's transcription or a baseline for scoring.

There are several valuable purposes for phonetic transcription comparisons, as mentioned above. However, the problem is that the process itself can be pretty tedious and time-consuming, especially for lengthier and more advanced, detailed transcriptions. If an instructor teaches a phonetics course, they will have to compare each student's answer against the standard transcription, which can be a laborious task. Also, comparisons can be susceptible to inconsistencies due to different perceptions and the natural subjectivity of transcription evaluators.

Dr. Dallin J. Bailey and Dr. Marisha Speights Atkins, from Auburn University, came up with the solution to the difficulties of hand comparisons of transcriptions by proposing the

development of a tool that involves utilizing computer algorithms for automating and operationalizing the process of transcription comparison. Thus, they suggested creating a free, user-friendly, readily available online phonetic transcription comparison web application, called the Automated Phonetic Transcription Comparison Tool (APTct).

This thesis aims to document the development and formation of the Automated Phonetic Transcription Comparison Tool in its entirety. The research goal was to develop a more efficient and accurate method of transcription comparative analysis by creating a unique tabular priority control design. This algorithmic system performs phonetics comparisons with a higher degree of accuracy than prior work. Being very versatile, it is also applicable to several other tabular-style and dynamic programming-based algorithms. Discussions on its performance, future work, and potential applications are included.

CHAPTER 2: LITERATURE REVIEW

This section reviews literature and research work related to theoretical and practical aspects of APTct and supports some of the decisions behind the creation process of the tool.

2.1 User Interface Design

2.1.1 The Role of Aesthetics in Web Design

Web sites are increasingly taking over media as the preferred choice for news, information, entertainment, education, etc. In the current generation, visual appearance plays quite an important role. This research aims to discuss the four essential areas in which aesthetics are vital for the successful design of a website: support for core functionality and subject, target audience's preference, generation of required perception for the sender, the primary genre of the website. Thorlacius (2007) [5], talks about the importance of graphic design elements and how they affect the appearance of websites. Visual effects play an essential role in the formation of aesthetic experiences. However, a balance has to be struck between visual elements and site functionality. To make a website user-friendly and comply with aspects of human-computer interaction, the main objective should be to create an experience where users can quickly and efficiently obtain the information they desire without delays.

In this research, there is a discussion on the historical aspect of websites through the years, and the relationship between functionality and aesthetics is debated. Jakob Nielsen, a critical researcher and contributor to human-computer interaction, argues against David Siegel, a proponent of aesthetic websites, that functionality trumps aesthetics and that less is more when it

comes to aesthetics. This is further corroborated when after the dot-com crisis, organizations began to cut down on the enormous costs of making websites with advanced aesthetic effects, resulting in a standard but more functional layout.

Thorlacius (2007) then concludes the research with a detailed and in-depth discussion on how visual aesthetics play a role in the development of websites and how the most successful websites are the sites that are created following the following four areas: sender image, functionality, genre, and target audience. The sender's perception should be conveyed on the website via linguistic, functional, and aesthetic effects. Web sites have been more user-friendly when there is more focus on the content and functional aspects and when there is support for navigation and interaction functions. Aesthetic effects must also be tuned to the primary genre of the website so that the design is more relevant, causing information extraction to be efficient. Finally, aesthetic effects should be conformed to the target audience to differentiate visual presentation based on demographics.

2.1.2 Websites that Satisfy Users: A Theoretical Framework for Web User Interface Design and Evaluation

Web designers have primarily been focused on the functional aspects of websites. However, motivational issues of web design and web user satisfaction studies have garnered little attention. This research article investigates features in the web domain that contribute to user satisfaction with a web interface and defines a conceptual foundation. The work of Zhang et al. (1999) [6] results in the identification of web design components that augmented the chances of user satisfaction and repeated visits to the website. In the study, a theoretical framework is

presented, consisting of three components. A user's satisfaction or dissatisfaction with a web interface results from the interaction between these three components (see Figure 1).

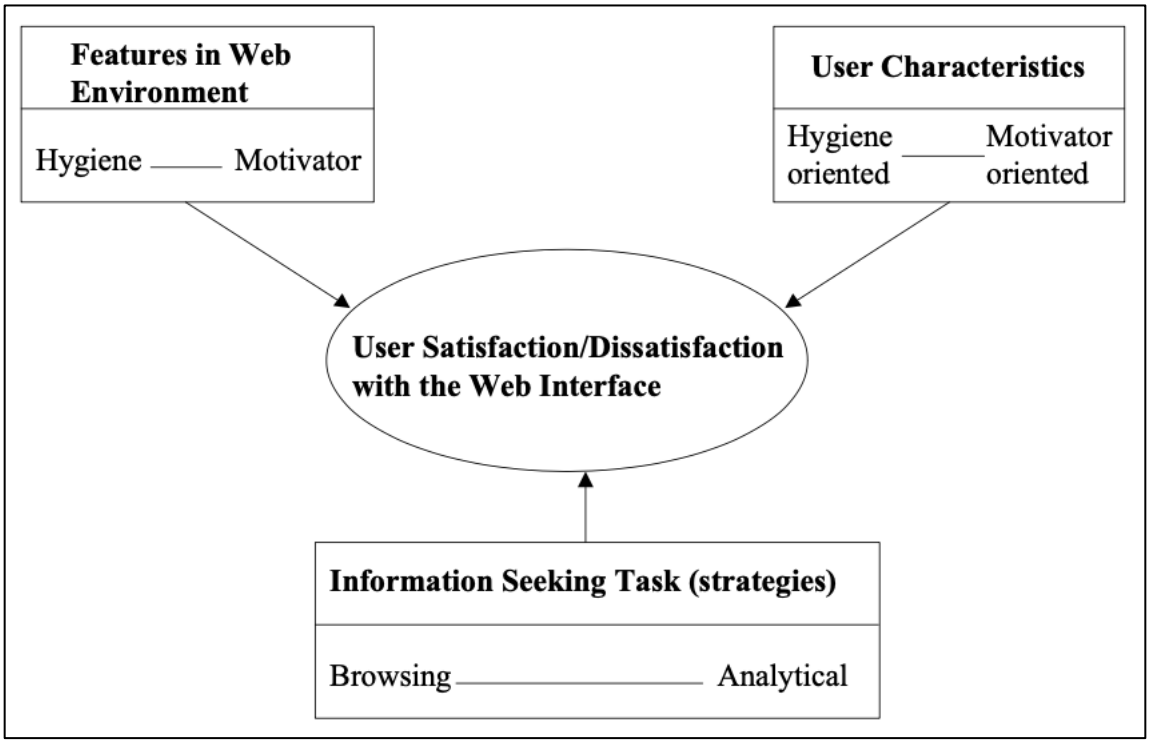

Figure 1

The first component deals with features in a web system that are either necessary (hygiene) or satisfactory (motivation). The second component defines how users interact with information on a website. They can either be browsing informally or can be examining information analytically. The third component focuses on individual psychological characteristics, such as empowerment, locus of control, etc., and how they impact user satisfaction with the web interface.

The reviewed literature in [6] focused more on the relationship between human behavior and information systems. Herzberg's two-factor theory application to the web environment [7] is

tabulated by the authors and categorized in a very detailed fashion. In this study, the primary research hypotheses are that the subjects, if dissatisfied, are due to the absence of hygiene features and, if satisfied, are due to the presence of both motivation and hygiene features. Also, there are certain features (hygiene and motivation) for a website more important than others. Ultimately, the authors conclude their research by stating that motivational factors in a competitive web environment can provide an edge and may prove to be an aggressive advantage.

2.1.3 Responsive Design and Development: Methods, Technologies and Current Issues

In web development, responsive design is crucial to serving various devices used for web browsing. Nevertheless, the application of responsive design to already existing websites often requires quite a bit of reengineering because of the underlying fluid grid concept of responsiveness. Additionally, responsive design and its applications are currently restricted to desktop-to-mobile adaptation. This instructional study introduces the main ideas behind a responsive design with emphasis on methods and technologies. Nebeling et al. (2013) [8] use previous research and underline quite a few limitations of the original approach, showing how responsive concepts can be extended to adapt to various viewing conditions, including large screens and devices with a touch screen.

Application developers have to deal with an increased range of devices and a variety of interface characteristics. These characteristics not only include screen size and resolution but also supported input and output modalities. On one end of the spectrum, there are small devices like mobile phones. On the other end, there are large interfaces like desktop computers. Tablets, slates and notebooks lie in the middle. Because of the rapid evolution and increased diversity of devices

used for web browsing, design strategies have to be rethought. Over the years, graceful degradation and progressive enhancement have played a significant role. However, given the diversification of devices, these two strategies are not feasible anymore. Therefore, responsive design is the new design trend. It refers to developing a web interface layout on fluid grids so that the website can dynamically adapt to diverse viewing environments.

The authors divide the tutorial study into two portions. The first portion provides an elementary introduction to responsive web design and its significant ongoing trends. New features of HTML5 and CSS3 are discussed, and the benefits and limitations of other context-aware adaptation approaches. The second portion gives a brief overview of the authors' systematic ways to tackle design issues. Methods and tools presented in the second portion include more than just the principles of responsive design. It involves technological and methodological points of view, in addition to different scenarios and use cases.

2.1.4 Responsive Web Design Techniques

Bader and Hammouri (2016) [9] presented comprehensive research on different responsive website design techniques that can be adapted to various devices and technologies while saving valuable time and effort required by programmers to maintain and edit the website. Although the demonstration of techniques is in ASP.net programming language, the ideas apply to other development languages and environments too. Web designers have to ensure that the users accessing their websites have the best possible experience, regardless of the device being used. More than the modality of access, the users are concerned with how quickly and comfortably they can get to content relevant to them. RWD, or Responsive Web Design, comprises a set of

techniques useful in developing one single website capable of resizing itself on various screen sizes, resolutions, and orientations. In the study, three standard techniques to produce a responsive website design are described. They are as follows –

- Fluid grid layouts or relative-based grids
- Images and media which are flexible in their sizes
- Media queries and screen resolution

In the fluid grid layout technique, the web page dimensions are set in proportion to the maximum size of the page. This means that percentages, rather than fixed pixel values, are used to set sizes and other units. For example, if the width of a header is specified as 100%, that means it is 100% of the maximum screen size, let us say 1024 pixels. Similarly, if a menu is specified as 25%, it is 25% of 1024 pixels, i.e., 256 pixels. Similar to the fluid grid layout technique is the usage of relative dimensions for images and media. This ensures that they automatically resize/crop with regards to the viewing size and resolution. For example, if the screen size is 1024 pixels, to have an image that is half the screen size, 50% can be set as the width instead of hardcoding a value of 512 pixels. The third RWD technique demonstrated in the paper is the use of media queries and screen resolutions. It gives the website designer the ability to specify multiple fixed versions of an element for different resolution ranges. CSS3 media queries are so powerful that they can change the entire website experience based on the different sizes of the viewing device. For example, the designer can specify that the height of an image should be 200px if the browser has a resolution width from 401px to 799px; otherwise, the height of that image should be 400px for higher browser resolution widths. The authors provide supporting HTML5 and CSS3 code for all these responsive techniques. Another interesting point of discussion highlights server-side responsive web design, used for some browsers and devices not compatible with CSS3. In this technique, a server can

detect if the page request is coming from a desktop device or a mobile device and serve the web page code accordingly. This is achieved via accessing a server request variable called "HTTP_USER_AGENT." Similarly, the authors have demonstrated another technique to access client resolution from the server-side.

There are certain advantages and disadvantages of responsive web design. Hiding non-essential spaces, broadcasting a single website to multiple devices, better user experience, etc., are some of the highlighted advantages. Harder implementation by designers, less support for some browsers/devices, more processing screen times, are some of the disadvantages mentioned [9]. The authors also acknowledge that although the techniques mentioned in this research work well, they are still in their early stages and that new ones will be produced as more internet devices and technologies emerge in the future.

2.1.5 IPA keyboards for iOS

Armstrong (2015) [10] reviews three iOS keyboards built specifically for IPA (International Phonetic Alphabet) symbols. The first app is called IPAChartApp keyboard, and it is free to download. This app presents a clickable window that reveals symbols in each section of the IPA chart. It is not designed as a traditional keyboard layout. Since it resembles the IPA chart, there is a large table for pulmonic consonants and vowel space trapezoid for vowel symbols. On an iPad, quite a bit of scrolling has to be done to see all the symbols. On an iPhone, there is constant swiping of the display in order to find relevant symbols. Since most phonetic words alternate between consonants and vowels, the user must constantly flip through the layouts. The second app is called Black Keys, and it is not free to download (priced $1.19). The keyboard design in this

app is a QWERTY layout with a serif font. Each English alphabet in the keyboard, when held down, displays associated symbols with that alphabet.

Diacritics are easy to access. Subscripts, superscripts, and suprasegmentals all have dedicated keys on the keyboard. Since the keyboard is based on English alphabets, it is more suitable for English phonemic transcriptions. The author states that Black Keys' most significant disadvantage is that fingers tend to cover symbols when a key is held down, making it difficult to see the character. The third keyboard reviewed in this article is called IPA Keyboard. It is not free to use as well (priced $2.49). The design of this keyboard is similar to Black Keys keyboard, but it only features IPA. There is no access to non-standard language keyboards. However, unlike Black Keys, symbol options are displayed above a user's finger when a key is held down, making it easier to see the characters. There is also a row for frequently used suprasegmentals. This keyboard is more inclusive than Black Keys and contains all the IPA symbols possible in iOS. The author concludes the article with his recommendations on the three keyboards and defining situations where each might be useful.

2.1.6 Complete IPA keyboard for iOS devices

Dobrowolski (2015) [11] introduces his personally designed keyboard application for the iOS operating system in this article. The primary purpose was to show how a regular keyboard was merged with the complete IPA symbol set. A list of functionalities is provided for the newly developed keyboard, with future possibilities for the project. IPA is probably the most versatile style of speech delineation. It is mainly used in speech sciences and language learning. Being in Unicode, it is compatible with almost all operating systems. However, the issue is with its

complicated nature of digital typing. IPA does not have a dedicated keyboard. The two most popular techniques for using IPA are “copy and paste” and utilizing regular keyboard buttons with modifier keys. There are also virtual keyboards that allow for point-and-click writing. However, the main drawback is that using a mouse for point-and-click typing becomes tedious and time-consuming after a while. It is even more complicated to use smaller devices like smartphones for typing due to ergonomic issues. To alleviate this situation, Dobrowolski (2015) launched his keyboard to the iOS app store, called the Complete IPA Keyboard, which he abbreviates as CIPAK. Instead of being arranged in a meaningless fashion, sounds, diacritics, and suprasegmentals were located based on their most matching standard Latin keyboard counterpart. This was done to make the keyboard more intuitive. Other features of the keyboard include key expansion (to include more symbols), segmentation (to offer a more comprehensive layout), and interchangeability (to retain the standard aspects of an iOS keyboard). Keyboard’s color scheme was made to resemble IPA’s signature color set to promote familiarity and to retain all the usability features. A few drawbacks of CIPAK are - no support for dictation, no ‘split’ and ‘unlock’ typing methods, compatibility limited to only the iOS operating system.

The author’s stated motivation behind CIPAK was to make writing in IPA easier and allow it to be more frequently used. CIPAK enables iOS users to access the complete set of IPA letters, diacritics, and suprasegmentals for everyday writing and achieves that without much modification to the standard keyboard layout.

2.2 Algorithms

2.2.1 Edit Distance Calculation by Phonetic Rules and Word-length Normalization

Edit distance calculation is a popular technique for string comparisons and is quite appropriate for alphabetic languages like English. But for agglutinative languages like Korean, where two or more letter units make a Korean syllable, edit distance calculation can be inefficient. Bae et al. (2012) [12] explore a new edit distance method by consonant normalization with normalization factor in this research. Edit distance is defined as the total cost of unit operations (insertion, deletion, replacement, transposition) required to transform one string into the other completely. Other distance measures include Hamming distance, Jaro-Winkler distance, etc.

A Korean word is a sequence of syllables. A Korean syllable is a combination of consonants and vowels, and the order of combination is strictly consonant-vowel-consonant, the last consonant being optional. To apply one-dimensional edit distance metric to Korean two-dimensional word system, the following three ways are discussed -

- Letter-based edit distance
- Syllable-based edit distance
- Hybrid edit distance

Letter-based edit distance is simply applying the conventional Levenshtein edit distance to a letter string. It does not consider the syllable characteristics of the Korean language. Syllable-based edit distance counts the number of different syllables in a string. However, this metric has a critical problem if a typo error occurs on a syllable boundary. Hybrid edit distance combines the above two metrics in one. Despite this, the hybrid approach could not wholly solve the syllable-boundary changes. Bae et al. (2012) introduced a new approach for Korean edit distance calculation, which

is a phoneme-based edit distance. It takes into account the phonetic change rules of the Korean language. This new method consists of two steps - phoneme normalization and edit distance normalization. In phoneme normalization, nine phonetic transformation rules are sequentially applied, followed by applying letter-based distance metric to modified input strings. In edit distance normalization, word-length normalization is introduced to modify word lengths appropriately for distance calculations. The proposed metric by the authors of this paper was formulated to improve the performance of syllable-based and letter-based metrics for word similarities. The performance of edit distance calculation was refined by the application of phonetic pronunciation rules and word-length normalization. They conclude the study with experimental results, showing that phoneme-based metric obtained better results when compared to letter-based, syllable-based, and hybrid-based distances.

2.2.2 Genetic Algorithms And The Multiple Sequence Alignment Problem In Biology

Karadimitriou et al. (2000) [13] talk about the problem of Multiple Sequence Alignment in molecular biology. Molecular Sequence Alignment is used for constructing evolutionary trees from DNA sequences, and this helps in analyzing the protein structures for new designs. This research mentions that, to date, most multiple alignment methods are based on a dynamic programming-based approach. But because of this, the time complexity is exponential. Authors also report the usage of tree-based algorithms to solve this problem, but it too suffers from certain drawbacks. They specify that multiple sequence alignment falls into a category of combinatorial problems.

To solve the problem of multiple sequence alignment, the authors demonstrate the technique of genetic algorithms. They create a population of random solutions and utilize the concepts of natural selection, crossover and mutation to build upon previous inadequacies and improve these solutions. Authors also showcase the potential of these algorithms to solve a variety of complex optimization problems by utilizing the advantages of operating on several solutions simultaneously.

From their results, the authors suggest that optimal or near-optimal solutions to the problem of multiple sequence alignment can be obtained with genetic algorithms faster than with dynamic programming methods. They also highlight that genetic algorithms are inherently parallel in nature, and they, therefore, can be implemented very efficiently on a parallel computer to achieve the desired optimality.

2.2.3 String Matching Algorithms and their Applicability in various Applications

Singla et al. (2012) [14] describe the applicability of various string matching algorithms and the best use case category for each. String matching is used in various real-world applications like database queries, network systems, sequence alignments, etc. There are two main techniques for string matching, exact and approximate. Examples for exact string matching algorithms would be Needleman-Wunsch, Smith-Waterman, KMP, etc. Examples for approximate string matching algorithms would be fuzzy string search, Rabin-Karp, brute force, etc. To achieve performance improvement in string matching, the goal is to derive non-trivial combinatorial properties from more complex patterns like trees, regular expressions, graphs, and point sets. The authors discuss the following algorithms briefly in the paper –

- Fuzzy string search – Finds approximate substring and dictionary matches.
- Rabin-Karp – Uses hashing to match a string against a set of patterns.
- Needleman-Wunsch – Performs a global alignment on two sequences.
- KMP – Searches for occurrences of a string inside a larger string.
- LCS – Finds the longest subsequence common to all sequences.
- Levenshtein Distance – Measures the edit distance between two strings.
- Smith-Waterman – Performs a local alignment on two sequences.
- Brute-force – Mathematically proves matching conditions between two strings.
- Boyer-Moore – Performs preprocessing on the target string for efficiency.
- Jaccard similarity – Measures similarity between two strings.

They also provide a comprehensive analysis table to examine the performance of algorithms mentioned above and describe their central characteristic. From the paper, a few string matching applications along with their optimal algorithms are stated briefly as follows –

- Text editor, digital library, search engine – Boyer-Moore
- Multimedia and computational biology – Needleman-Wunsch, Smith-Waterman
- Medical tests - Boyer–Moore–Horspool
- String prefix matching – KMP
- Polymorphic string matching – Boyer-Moore with KMP
- Network intrusion detection system – Boyer-Moore
- Retrieving music patterns from a music DB – Levenshtein distance with Jaccard similarity

Most applications use Boyer-Moore, Boyer-Moore-Horspool, or KMP for their efficient functionality. Rather than using a generalized approach, it is much more effective to apply

appropriate algorithms best suited for each use case. The authors conclude the paper by expressing the importance of string algorithms and the work being done on software and hardware levels to make pattern searching faster.

2.2.4 Phonetic Alignment and Similarity

In several applications of computational phonology, including dialectometry, the computation of the optimal phonetic alignment and the phonetic similarity between two words is an important step. Kondrak (2003) [4] presents an original approach to the problem, which utilizes a scoring scheme for computing phonetic similarity between phonetic segments in accordance with multivalued articulatory phonetic features. This new algorithm developed by the author makes use of several techniques developed for sequence comparison, like an extended set of edit operations, local and semi-global modes of alignment, retrieval of near-optimal alignment sets, etc.

There are two main components for both word similarity and word alignment algorithms: a metric for measuring phonetic segments and a process for finding the optimal alignment. The author reviews several algorithms for calculating the phonetic alignment and similarity. Some of them are –

- Covington (1996) developed an algorithm which, on the basis of phonetic similarity, aligns cognates.
- Somers (1998) introduced an algorithm for aligning children's articulation data with the adult model.
- Kessler (1995) ran tests on various methods for calculating distances between Irish dialects.

- Oakes' (2000) program JAKARTA implements a phonetically-based alignment algorithm with the purpose of discovering regular sound changes.

To compute optimal phonetic alignment, dynamic programming algorithms are fast and efficient choices. Kondrak supports this statement and disputes other search strategies by providing arguments stating that a greedy search is not enough due to its erroneous recursive nature and that an exhaustive search strategy can be excessively time-consuming due to various combinatorial possibilities. The author also describes several extensions to the basic dynamic algorithm, which had been proposed to address issues in DNA alignment primarily. These extensions apply to phonetic alignment as well. They are as follows: retrieving a set of best alignments, string similarity, local and semi-global alignment, affine gap functions, and additional edit operations.

In phonetic alignment, the distance and similarity functions are essential. In the paper, designing an improved function is described, something that encodes the knowledge about universal characteristics of sounds. Discussions include feature-based metrics, similarity/distance, and multivalued features. To display the phonetic alignment approach recommended in this paper, the author introduces an implementation called ALINE. It is written in C++ and runs on Unix systems. The program is controlled via command-line params. It takes a list of word pairs as the input and produces a list of alignments and their similarity scores as the output.

To evaluate ALINE, qualitative and quantitative tests were performed, which used a set of 82 cognate pairs compiled by Covington (1996), containing words mainly from English, German, French, Spanish, and Latin. The evaluation involved alignment algorithms of Covington (1996),

Somers (1999), and Oakes (2000), along with ALINE and a binary feature-based algorithm. ALINE performed well in the qualitative evaluation and was the clear winner in quantitative evaluation, with over 95% accuracy. ALINE also produces an overall similarity score besides finding the optimal alignment. Altogether, ALINE produces better alignments than other algorithms based on the results of the evaluation. Even though ALINE was explicitly developed for cognate identification, its foundation lies in the general principles of articulatory phonetics. It has since proved its usefulness in a diverse range of applications.

2.3 Computational Tools

2.3.1 A Tool for Automatic Scoring of Spelling Performance

Themistocleous et al. (2020) [15] propose a novel method to automate spelling scoring based on distance metrics. Manual spelling scoring performance is an arduous and error-prone process. In this research, six automatic distance metrics were compared to identify which best corresponds to manual scoring.

Data used in this study is obtained manually from the spelling scores of individuals with primary progressive aphasia, a neurological syndrome in which language capabilities become slowly and progressively impaired. To gather this data, 3540 word and non-word spellings from 42 individuals (with primary progressive aphasia) were scored by hand. These manual scores were compared against the following six automated distance metrics –

- Sequence matcher ratio
- Damerau-Levenshtein distance
- Normalized Damerau-Levenshtein distance

- Jaccard distance
- Masi distance
- Jaro-Winkler similarity distance

Each distance metric was evaluated with the manual spelling score by the authors. The output of the comparison was a Spearman's rank correlation coefficient, which denotes the extent of correlation between word accuracy scores. Python3 was used to calculate the distance metrics. The results obtained from the evaluation showed that all the automatic distance scores had a high correlation with the manual method of scoring. Normalized Damerau-Levenshtein distance metric provided the highest correlation with the manual scoring (rs = 0.99 for words and rs = 0.95 for non-words).

The authors conclude the paper by stating that the high correlation between the automated and manual methods suggests that automatic spelling scoring forms a quick and objective approach and can be trusted to replace the existing, time-consuming manual spelling scoring process. This potentially makes it an essential asset for researchers and clinicians.

2.3.2 STAT: Speech Transcription Analysis Tool

The study of human accented speech and its theoretical and practical aspects has been of interest to various groups of people like linguists, language teachers, etc. In this research paper, Kunath et al. (2009) [16] introduce the Speech Transcription Analysis Tool (STAT), which is an open-source program for aligning and comparing two phonetically transcribed texts of human speech. Initially, STAT was developed to provide sets of phonological speech patterns (PSPs) in

the comparisons of different English accents. But its scope and utility can be extended to areas like language assessment, phonetic training, forensic linguistics, and speech recognition.

The process of manually comparing two phonetically transcribed speech samples requires extensive training and is time-consuming, to say the least. Specific issues are associated with this manual process, cost and time to train linguists to perform uniform PSP analyses being the major ones. There is a lack of a quick comparison framework. To mitigate this problem, the authors developed STAT. It allows fast and pointed comparisons of any two phonetically transcribed speech samples. It also serves as a quick and systematic way of checking human transcription accuracy, in addition to providing a human diagnostic factor in spectrographic speech analysis.

STAT is a combination of various components. Interaction with the tool takes place mainly via the web. The front-end is designed with Ruby on Rails, and the back-end is programmed in Java. Linguists can play an audio file and export it using STAT. It also provides a mechanism to manage phonetic transcriptions. The language management component of STAT allows a user to define a new speaker source language. Linguists can also search transcriptions by speaker demographics, phonetic inventories, PSPs, and speech quality assessments.

For the alignment of transcriptions, a special implementation of Kondrak's phonetic alignment algorithm (Kondrak, 2000) is used. This outputs a complete phone-to-phone alignment of transcriptions. Linguists can also control various parameters through the tool, like phoneme distance measures, constraint sets for PSPs, etc. After transcription alignment, phonological speech pattern analysis is performed by the algorithm to determine unique patterns.

Kunath et al. (2009) advocate the need to incorporate automated methods for weight setting suggestions based on language selection. There are also plans for the integration of a spectrographic analysis mechanism and speaker accent identification algorithms. Additionally, they will be investigating possible applications of the tool to help speech pathologists assess disordered speech patterns.

CHAPTER 3: METHODOLOGY

3.1 Research Problem

The research problem was to develop a more efficient and accurate phonetic transcription comparison analysis method by creating a unique tabular priority control system. In consultation with Dr. Bailey and Dr. Speights Atkins, it was found that the previous work did not accurately produce precise calculations. This was due to the lack of phonetic principles implementation, like nucleus alignment, for instance, and missing parameters in initial computations. The aim is to provide an innovative solution for phonetics comparisons with a higher degree of accuracy than prior work.

The core functionality of APTct involves the ability to calculate the distance between two phonetic strings and provide us a measure of the phonetic transcription comparison. This distance metric is the edit distance between the phonetic transcription strings being compared. Edit distance is a method of gauging how similar or dissimilar two strings are based on the minimum number of operations needed to transform one string into the other [17]. Different variations of edit distance computation use different operations. The following are some examples –

1. Levenshtein Edit Distance – Allows insertion, deletion and substitution of a character.
2. Longest Common Subsequence – Allows only insertion and deletion.
3. Hamming Distance – Allows only substitution.

There are several others, but for APTct, Levenshtein Edit Distance was found to be the most useful metric by Dr. Bailey and Dr. Speights Atkins. Each operation (insertion, substitution,

or deletion) carries a penalty of 1.0. Let us see a basic example for it. Suppose we have two strings, "cat" and "bark." Operations required to transform "cat" into "bark are as follows –

- "c" – "b" | substitution | 1.0
- "a" – "a" | alignment | 0.0
- "t" – "r" | substitution | 1.0
- "k" | insertion | 1.0

The total cost of all these operations is 3.0. Therefore, the edit distance between "cat" and "bark" is 3.0.

To automate phonetic transcription comparison, Levenshtein Edit Distance is, therefore, the preferred option since it can calculate the similarity/dissimilarity between two transcriptions, providing helpful information to the transcriber.

3.2 Original Solution for Automated Comparison

The original solution for automating comparison was implemented in Microsoft's VBScript by Dr. Bailey's associate (see Figures 2 - 4).

```
'*************************************************************************************************************
'************** Levenshtein (weighted) ***********************************************************************
' Inputs: 2 Strings s1 and s2
' Output: 1 string concatenation of the final distance and the word alignment string
'Function Description: The function is called from the worksheet. It takes two strings. It compares the two strings,
'character by character, looking for the minimum distance between the two words. The distance is found by calculating
'the insertions, deletions, and substitutions to go from one word to the next. Only vowels can be substituted for vowels,
'and consonants for consonants. The "PriorityTable" tab has columns defining vowels and consonants. Precedence is given
'for vowel substitution rather than consonant. This is done by creating a weighted distance. The weight for a consonant
'is far greater than the weight of a vowel, giving precedence for vowel substitution.
'In addition to a distance, a word alignment is also created. This alignment lines up between the two input words to give
'a visual on the substitutions, insertions, and deletions used to go from the first word to the second. Each character of
'the word alignment is created using the following table:
'
'   no change, both characters are vowels      -> *
'   no change, both characters are consonants   -> |
'   consonant insertion                          -> %
'   vowel insertion                             -> i
'   deletion                                   -> d
'   substitution                                -> s
'   numeral deletion                             -> !
'
'The second input word may contain numerals. Each numeral is always paired with a character and adds 0.5 to the final actual
'distance. This function first parses through the second word to delete all of the numerals, while recording where they were,
'and adding 0.5 to the starting distance for each numeral found. Near the end of the function, a "!" is added to the word
'alignment to signify a numeral insertion.
'
'The actual distance is calculated by adding 1 for insertions, deletions, or substitutions and 0.5 for each numeral. The
'actual distances are kept in the 2D d array. The weighted distance (using vowel/consonant weights) is kept in the 2D p array.
'Decisions to find the shortest distance are made using the weighted distance. However, at the end only the actual distance is
'reported. The shortest distance is the bottom right entry of the array at the end.
'
'At the end the actual (not weighted) distance and the word alignment are returned as one concantenated string. These are
'separated back out in the worksheet.
'
'The iReplace function takes the original first word and the word alignment. It adds "?" to the input word for each consonant
'or vowel insertion and "#" for each numeral insertion. It returns the final input word.
'
'The dReplace function takes the original second word and the word alignment. It adds "~" to the input word for each character
'deletion. It returns the final input word.
'*************************************************************************************************************
```

Figure 2

```
    'weights and characters for substitution, deletion, insertion
    wghts = 1
    wghtd = 1
    wghti = 1
    vai = "i"

    'if string 2 character is consonant, insertion character should be %
    ptwght = PriorityTable(Mid(s2U, j, 1), Mid(s2U, j, 1))
    If ptwght = consonantWeight Then
       vai = "%"
    End If

    vad = "d"
    vas = "s"

    'evaluate deletion and insertion actual distance, weighted distance, and alignment character
    min1 = d(i - 1, j) + wghtd
    min2 = d(i, j - 1) + wghti
    p1 = p(i - 1, j) + wghtd
    p2 = p(i, j - 1) + wghti
    tS1 = tS(i - 1, j) + vad
    tS2 = tS(i, j - 1) + vai

    'compare deletion and insertion, keep smaller
    If p2 < p1 Then
       min1 = min2
       p1 = p2
       tS1 = tS2
    End If

    'evaluate substitution actual distance, weighted distance, and alignment character
    min2 = d(i - 1, j - 1) + wghts
    p2 = p(i - 1, j - 1) + wghts
    tS2 = tS(i - 1, j - 1) + vas
```

Figure 3

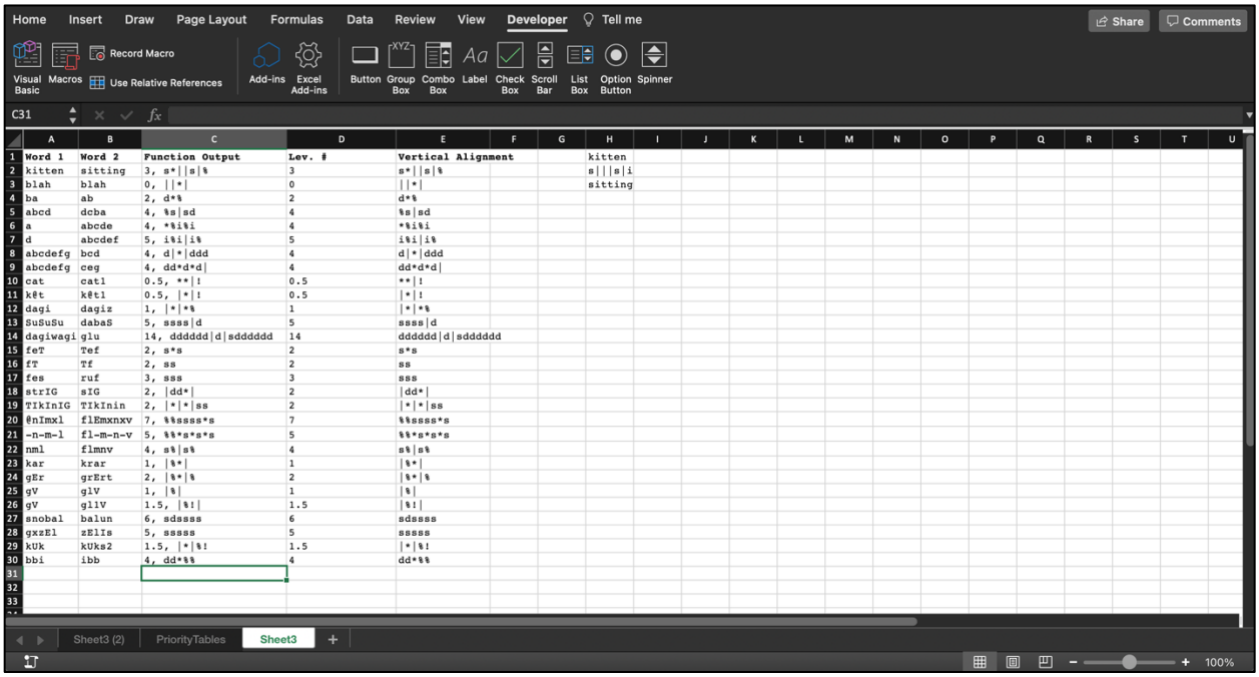

Figure 4

To use this, a person had to input the transcriptions into two columns in a Microsoft Excel workbook (see Figure 4). The third column was attached to a macro that ran the Levenshtein algorithm and outputted the result. The fourth column showed the vertical alignment of the two transcriptions.

3.3 Problems and Limitations of the Original Solution

While this setup produces correct and accurate comparison results, there are a few problems and limitations that this tool runs into.

- IPA symbols cannot be used because the tool is implemented inside Excel and Excel only supports English alphabets. As a result, the knowledge of the Klattese writing system [18] is a requirement for the user, which is a mapping from IPA symbols to English alphabets.

Diacritics are represented as numbers. Many clinicians might not be familiar with this writing convention, which elongates the learning curve to use the tool.

- The nucleus alignment principle of phonetic transcription comparison breaks at 20 characters and an incorrect distance is produced as a result.
- Vertical alignment of the comparison, as shown in the fourth column of the tool, is not very intuitive and is difficult to comprehend at a glance.
- Since the tool is implemented inside Excel, it is not widely available to the public. If someone wishes to use it, they need to have the Excel file on their computer. This further makes the tool harder to use.

These problems led to the development of a web-based phonetic transcription comparison tool, which was the first version of APTct.

3.4 First Version of APTct Calculator

The first version of APTct (see Figures 5 -7) was developed (by CS Ph.D. students) as a web-based tool, using HTML5, CSS3, and JS for the front end, Java Servlets, and JSP for the back end. Hosted on Auburn University's server, it supported the IPA character set for transcription, significantly improving the Excel workbook solution. There was also an on-screen keyboard to allow users to input IPA characters, using which they could transcribe speech and compare those transcriptions. This version of APTct also had a summary section, which displayed useful aggregate information about the comparisons. Being a web application, it was readily available to the general public.

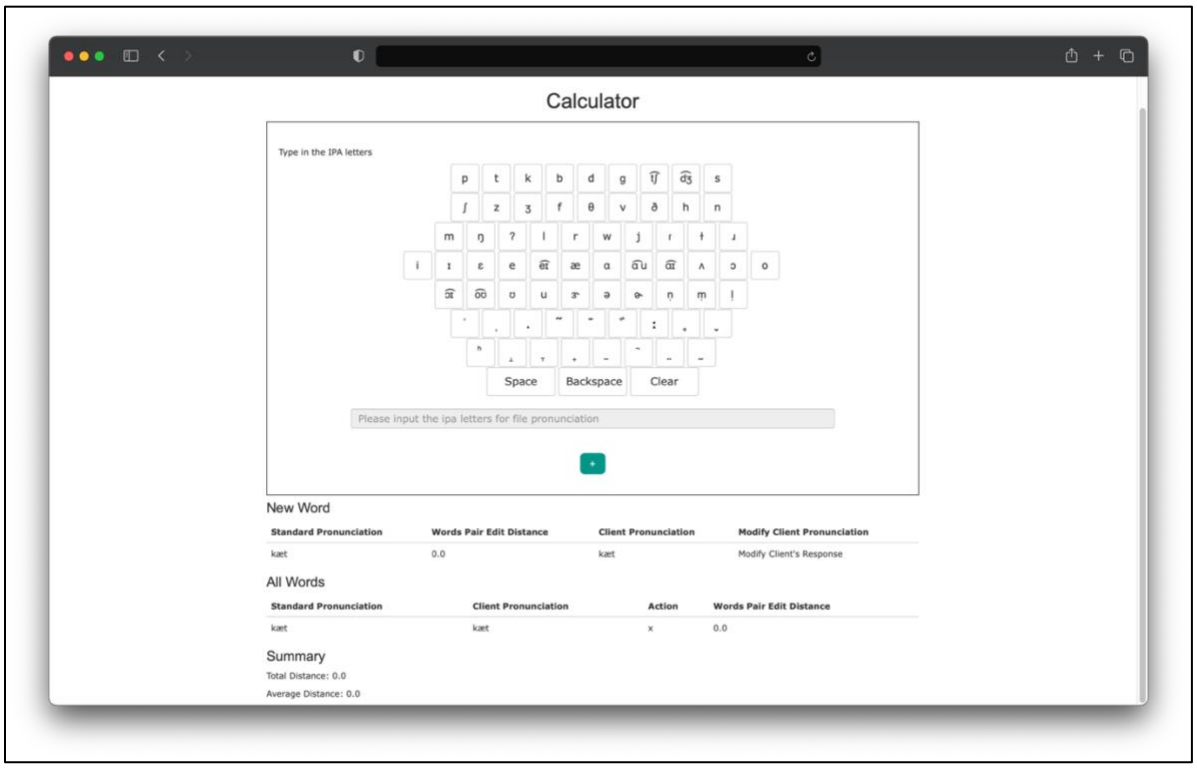

Figure 5

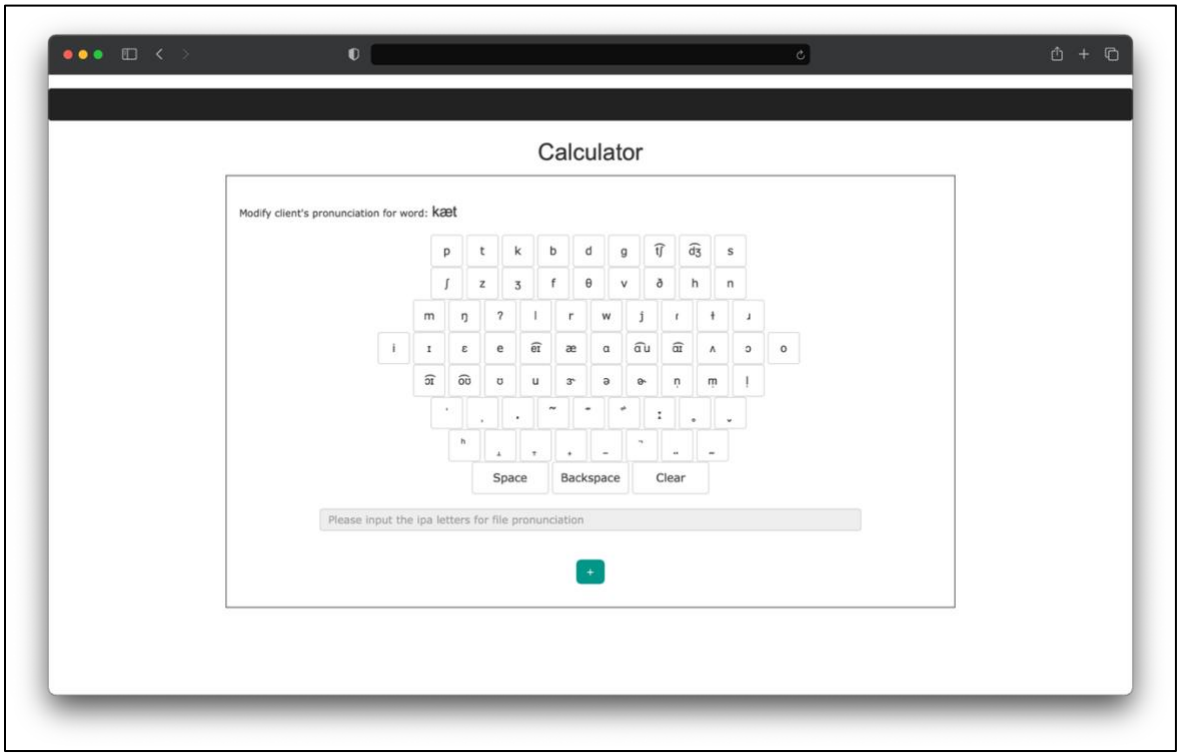

Figure 6

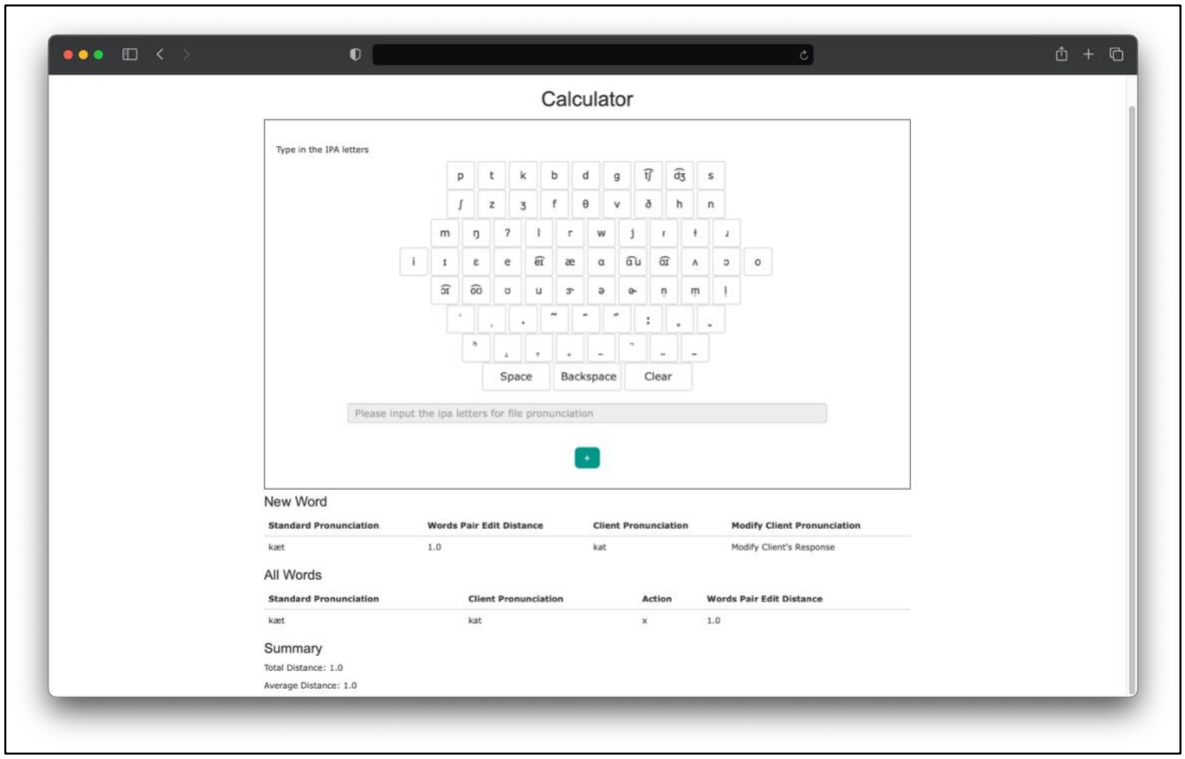

Figure 7

3.5 Problems and Limitations of the First Version

Although this version was a significant enhancement from the previous Excel tool, it involved a few drawbacks.

- This calculator does not support the nucleus alignment principle of phonetic transcription comparison, which is an essential postulate in emulating the hand transcription process of phoneticians.
- The algorithm implemented underneath the tool was producing a few incorrect results for some comparisons.
- Complete IPA character set was not supported, and the calculator itself was not very intuitive to use.

- There was no arrangement for users to use English alphabets as physical keyboard entry was disabled.
- The calculator sent data over the HTTP protocol (unsecure) instead of HTTPS protocol (secure).

Given these issues, there was a pressing need to develop a wholesome, universal phonetic comparison tool for phoneticians, linguists, teachers, and other personnel to utilize. Sections below describe several specifications and requirements needed to achieve the same.

3.6 Functional Requirements

Dr. Bailey and Dr. Speights Atkins found the following functional requirements to be essential in the development of the new version of the calculator.

- The algorithm which powers the computation should have a high degree of accuracy and should hold up to several edge cases of transcription comparisons.
- It must accomplish the nucleus alignment principle of phonetic transcription comparison.
- There has to be full support for all the characters in the IPA set.
- Users should also have the option to use English alphabets typed via physical keyboards on their machines.
- The tool should consist of an on-screen keyboard for users to employ IPA symbols in their respective transcriptions.
- Users should be able to input two transcriptions at a time for comparison.
- There should be a section that displays all the transcription comparisons computed in the current session for the user's convenience.

- None of the transcription data should be saved permanently on the server. It is okay for data to exist temporarily on the browser so that the user can see their summarized transcriptions, but it should be removed entirely once the transcription comparison session is terminated.

3.7 Non-functional Requirements

The following non-functional requirements have been considered by Dr. Bailey and Dr. Speights Atkins to further improve user experience. These specifications also make the tool more professional in terms of public exposure.

- There should be an option to export all the transcription comparisons (performed by a user) in a distributable format so that they can save and document their tool usage for study and research purposes.
- Alignment sequences for comparisons between transcriptions must be made available so that users can better visualize and understand the working behind the algorithm.
- The web application should be responsive on all devices, including mobile phones and tablets, to enable universal access to the tool.
- A splash page containing citation and tool usage information has to be set up.
- IPA guides must be present to provide users with a quick, official reference to IPA charts.
- An on-screen keyboard must be implemented to closely resemble IPA charts for a better typing experience.
- There should be a contact form so that users can reach out to Dr. Bailey and Dr. Speights Atkins if they have any questions.

- All the data transfer to and from the server should be facilitated over HTTPS protocol, enhancing application security. To achieve this, an SSL certificate has to be set up on the server.

3.8 System Requirements

Since the tool will be made available publicly as an online web application, the requirements are relatively straightforward.

- Software Requirements
    - One of the latest available web browsers (Chrome or Safari preferred). JavaScript should be enabled in the browser for APTct to function properly.

- Hardware Requirements
    - Desktop computer, laptop, tablet or a mobile device, equipped with a web browser to access the website.
    - An internet connection.

3.9 Wireframe and Preliminary Design

With all the initial requirements firmly set, an application wireframe and a preliminary design was developed for the main page of APTct.

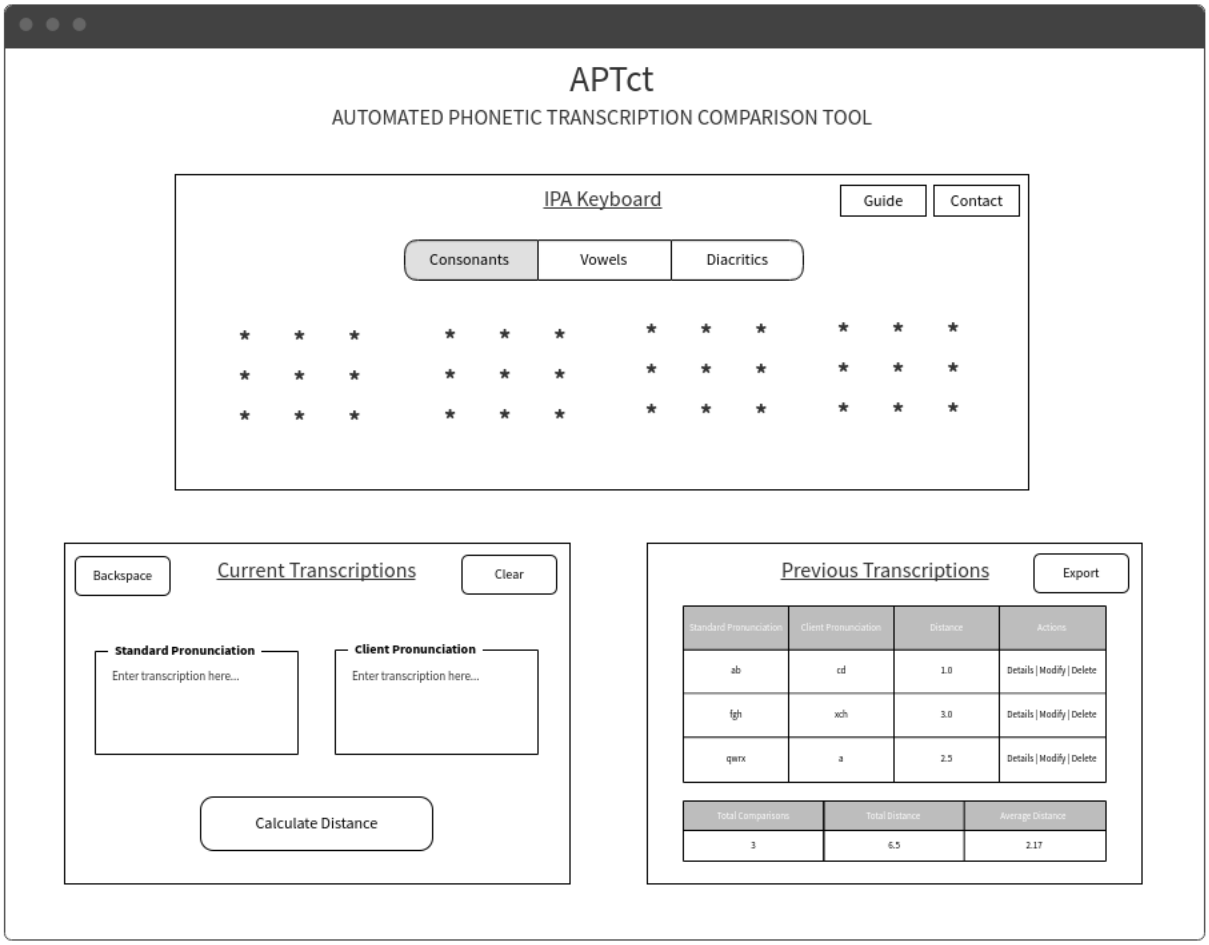

Figure 8

Figure 8 illustrates the three main design components for APTct, as described below –

- Keyboard - This section contains the main IPA keyboard, with three distinct tabs for switching between three parts of the keyboard, which are “Consonants”, “Vowels” and “Diacritics”. Each part contains relevant IPA symbols which fit into that category. There are also two buttons, named “Guide” and “Contact”. The “Guide” button can be used to open the official IPA chart, serving as a guide/reference. The “Contact” button can be used to reach Dr. Bailey and Dr. Speights Atkins if the user has any questions.

- Current Transcriptions - This section displays the ongoing transcription process which the user is performing. It has two parts, “Standard Pronunciation” and “Client Pronunciation.” They are essentially the two transcriptions that the user wants to compare. It also includes “Backspace” and “Clear” buttons for deleting one character and for clearing the entire transcription, respectively. Lastly, the “Calculate Distance” button can be pressed for initiating the phonetic transcription comparison process between the two transcriptions entered by the user.

- Previous Transcriptions - This section displays all the transcription comparisons performed by a user in their ongoing session. It also includes a summary of all the calculations and is updated continuously as the user’s transcription comparison process continues. There is an “Export” button in this section, which can export all the transcription comparisons performed by the user, in a compact PDF file. For each comparison, the user will have the following actions available to them –

    - Details – This contains alignment information and other important elements of the comparison process for the two transcriptions.
    - Modify – This removes the comparison from the “Previous Transcriptions” section and copies the two transcriptions to their respective fields in the “Current Transcriptions” section, allowing modifications.
    - Delete – This deletes the comparison entry from the “Previous Transcriptions” section.

3.10 Interaction Scenarios

In order to understand the functioning of APTct, three hypothetical scenarios were developed in relation to the preliminary design of the application.

3.10.1 Scenario 1

Dr. John Lucas is a speech-language pathologist who regularly works with patients having speech and language disorders. To store his patients' disordered speech samples, he transcribes their utterances and saves them as text. He then produces and transcribes the correct version of the same speech for reference. Dr. Lucas can see how far (in terms of distance) his patient's speech samples are from the authentic version by inputting the standard transcription as transcription one and the patient's speech transcription as transcription 2 in APTct. APTct will then calculate the distance between the transcriptions. It will also provide Dr. Lucas with comparison alignment details, in which he can identify key data points to assist him in the correction of his patient's speech. Since APTct also has an export option, he can export these comparison results to his computer and track the improvement in his patients' articulation over a period of time.

3.10.2 Scenario 2

Dr. Stacey Brown is a distinguished linguist who specializes in studying the language of people from lesser-known origins. A person is brought into her lab whose dialect cannot be determined. Dr. Stacey asks him a few questions, to which he replies in his dialect. She records her perception of the person's speech using APTct and exports the results to her computer. She then shares the interaction transcription with her co-researchers for analysis of the person's language. After examining various features in the person's speech, Dr. Stacey assembles a list of

a few dialects closest to the person's dialect and prepares a standard transcription for each. Using APTct, she compares each of these transcriptions to the person's speech transcription and calculates the distance. The distance and alignment details of the comparisons provided by APTct, help Dr. Stacey approximate the person's dialect.

3.10.3 Scenario 3

Dr. James Morris is a professor at Auburn University and teaches a phonetics course to undergraduate students. Throughout the course, he decides to use APTct for teaching students how the phonetic transcription comparison process works, which is more effective than manually drawing the IPA symbols and comparing and calculating the distance between transcriptions since the automated computation of the tool saves valuable time. This is especially applicable for longer and more complex transcriptions, usually prone to errors and inconsistencies. Also, APTct provides a descriptive and visual representation of the alignment, which can be pretty helpful for students as it will aid them in understanding the transcription comparison more thoroughly. Additionally, Dr. James can give out assignments to his students which they can complete on APTct. Using the export feature, students can submit their answers, which can be graded by Dr. James accurately.

CHAPTER 4: IMPLEMENTATION

All the development for APTct was performed in software sprints because the software development process chosen was Scrum, and requirements for this project were dynamic. Scrum is an agile framework used in software development to produce sustainable and complex products [19]. It is an iterative and incremental process used for managing software builds efficiently. For this reason, Scrum is well suited for environments where requirements can change from time to time and flexibility is desired. APTct fits into this category. This section will describe in detail the creation of APTct, including all the configurations implemented on the server to realize the tool.

4.1 Languages and Tools

Specific categories and technologies used for development are described below.

- Front-end: For front-end development, HTML5, CSS3 (with Bootstrap 4.5) and JavaScript ES6 were used.
- Back-end: For back-end development, Java Server Pages (JSP) and Servlets were used. These are implemented using Java, and Apache Tomcat is the server hosting the website. For APTct, Apache Tomcat 9 was used.
- Other: APTct's webpage was designed on Brackets editor, which has several useful features to facilitate faster execution of code and ideas. Eclipse was the choice of IDE for writing and developing all the server-side code. GitHub was used for source control and versioning.

4.2 Server Setup and Hosting

This section describes the configurations done to make APTct web application available at the following URL: aptct.auburn.edu, with an SSL certificate (provided by Auburn University).

4.2.1 Problem

APTct was set up on one of Auburn University's several server computers. APTct's base machine is a computer running Windows Server 2012 R2 as the operating system. Auburn University provided us with three domain names pointing to this machine. They are –

1. auburnalt.cse.eng.auburn.edu
2. aptgtlms.auburn.edu
3. aptct.auburn.edu

The first URL, auburnalt.cse.eng.auburn.edu, is the base hostname attached to the machine. The second one, aptgtlms.auburn.edu, is the hostname for another project coordinated by Dr. Speights Atkins and Dr. Bailey. The third URL, aptct.auburn.edu, is the desired hostname to reach APTct's online application via the internet.

The same machine controls all the three URLs mentioned above. Because of this, the same content was being displayed on all three hostnames (see Figure 9), which was the homepage of IIS (standing for Internet Information Services), the default server running on Windows Server 2012 R2. This is problematic because all three hostnames and their content have particular application concerns. Apart from that, both aptgtlms.auburn.edu and aptct.auburn.edu applications have to be run on Apache Tomcat because they both are built using Java Servlets and render web pages using Java Server Page coding. The only way to reach the machine's underlying Tomcat instance,

without referring to IIS, was to use port 8080 (default server port for Apache Tomcat) in the URLs. If this method was undertaken, then APTct's URL would have become something like aptct.auburn.edu:8080. This is less than ideal and slightly unprofessional. So, the requests to the server from port 80 (HTTP) and port 443 (HTTPS) have to be forwarded from IIS to Apache Tomcat. Furthermore, since there is only one instance of Apache Tomcat running on the machine (as shown in Figure 10), it has to be configured to serve different content on different context paths, i.e., aptgtlms.auburn.edu and aptct.auburn.edu.

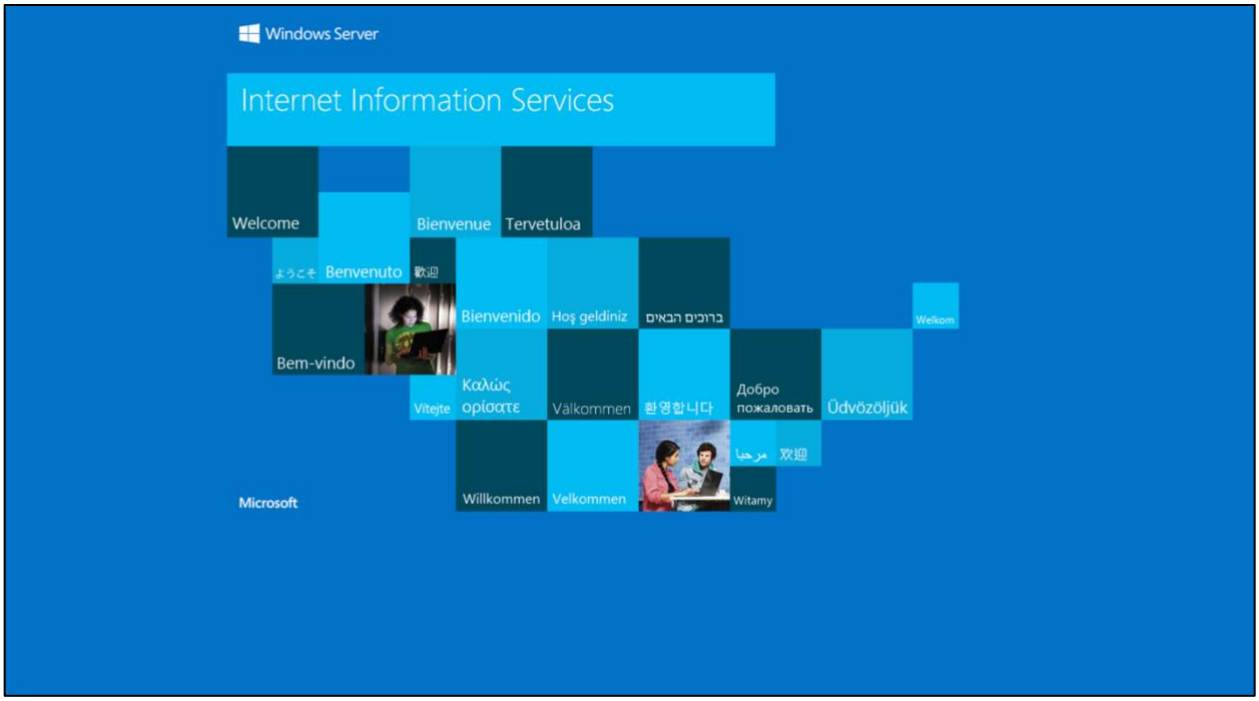

Figure 9

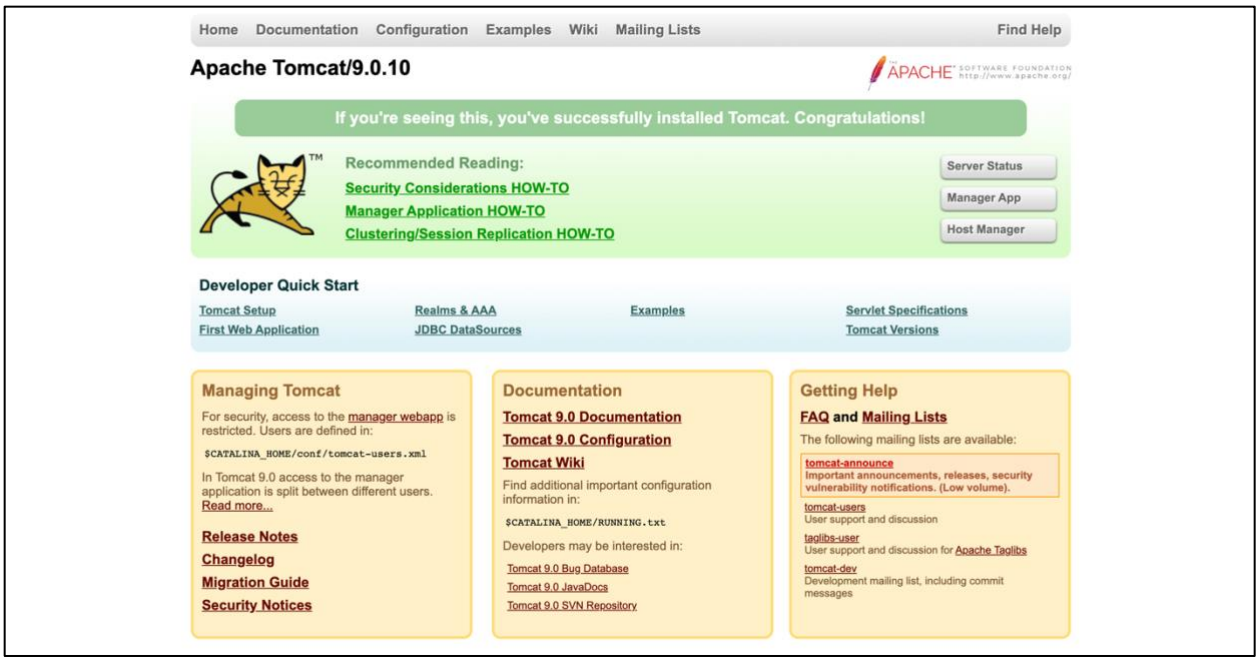

Figure 10

4.2.2 Solution

A connectional bridge needs to be established so that all requests to APTct are forwarded from IIS to Apache Tomcat. Apache Tomcat Connectors [20] were used to do this, which essentially are plugins to connect web servers with Tomcat and other backends. For IIS specifically, a plugin (extension) named ISAPI redirector is used. This plugin is compatible with IIS running on Windows Server 2012 or later and Tomcat 7 or later, both of which requirements are satisfied by APTct's server machine. Following the instructions given on this webpage - https://tomcat.apache.org/connectors-doc/webserver_howto/iis.html, ISAPI redirector was successfully configured with Tomcat. This enabled all the requests arriving at aptct.auburn.edu to be forwarded to Tomcat via IIS without using port 8080 at the end of the URL. Next, to ensure that different applications are being served at different context paths (namely aptgtlms.auburn.edu and aptct.auburn.edu) by Tomcat, virtual hosting [21] was configured on the server machine. This

allowed effective separation of application concerns. Finally, SSL certificates, provided by Auburn University, were configured on the server (see Figure 11) so that aptct.auburn.edu uses HTTPS by default, i.e., https://aptct.auburn.edu/.

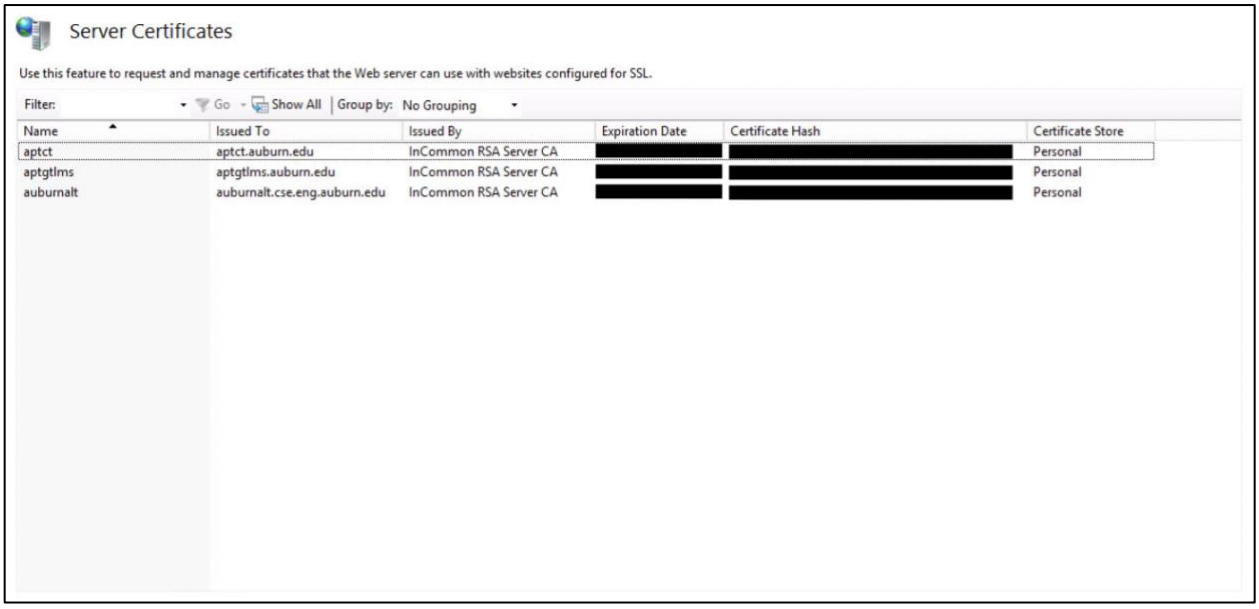

Figure 11

4.3 Application Architecture

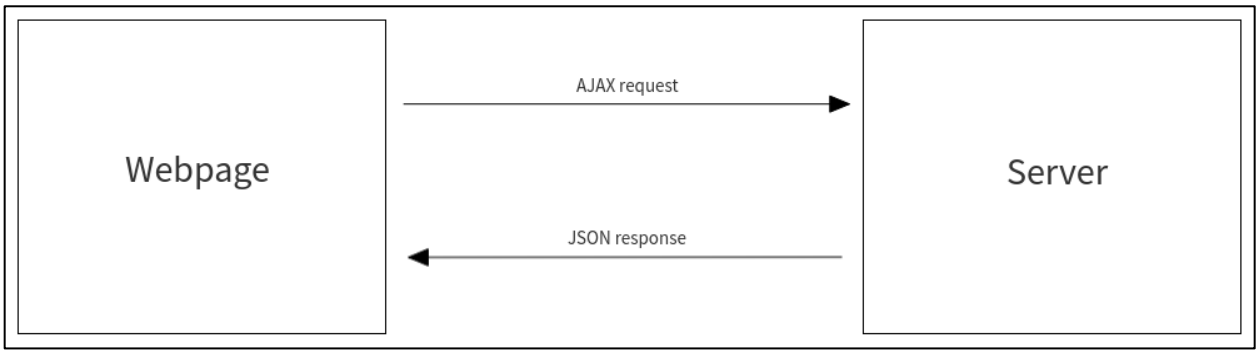

Figure 12

APTct's base architecture is relatively straightforward (see Figure 12). Two main modules are communicating with each other. The front-end, or the webpage, sends a POST request to the server with the user's inputted transcriptions as the data. The server performs all the required computations and calculations in the backend. It then returns those computed results (formatted as JSON) to the front-end. The front-end displays these results, including alignment details, to the user in an organized fashion. The results are also temporarily stored on the browser so that users can view their past transcription comparisons in an ongoing session. These results are cleared from memory as soon as the browser tab with APTct running on it is closed. Further details will be discussed in the development sprints.

4.4 Sprints

This section will describe all the five sprints undertaken to complete APTct and make it ready for clinicians, linguists, teachers, and others. Every sprint discussion covers the development performed for categories like front-end, back-end, etc. Additionally, the testing section of each sprint talks about the validation of that sprint conducted by Dr. Bailey and Dr. Speights Atkins.

4.4.1 Sprint 1

This sprint marks the beginning of the APTct application. It was started after Dr. Bailey and Dr. Speights Atkins concluded the first functional requirements for the tool, most of which aimed to improve the earlier version.

4.4.1.1 Front-end

The user interface for the new APTct was completely overhauled and modernized from the earlier version (see Figures 13 and 14). The new web page is divided into three main sections. The first one is for the point and click IPA keyboard. The second one is for user input. The third section is for results.

APTct's webpage interaction and interface are powered extensively by JavaScript and jQuery, a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and AJAX much simpler with an easy-to-use API that works across a multitude of browsers [22]. For responsive CSS and layout support, Bootstrap 4.5 was used. The Doulos Sil font [23] was embedded within the website to provide maximum compatibility for IPA symbols.

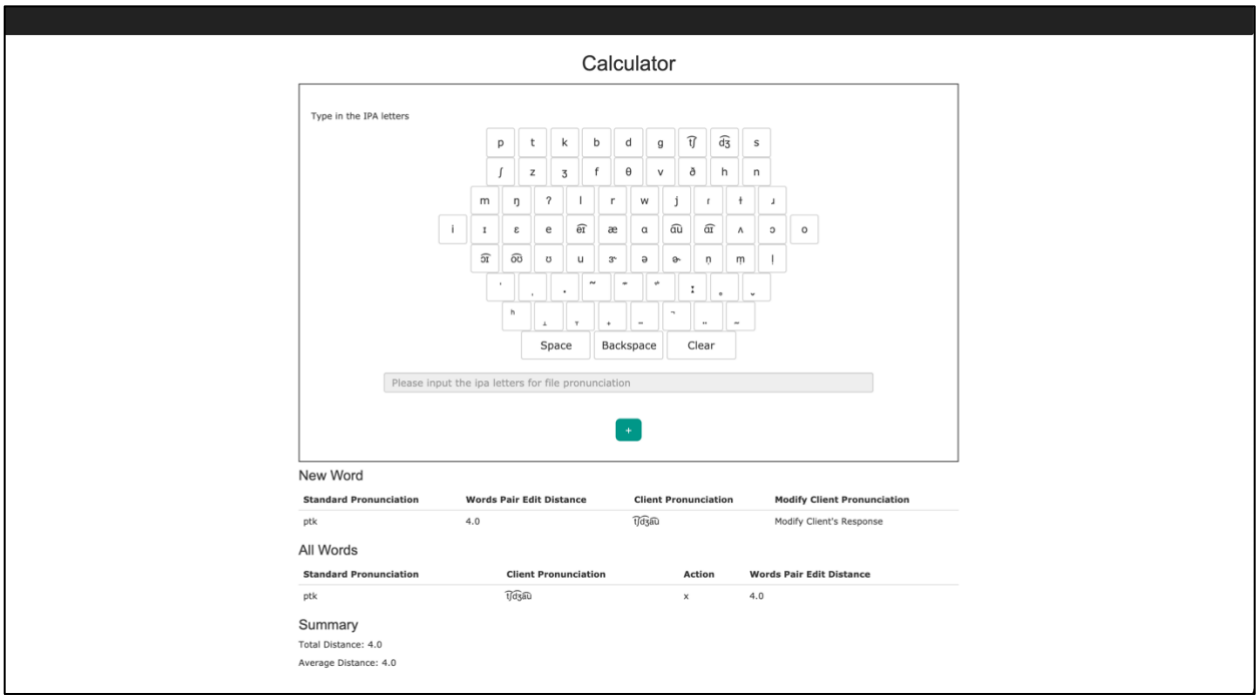

Figure 13

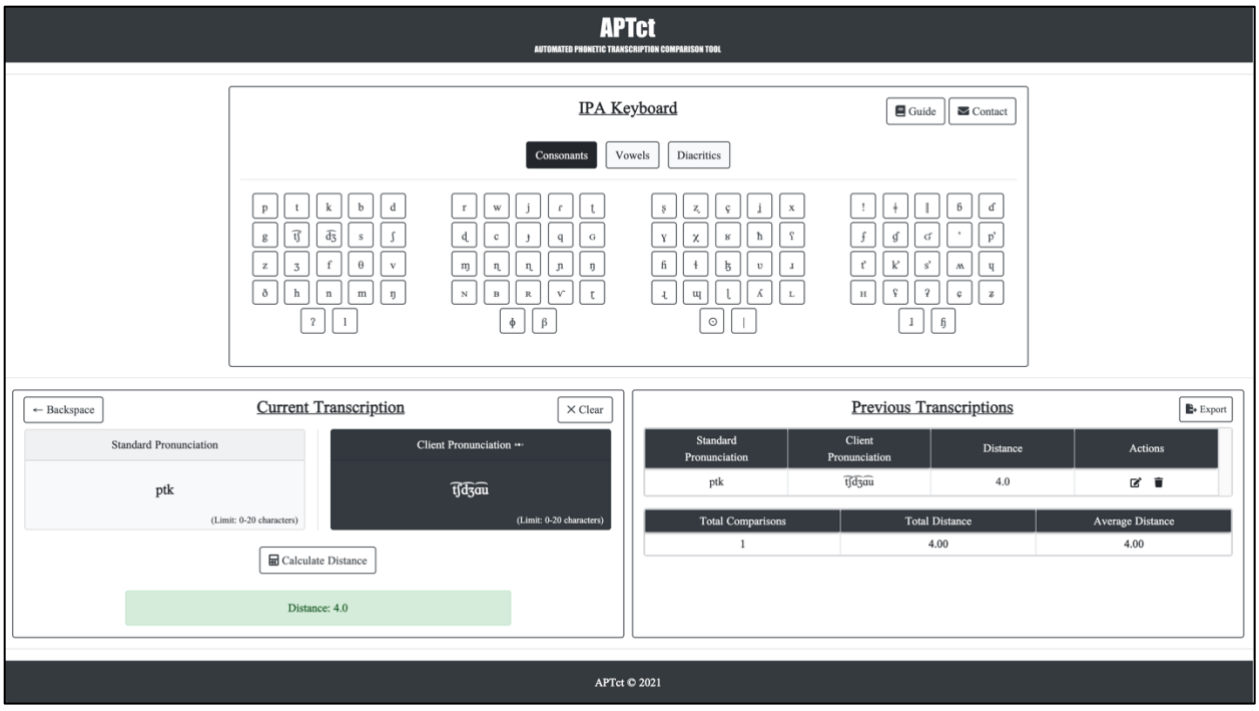

Figure 14

For the keyboard, instead of having a single segment with all the IPA symbols in it, the new UI has three separate tabs for each category of IPA symbols, which are “Consonants”, “Vowels”, and “Diacritics”. A user can switch between these tabs with just a click. This provides easy access to different symbols and removes the difficulty of finding the right character since these symbols are categorized distinctly. A guide button (which displays the official IPA chart) and a contact button (which allows users to contact Dr. Bailey and Dr. Speights Atkins) was also added above the keyboard.

For the user input section, the new interface provides two text fields for transcription one and two, allowing for simultaneous input of both transcriptions. This is far more efficient than the previous interface because earlier, a user would have to perform an unnecessary comparison before

they were able to change the second transcription. Standard text interaction buttons like “Backspace” and “Clear” are also provided.

For the results section, the new interface displays a compact table featuring all the previous comparisons and their details (transcriptions being compared and the distance). Like the old version, a summarized statistic of all the comparisons performed in the ongoing session is also provided. There is an added functionality to export the results in a PDF file for later access. This feature can be handy in cases where comparison results have to be distributed to different people or entities for research purposes, etc.

Whenever a user clicks on a symbol while transcribing, it is stored in a buffer array as a string. When the “Calculate Distance” button is pressed, both the buffer arrays (one for each transcription) containing IPA symbol strings are sent to the backend. This is different from the earlier version of APTct, which sent these transcriptions to the backend as whole strings instead of an array of strings. For example, let us say that the word “bottle” is entered as a transcription. The old version would have sent this to the backend as “bottle”, i.e., a single string. The new version sends this to the backend as [“b”, “o”, “t”, “t”, “l”, “e”]. The benefits of this new approach are discussed in further sections.

4.4.1.2 Back-end

There is a massive improvement in the response time of new APTct when compared with its earlier counterpart.

Previously, transcription entered by a user was sent to the server via a traditional POST request. The server then compared the transcription with the transcription itself and then returned the calculated distance (which was 0 because the transcription was compared with itself) to the front-end, which displayed the results after a page refresh (standard procedure). Then, to enter the second transcription for comparison with the first one, the user had to click on the "Modify Client's Response" button. This took the user to another page where they had to input the second transcription, and then from there, they returned to the first page, which then showed the requested comparison between the two transcriptions. The entire process was arduous and time-consuming.

In the new version of APTct, all the extra steps have been truncated. To compare two transcriptions, a user can enter both the transcriptions on the same page simultaneously without going back and forth between web pages. When the "Calculate Distance" button is pressed, an AJAX POST request is sent to the server instead of a traditional POST request. AJAX stands for **A**synchronous **J**avaScript **A**nd **X**ML and is used to send and receive data from the server asynchronously. This means that there is no page refresh at all, and the application works quite fast. Therefore, users see the result of comparison almost instantaneously as all the data exchange is performed in the background. There is no page refresh. On the server-side, a Java servlet receives both the transcriptions as input, runs the comparison algorithm on them, and then returns the result (formatted as JSON) to the front-end for display.

4.4.1.3 Algorithm

When we talk about phonetic transcription comparison, we intend to describe how near or far two transcriptions are from each other. Essentially, we are calculating the minimum number of

operations required to transform one word into the other. The distance resulting from this calculation is known as the edit distance between two strings [24].

Several algorithms can compute the edit distance between two strings. They differ in terms of the operations that are allowed on the strings being compared. For example, Levenshtein distance allows the insertion, deletion, and substitution of characters, LCS (longest common subsequence) distance allows only insertion and deletion, but not substitution. Other algorithms include Hamming distance, Jaro distance, etc. After researching, Dr. Bailey and Dr. Speights Atkins concluded that Levenshtein distance best suited the computational needs for the transcription comparison tool. Therefore, Levenshtein distance has been used as the algorithm of choice for APTct. Although the base algorithm is the same, there has been a minor scoring modification in Levenshtein distance suggested by Dr. Bailey and Dr. Speights Atkins to align the computation with phonetic transcription comparison better. Instead of giving the same penalty of 1 for all the characters and the operations (insertion, deletion, substitution), the following penalty rubric was decided upon –

- **Insertion & Deletion**
  - Consonant: 1.0
  - Vowel: 1.0
  - Diacritic: 0.5
  - Special: 1.0
- **Substitution**
  - Consonant → Consonant: 1.0
  - Vowel → Vowel: 1.0
  - Consonant → Vowel (and vice-versa): 2.0
  - Diacritic → Diacritic: 0.5
  - Diacritic → Consonant/Vowel (and vice-versa): 1.5
  - Special → Consonant/Vowel (and vice-versa): 1
  - Special → Diacritic (and vice-versa): 1.5
- **Alignment**
  - Consonant ↔ Consonant: 0.0

- Vowel ↔ Vowel: 0.0
- Diacritic ↔ Diacritic: 0.0
- Special ↔ Special: 0.0

In the earlier version of APTct, there was an additional pre-processing step before the Levenshtein implementation. That step was required because the front-end sent the transcriptions to the backend as strings, so certain characters which are naturally grouped in IPA had to be combined as one string because they form a single symbol in the IPA. This step is not required in the new version because those symbols are received from the front-end directly as a grouped string (instead of characters), eliminating the need to perform grouping again. In the backend, those symbol strings are matched against a store of symbol signatures from the IPA for proper identification of category (the symbol could be a consonant, a vowel, or a diacritic).

The scoring principle of the dynamic programming implementation of Levenshtein distance was modified as shown in Figures 15 and 16. The function "cost" in the modified version returns the operational penalty for a specific character, or characters in case of a substitution.

Original

```
dp[i][j] =
    min(dp[i - 1][j - 1] + 1, // Substitution
        dp[i - 1][j] + 1,     // Deletion
        dp[i][j - 1] + 1      // Insertion
    );
```

Figure 15

Modified

```
dp[i][j] =
    min(dp[i - 1][j - 1] + cost(standard[i - 1], client[j - 1]), // Substitution
        dp[i - 1][j] + cost(standard[i - 1], ""),                // Deletion
        dp[i][j - 1] + cost("", client[j - 1])                   // Insertion
    );
```

Figure 16

The time complexity of the entire calculation after this modification reduced slightly because the pre-processing procedure (with O(n) complexity for each string) was removed in the new version. However, since Levenshtein's dynamic programming procedure is the dominant factor, the overall complexity remained O(n*m), where n and m are lengths of the two words being compared.

4.4.1.4 Testing

After the first sprint was finished and the development was complete, Dr. Bailey and Dr. Speights Atkins employed a research assistant to test the computation of various transcription comparisons using the new version of APTct. The research assistant ran several computations using the tool and confirmed the new system's improvement in validity and usability. However, a vital segment of the algorithm for phonetic transcription comparison, the prioritization of vowel alignment, had not been implemented. Also, Dr. Bailey and Dr. Speights Atkins requested the addition of extIPA (extensions to the IPA) symbols to APTct, which are a set of letters and diacritics devised by the International Clinical Phonetics and Linguistics Association to augment the International Phonetic Alphabet for the phonetic transcription of disordered speech [25]. These changes form the basis for sprint two.

4.4.2 Sprint 2

This sprint is unarguably the most important because it features a priority control system to incorporate the nucleus alignment principle of phonetic transcription comparison in APTct's algorithm. The tool was also expanded in this sprint to include extIPA, which are additional letters and diacritics to supplement the IPA for the phonetic transcription of disordered speech.

4.4.2.1 Front-end

In this sprint, APTct's phonetic character set was augmented to support extIPA, which are symbol extensions to the International Phonetic Alphabet (IPA). They are particularly important for the phonetic transcription of disordered speech. To incorporate it into the front-end, a dropdown was added to the top portion of the keyboard. This dropdown indicated the current keyboard in use and allowed the users to choose between IPA and extIPA. Figures 17 and 18 demonstrate the modification.

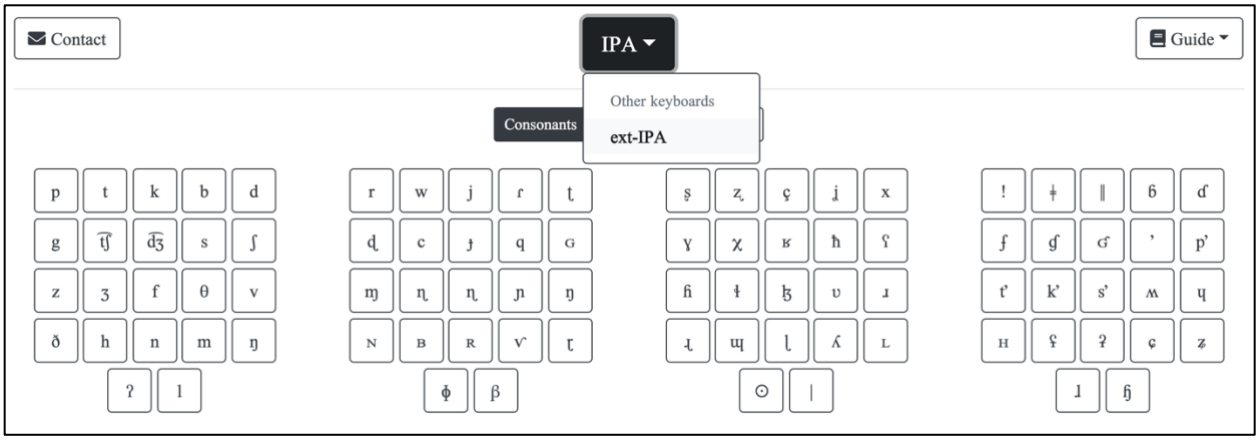

Figure 17

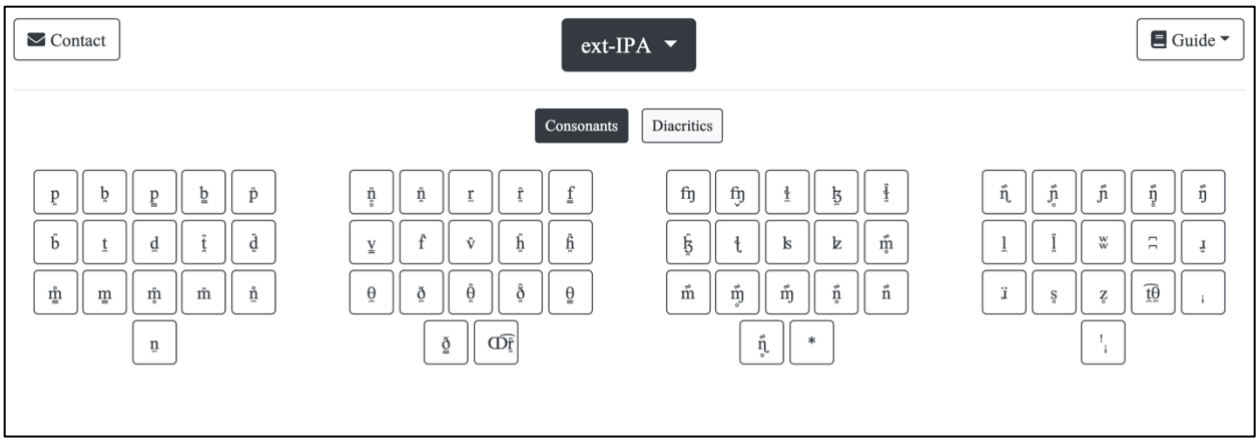

Figure 18

The most impactful change to APTct, however, was the addition of vowel alignment prioritization. Its theoretical and computational aspects will be covered in the algorithm section. In terms of the user interface, a "Details" button was added (see Figure 19) beside every transcription, clicking which will show users all the information about that specific comparison. This addition is vital because it shows the user precisely what steps/decisions were taken by the algorithm to reach the answer of comparison, i.e., the calculated distance.

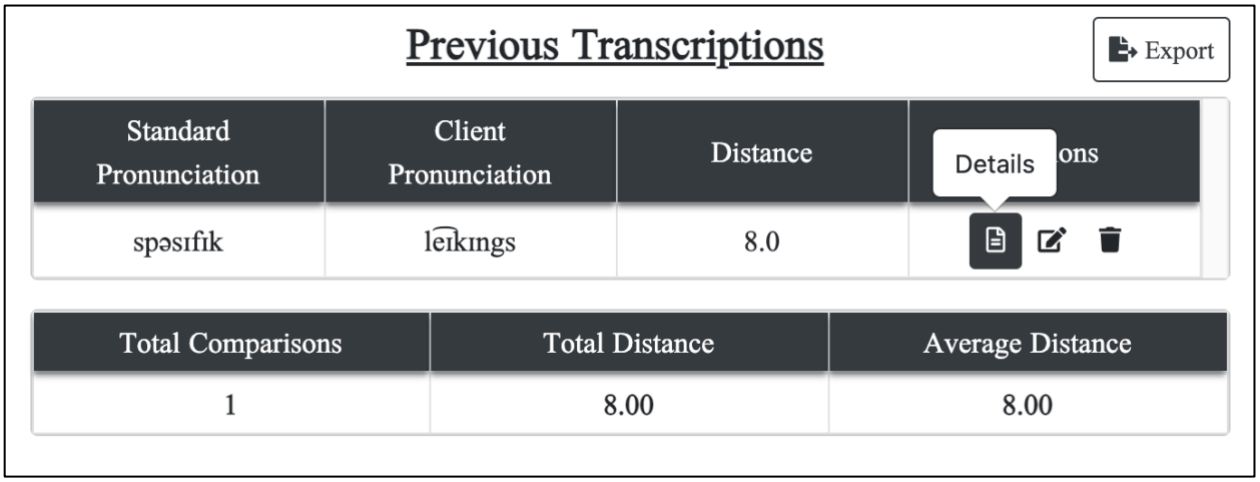

Figure 19

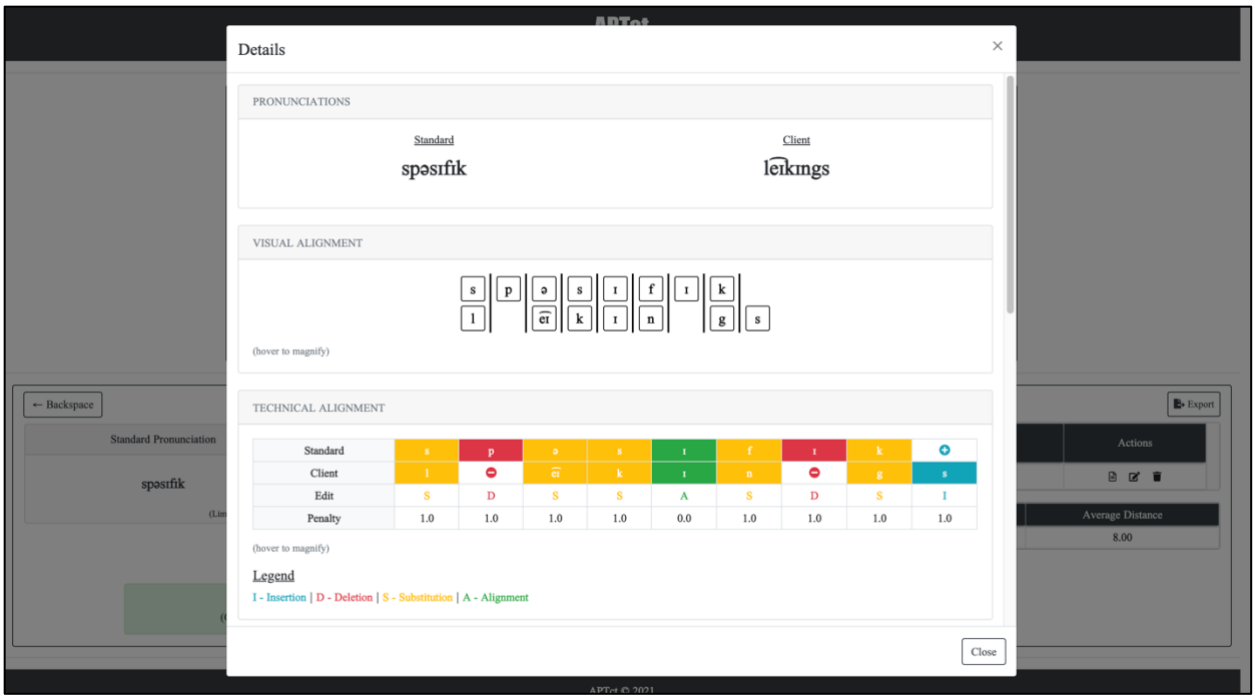

Figure 20

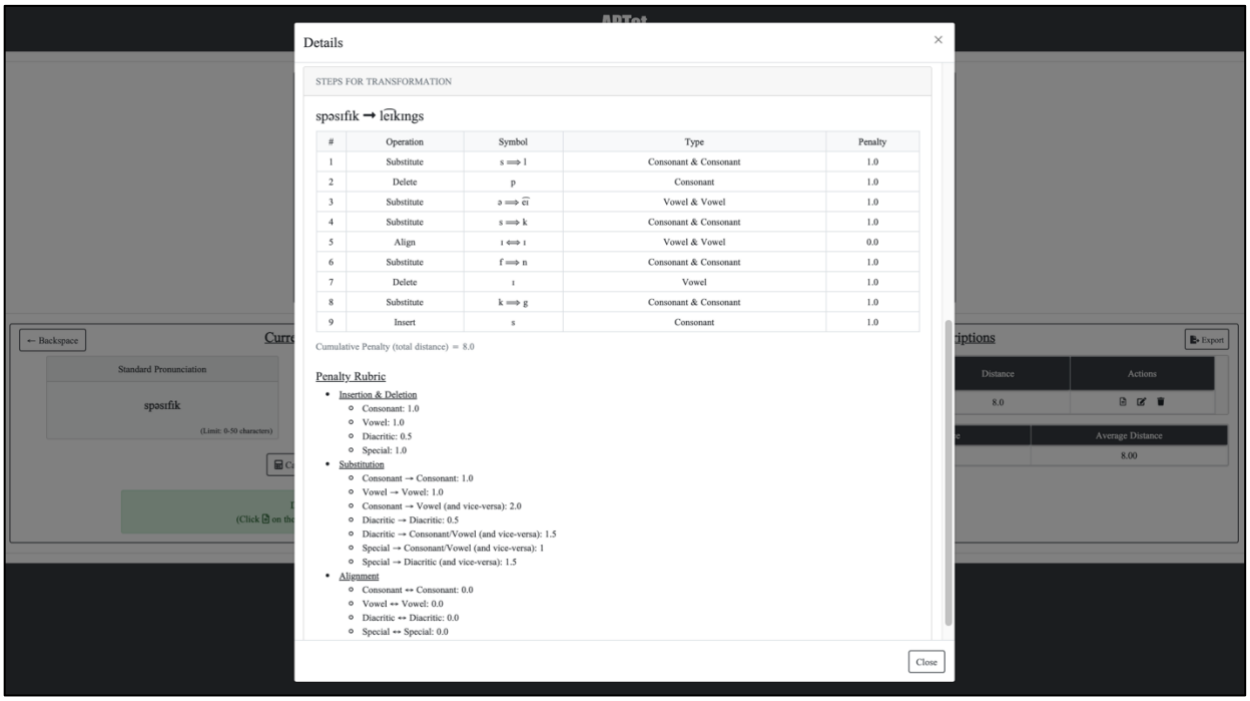

Figure 21

As shown in Figures 20 and 21, there are four sections provided for comparison details. The first section, “Pronunciation,” shows the two transcriptions being compared. The second section, “Visual Alignment,” displays the basic alignment between the two transcriptions. The third section, “Technical Alignment,” represents which operations cause the two transcriptions to align the way they align. The fourth section, “Steps For Transformation,” describes what steps have to be carried out to transform the first transcription into the second one. These details are imperative for users’ understanding of the tool. They are stored on the browser (using session storage functionality) until the user quits APTct’s tab or the whole browser.

There was another essential feature implemented under the hood as per Dr. Speights Atkins’ and Dr. Bailey’s suggestion. It was to allow users to type in English alphabets from the physical keyboards on their machine. One important point to note here is that only alphabets (a-z and A-Z) and action keys like tab, backspace, and shift were allowed to prevent the user from typing in unsupported symbols like a percentage sign (%) or an ampersand (&).

4.4.2.2 Back-end

To support phonetic transcription comparison for extIPA, extIPA symbols had to be added to the symbol store in the backend code to enable the server to perform string matches while operating the algorithm. To support the display of comparison details on the front-end, several result data fields were also added to the JSON output by the server, which previously consisted of only the calculated distance and any errors encountered during the computation.

4.4.2.3 Algorithm

This sprint is particularly important because the algorithm for APTct was expanded to support the nucleus alignment principle of phonetic transcription comparison.

Let us first understand what the nucleus alignment principle is. For example, let us assume two transcriptions, “bbbbi” and “ibbbb”. Each contains several consonants but only one vowel. If APTct’s non-prioritized algorithm was used to compare these two words, the calculated distance would be 2.0 because all the consonants would be aligned (penalty of 0.0), and a score of 2 would be obtained due to insertion and deletion of the vowel “i” (penalty of 1.0 for each). Note that this is also the minimum distance. Without any special modifications, Levenshtein edit distance will **always** yield the minimum score possible. However, Dr. Bailey and Dr. Speights Atkins specify that ideally, for phonetic transcription comparison, it is more important to ensure that vowels are aligned first before other things, regardless of whether the distance is minimized or not. Therefore, the minimum distance can be assigned a secondary order of importance to keep this prioritization of vowel alignment as the first priority. In our example, if vowel alignment is prioritized, then the calculated distance would be 8.0 because the vowels are aligned (penalty of 0.0), and there are four consonant insertions and four consonant deletions (penalty of 1.0 for each). Figures 22 and 23 demonstrate the comparisons.

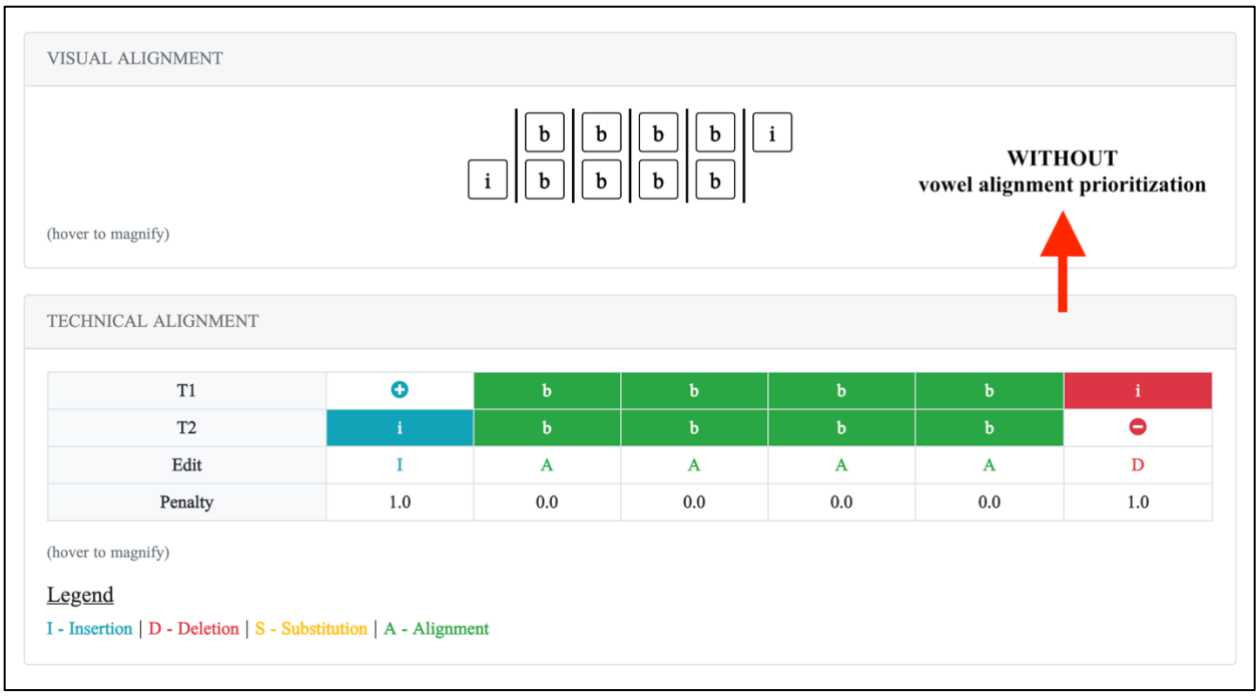

Figure 22

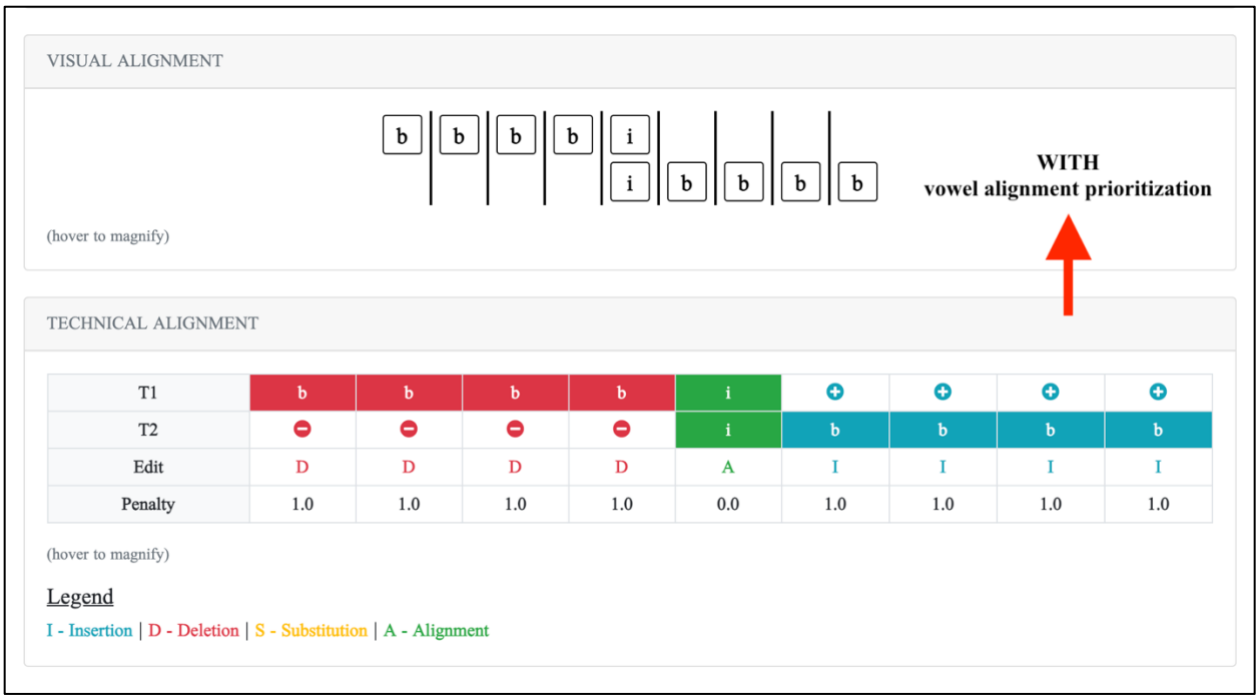

Figure 23

So how exactly is this prioritization of vowel alignment achieved? A novel and unique tabular priority control system was designed to solve this problem.

4.4.2.3.1 Tabular Priority Control Design

Our goal is to dictate and control the path of operations taken by the algorithm when it compares two words. In the original Levenshtein algorithm, this path is dictated by the distance values in the surrounding cells. It chooses an operation (substitution, deletion, or insertion) which yields the minimum penalty. It does not consider if one operation is prioritized over the other or if there is a preference for operations over certain characters than others. In the example above, the actual Levenshtein edit distance between “bbbbi” and “ibbbb” was 2.0 because it is the minimum distance. It does not factor in the requirement of giving more importance to the alignment operation of character “i”, because if it does, then the distance is not minimum anymore (it is 8.0, as shown in the example above).

To implement this operational priority control, a distance table and another table, which we can call the priority table, are used. This priority table holds the priority of each character operation as the algorithm proceeds. Whenever the algorithm has to decide which path to take, it consults values in the priority table instead of values in the distance table. The distance table merely holds the distance values of each character operation. It does not affect the decision path. When the algorithm completes, the value in bottom right corner cell of the distance table is the final calculated distance.

4.4.2.3.2 Pseudocode

Let's see how the Levenshtein edit distance is modified to include the priority table calculations.

Previously in sprint 1, a cost function was added to the original Levenshtein algorithm (see Figure 24) to properly factor in the penalties for different operations and characters, instead of all being 1.0. In the pseudocode below, functions "getSubstitutionCost", "getDeletionCost" and "getInsertionCost" return the specific penalties associated with the operation on a specific character category.

WITHOUT
Priority Control

```
distance[i][j] = min(
                    distance[i-1][j-1] + getSubstitutionCost(word1[i-1], word2[j-1]),
                    distance[i-1][j] + getDeletionCost(word1[i-1]),
                    distance[i][j-1] + getInsertionCost(word2[j-1])
                );
```

Figure 24

The pseudocode, after we add priority control to the algorithm, is shown in Figure 25.

WITH
Priority Control

```
1.  char1 = word1[i-1];
2.  char2 = word2[j-1];

3.  substitutionCost = getSubstitutionCost(char1, char2);
4.  deletionCost = getDeletionCost(char1);
5.  insertionCost = getInsertionCost(char2);

6.  substitutionPriority = priority[i-1][j-1] + substitutionCost + getSubstitutionPriority(char1, char2);
7.  deletionPriority = priority[i - 1][j] + deletionCost + getDeletionPriority(char1);
8.  insertionPriority = priority[i][j-1] + insertionCost + getInsertionPriority(char2);

9.  tempPriority = deletionPriority;
10. tempDistance = distance[i-1][j] + deletionCost;


11. if (tempPriority > insertionPriority) {
12.     tempPriority = insertionPriority;
13.     tempDistance = distance[i][j - 1] + insertionCost;
14. }

15. if (tempPriority > substitutionPriority) {
16.     tempPriority = substitutionPriority;
17.     tempDistance = distance[i - 1][j - 1] + substitutionCost;
18. }

19. priority[i][j] = tempPriority;
20. distance[i][j] = tempDistance;
```

Figure 25

In Figure 25, indices "i" and "j" represent the current iteration of the Levenshtein algorithm. At each iteration, the pseudocode above dictates how the distance and priority tables are filled, since it is dynamic programming at its core. Let us go over the code line by line.

- Lines 1-2: Characters from each word being compared.
- Lines 3-5: Functions "getSubstitutionCost", "getDeletionCost" and "getInsertionCost" return specific costs/penalties for that operation and character category (consonants, vowels or diacritics).
- Lines 6-8: These three lines determine the priority, instead of the distance, of each operation (with the characters involved). Just like the distance calculation, we look at the previous values in the priority table - diagonal (up & left) cell for substitution, vertical (up) cell for deletion, and horizontal (left) cell for insertion. Functions "getSubstitutionPriority", "getDeletionPriority" and "getInsertionPriority" return priorities

(a negative number in our case) for that operation and character category. Note that here, a lower priority value means higher prioritization. An interesting observation would be that substitution, deletion, and insertion costs are also added to the priority. This is because if the priorities of any two operations are the same, then we want to proceed with the one that produces the minimum distance.

- Lines 9-10: The comparison process for determining prioritization starts from here. The two variables, "tempPriority" and "tempDistance", keep track of the lowest priority value (highest prioritization) and the distance. Deletion priority and distance are assigned first.
- Lines 11-14: Deletion and insertion priority are compared. The one with the lowest value (higher prioritization) is chosen. Distance is updated accordingly.
- Lines 15-18: Operation chosen in lines 11-14 is then compared with substitution. Once again, the one with a lower priority value is picked. Distance is updated accordingly.
- Lines 19-20: Current cells for both priority and distance tables are updated with the chosen values.

Once the algorithm completes, the final edit distance (based on operational priority instead of minimum distance) can be obtained by retrieving the value in the bottom right corner of the distance table.

4.4.2.3.3 Results and Comparison

Let us consider our previous example, "bbbbi" and "ibbbb", and analyze the path taken by the algorithm. with and without priority control.

With the application of priority control, we obtain Table 1 (distance) and Table 2 (priority) as shown –

Distance table

	word 2 →	**i**	**b**	**b**	**b**	**b**
word 1 ↓	0.0	1.0	2.0	3.0	4.0	5.0
b	1.0	2.0	1.0	2.0	3.0	4.0
b	2.0	3.0	2.0	1.0	2.0	3.0
b	3.0	4.0	3.0	2.0	1.0	2.0
b	4.0	5.0	4.0	3.0	2.0	1.0
i	5.0	4.0	5.0	6.0	7.0	8.0

Table 1

Priority table

	word 2 →	**i**	**b**	**b**	**b**	**b**
word 1 ↓	0.0	1.0	2.0	3.0	4.0	5.0
b	1.0	2.0	1.0	2.0	3.0	4.0
b	2.0	3.0	2.0	1.0	2.0	3.0
b	3.0	4.0	3.0	2.0	1.0	2.0
b	4.0	5.0	4.0	3.0	2.0	1.0
i	5.0	-96.0	-95.0	-94.0	-93.0	-92.0

Table 2

Without the application of priority control, we obtain Table 3 (distance) as shown –

Distance table

	word 2 →	**i**	**b**	**b**	**b**	**b**
word 1 ↓	0.0	1.0	2.0	3.0	4.0	5.0
b	1.0	2.0	1.0	2.0	3.0	4.0
b	2.0	3.0	2.0	1.0	2.0	3.0
b	3.0	4.0	3.0	2.0	1.0	2.0
b	4.0	5.0	4.0	3.0	2.0	1.0
i	5.0	4.0	5.0	4.0	3.0	2.0

Table 3

It can be seen that controlling the priority of operations changes the path taken by the algorithm to reach the final edit distance, which may or may not be the minimum distance. In our example, we gave the vowel alignment operation a priority value of -100.0, and the others were given 0.0. We can see from Table 2, which is the priority table, that as soon as a vowel alignment is detected, its corresponding path is chosen since it has the highest priority or the least value, which is -96.0 (-100.0 + 4.0). To recreate the sequence of edits, we traverse backward from the bottom right corner of the table to the top left corner. However, instead of operating on the distance table like in the vanilla Levenshtein algorithm, the procedure is carried out on the priority table. Although the arrows shown in the tables go from the top-left to the bottom-right corner, the backtrack is carried out from bottom-right to top-left and then reversed at the end to give us the final list of edits.

4.4.2.3.4 Complexity Analysis

Let us discuss the time and space complexity of our tabular priority control modification to Levenshtein edit distance. The time complexity of basic Levenshtein edit distance is O(n*m), where n and m are lengths of the two words being compared. If we notice, the priority control design does not affect the time complexity at all. It still stays O(n*m) because all the extra calls/operations that we have added in the modification have an individual complexity of O(1), which is constant. The space complexity of basic Levenshtein edit distance is also O(n*m) because we have to build the distance table to reach the final edit distance. There is an extra O(n*m) space for storing the priority table along with the distance table in our version of priority controlled edit distance. But the overall complexity still stays O(n*m) because O(n*m) + O(n*m) = O(n*m).

4.4.2.3.5 Tabular Priority Control Modifications and Applications

There are various tweaks and variations that can be added to the priority control design specified above. For the nucleus alignment principle, we give vowel substitution a priority value of -100.0 and other operations 0.0. Since our priority control design is quite open-ended, it can be configured to fulfill any use case. For example, let us say we want consonant alignments to take precedence over vowel alignments if they are more than a certain count, let us say five. Instead of adding unnecessary code to enforce such a rule, we can easily set the priority value of consonant alignments to -20.0. Therefore, if there is only one vowel alignment and more than five consonant alignments, six for instance, prioritization will switch from vowel to consonant because the priority value for vowel alignment (-100.0) will become larger than the priority value for consonant alignments (-20.0 * 6 = -120.0). Therefore, this priority control framework can also be structured

to take into account the effect of cumulative operations rather than focusing solely on individual operations.

The priority control design described in this section is highly adjustable; it can find applications in several other dynamic programming or table-based algorithms. For example, in the famous coin-change problem, we can apply the priority control technique to prioritize the selection of certain coins over others. Another famous problem, 0-1 Knapsack, can also potentially use the priority control mechanism in a real-life scenario to prioritize selecting certain items over others and not just basing the decision on the weight or value of those items. Apart from such examples, one can also consider applying the mechanism to consequential problems like genomic sequence alignments, where prioritizing the alignment of specific pairs over others can prove very useful. In fact, since such sequences tend to be huge, it is also advantageous if the algorithm operating over them works in an optimal time frame. This is not an issue since the priority control mechanism does not add unnecessary complexity in its implementation.

Because of its versatility, the possibilities of configurations and applications are perhaps endless.

4.4.2.4 Testing

Dr. Bailey and Dr. Speights Atkins employed a graduate research assistant to thoroughly test the modified version of the algorithm by running several hundred comparisons, if not more. Various theoretical aspects of phonetic transcription comparison were also tested, like strict order principle, nucleus alignment principle, phoneme modification diacritic scoring, stress assignment

diacritic scoring, etc. The algorithm performed well in all these categories, in addition to the general comparisons, and the results matched with hand computations. A usability test for the user interface was also carried out.

4.4.3 Sprint 3

For the third sprint, Dr. Speights Atkins and Dr. Bailey wanted to rearrange the layout of the screen so that the left half contained the keyboard, and the right half contained all the transcriptions.

4.4.3.1 Front-end

Previously, the top portion of the application comprised the keyboard, and the bottom portion contained all the transcriptions. After some altering and refining, the keyboard was shifted to the left side, and the remaining sections of the tool were moved to the right side in the new interface.

4.4.3.2 Back-end

No modification.

4.4.3.3 Algorithm

No modification.

4.4.3.4 Testing

Usability tests were carried out for the changes in the interface.

4.4.4 Sprint 4

Dr. Bailey and Dr. Speights Atkins wanted to make the tie bars and syllabic diacritics independent from consonants and vowels (affricates, for instance) for this sprint.

4.4.4.1 Front-end

Figure 26 shows the keys which were removed from the keyboard –

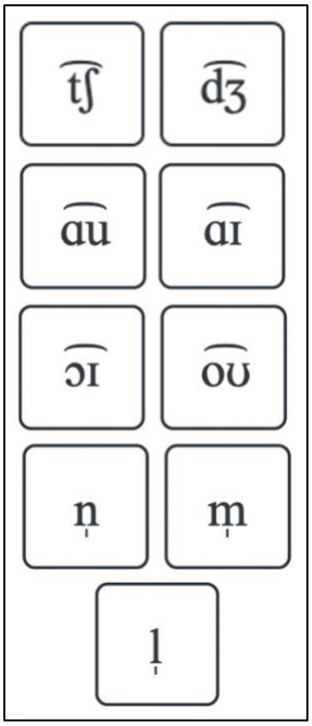

Figure 26

Instead of having pre-combined keys like them, the goal was to allow tie bars and syllabic to be combined with any symbol desired. Therefore, after removing the keys shown in Figure 26, tie bar and syllabic keys (see Figure 27) were added to the keyboard.

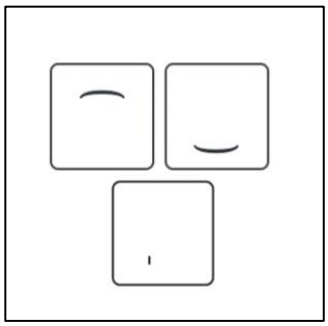

Figure 27

4.4.4.2 Back-end

No modification.

4.4.4.3 Algorithm

To allow the tie bars to group consonants/vowels together and syllabic diacritic to modify a consonant into a vowel, a preprocessing step had to be added before the Levenshtein edit distance could be applied. This is because the related characters have to coalesce into a single string before the distance calculation procedure can begin, as the algorithm matches these string signatures against a symbol store to establish a symbol category. There are three string traversals in the preprocessing step. The first one is for the syllabic. Rest two are for the two tie bars. For the syllabic diacritic, if the character after it is a consonant, then these two characters are grouped into a single unit as a vowel. The algorithm takes into consideration the occurrence of diacritics around these characters and handles those cases accordingly. For each tie bar, if the character before and after it belongs to the same category (either a consonant or a vowel), then these three characters are grouped into a single unit. Again, the algorithm also takes into consideration the occurrence of diacritics around these characters and handles those cases accordingly.

There are three string traversals, one for syllabic and two for tie bars. The overall time complexity for the preprocessing step for each string is, therefore, O(n), where n is the length of the string.

4.4.4.4 Testing

Dr. Bailey and Dr. Speights Atkins personally tested several computations involving tie bars and the syllabic diacritic and found the tool to be consistently producing the right results, even for some devilish calculations.

4.4.5 Sprint 5

After using the tool (resulting from the fourth sprint's development) for a while, Dr. Bailey and Dr. Speights Atkins found the transcribing process to be a bit slow due to constant switching between consonant, vowel, and diacritic tabs. This sprint, therefore, features a major keyboard interface change.

4.4.5.1 Front-end

There were several small-scale and large-scale changes brought about on the front-end of APTct. The most significant modification was the keyboard. Instead of a tab-based layout in the previous sprints, the keyboard was completely modified and made to look like the official IPA chart as much as possible (see Figure 28). This was done to prevent time wastage that a user might have faced when switching tabs to find relevant symbols for transcription. The same was done for the extIPA section of the keyboard. It was also changed and made to resemble the official extIPA chart as much as possible.

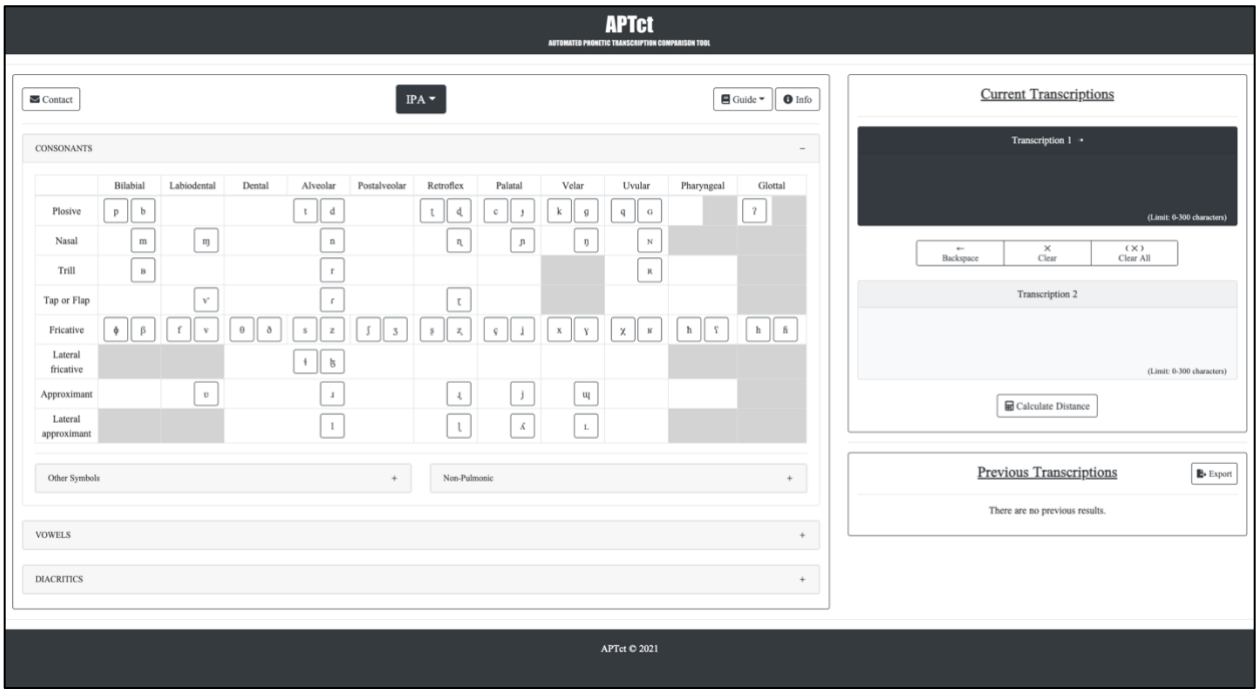

Figure 28

Care was also taken to preserve the aspect ratio of the keyboard on smaller screens. This was done by making the chart keyboard automatically draggable on small screens so that a user can simply drag by clicking anywhere inside the chart keyboard instead of manually dragging the horizontal scroll bar. Collapsible sections were added to only show the relevant parts of the keyboard at a time. Website wording was changed from “Standard” and “Client” to “Transcription 1” and “Transcription 2” in order to make everything more generalized. On the transcription side of the layout, a “Clear All” button was added to facilitate the clearing of fields for transcription one and two in just a single click. Another essential and subtle feature implemented was to make the right side of the keyboard (which contains the transcription fields) float and move along with the left side. This was done to ensure that users can see what they are typing at all times, especially

when all the sections are open, and a user would have had to scroll all the way down to choose their desired symbols.

4.4.5.2 Back-end

Error handling was made more robust for unsupported characters in a transcription. Also, Google Analytics was added to the website for a detailed analysis of user traffic and engagement.

4.4.5.3 Algorithm

No modification.

4.4.5.4 Testing

Since this was the final sprint for APTct's development, Dr. Bailey and Dr. Speights Atkins, along with their trained research assistant, ran several comprehensive tests on the tool, including usability studies to assess the quality of transcriptions and the algorithm. They were delighted with the obtained results and approved the tool to be published at the production level.

CHAPTER 5: EVALUATION AND RESULTS

After all the sprints were developed, the final version of APTct was made available at the following URL - https://aptct.auburn.edu/. The landing page (see Figure 29) provides useful information about the tool and the authors. The calculator page contains the actual tool (see Figure 30).

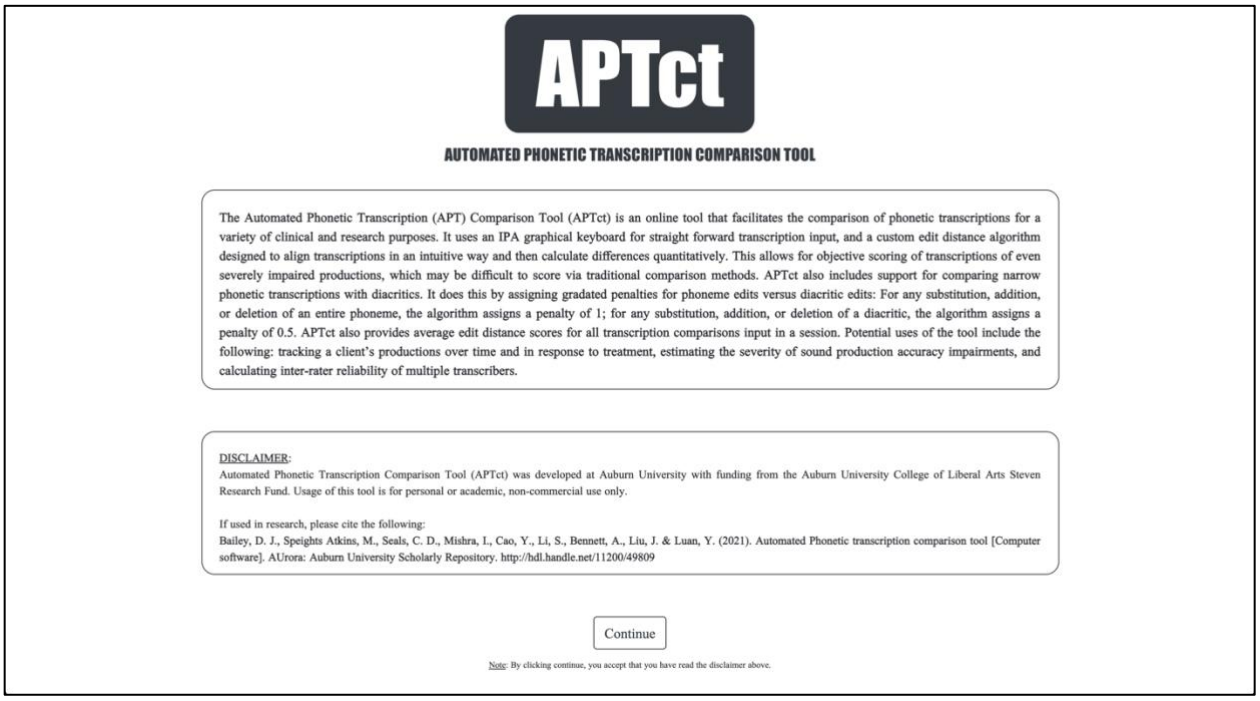

Figure 29

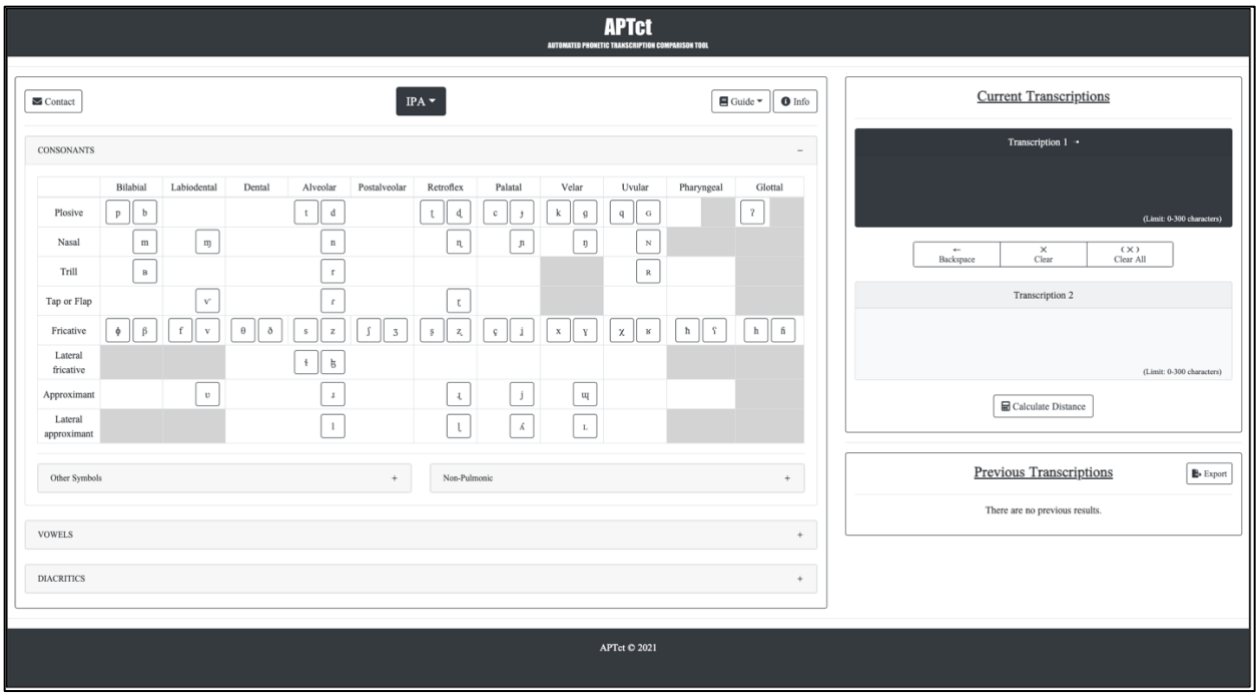

Figure 30

To gauge the performance of APTct in producing correct results for phonetic transcription comparisons, Dr. Bailey and Dr. Speights Atkins, along with their research assistant, utilized the tool to perform numerous comparisons. They checked several fundamental aspects of the algorithm powering APTct. The accuracy, resulting from the outcome of testing, was very high. There were a few inconsistencies, but those can be mitigated by better training of the research assistant and further improving the user interface.

After testing, Dr. Bailey and Dr. Speights Atkins submitted APTct to Clinical Linguistics & Phonetics, an international journal publishing research into linguistics and phonetics, including speech and language disorders, hearing impairment, and sign language [26]. The journal has accepted the paper and published it internationally [27]. A few test results included in the paper are demonstrated in Figures 31 - 35.

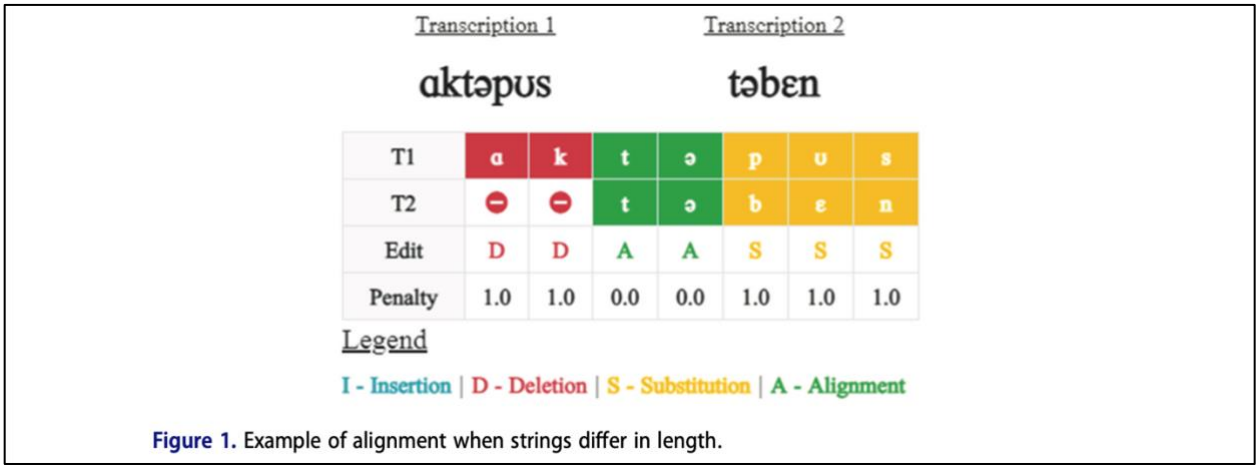

Figure 1. Example of alignment when strings differ in length.

Figure 31

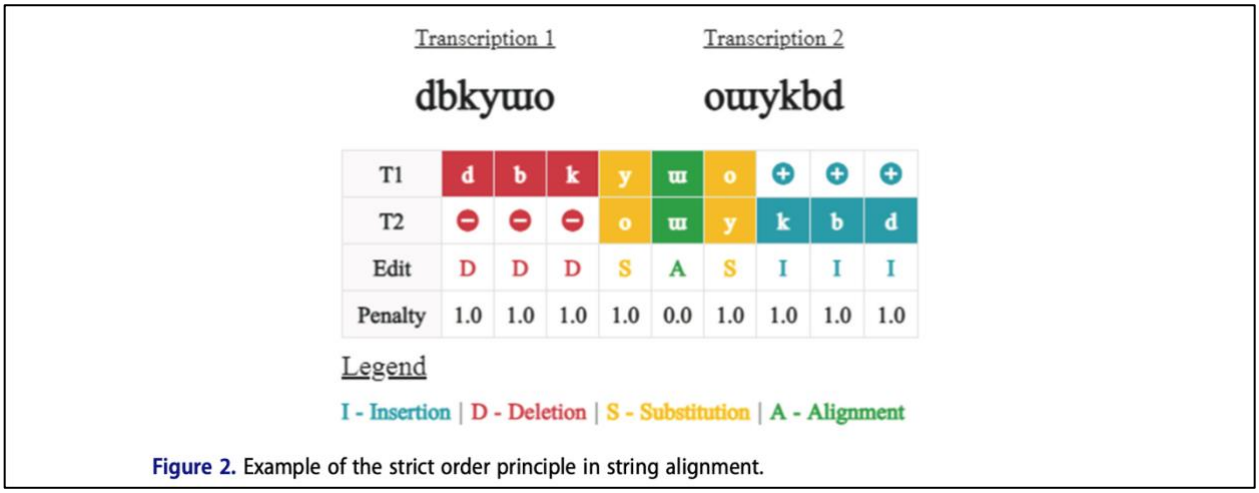

Figure 2. Example of the strict order principle in string alignment.

Figure 32

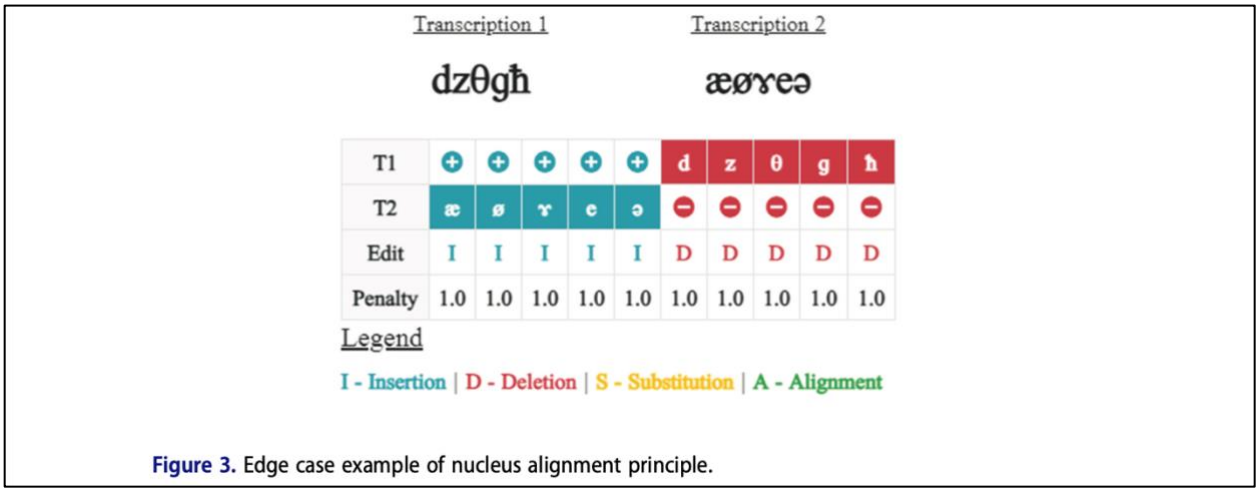

Figure 3. Edge case example of nucleus alignment principle.

Figure 33

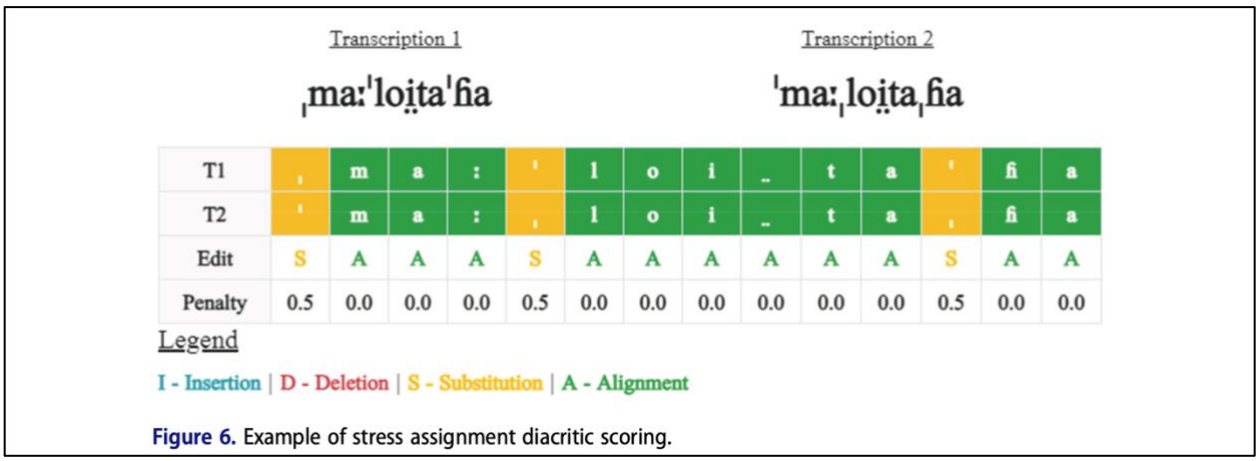

Figure 6. Example of stress assignment diacritic scoring.

Figure 34

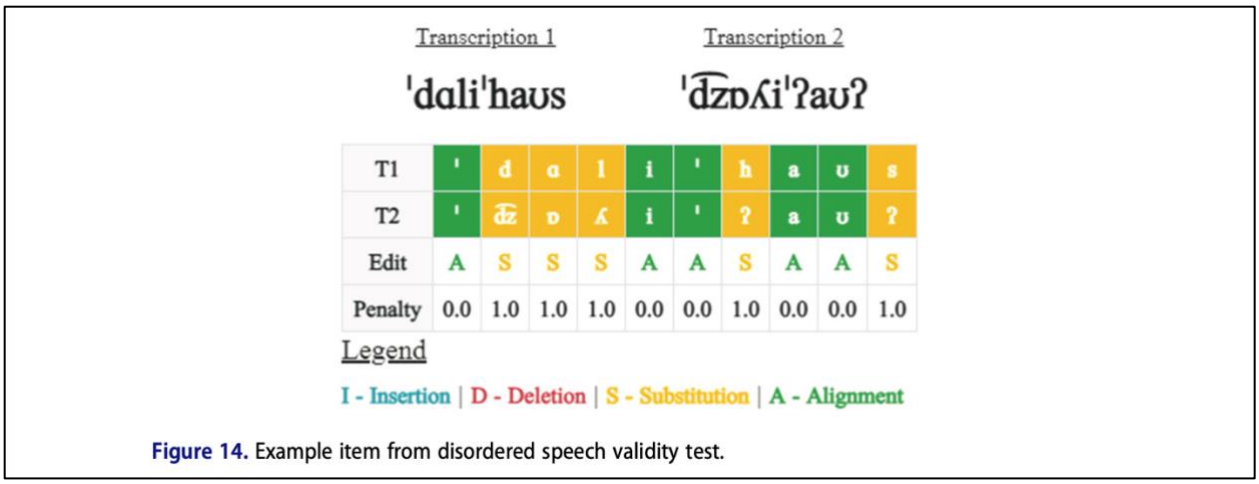

Figure 14. Example item from disordered speech validity test.

Figure 35

Validity tests performed on the tool also bolster the fact that the tool itself can serve as a viable replacement for hand calculations of phonetic transcription comparisons and that the integrity of results can be trusted. Figures 36 and 37 display the results of disordered speech sample calculation and dialect calculation tests carried out by Dr. Bailey and Dr. Speights Atkins, with assistance from their research assistant.

transcription 1			transcription 2		hand calculated edit distance	as computed by APTct 12/17/2020	input errors?
english	askhatʰubɹĩŋðiːsʂĩnʐwɪθhɜfɹʌmðəstɔə	3	farsi1	aːshəɹtubɹĩŋɖɪstĩŋkswɪfhəɹfɹ5msɖəstɔɹ	21	21	
english	æskhəˈrəbɹĩŋŋiːzθĩnʐwɪθhəˈfɹʌ̃mɳəstɔɹ	144	xiang1	æshəɹtubɹĩŋðisθĩɳswɪθhɜfɹʌmdistɔʷə	17	18	
english	æskəˈrəbɹĩŋðiːzθĩnzwɪθhəˈfɹʌ̃mðəstoə̃ɹ	14	bengali1	ækshoɹtubɹĩŋɖiʐθɪɳswiθhɜfɹɑ̃mðəstoɹ	17	17	
english	æskhaɹtʰubɹĩŋðiːzθĩnzwɪθhɜfɹʌ̃mðəstɔɹ	106	bai1	askhətubɹiŋðiːzθiɳkswɪθəhɜfɹʌɳðəstɔ	14	14	
english	ææskhəɹrörbɹĩŋðiːsʂθĩnzwɪθhəɹfɹə̃mðə̃stoəɹ	23	spanish1	æksheɹtubrĩŋɖisθĩɳswiθhɛɹfrɑ̃mɖɛstoɹ	19	21	yes
english	aʂ̈skhɜɹəbɹɪŋðiːzʂɪnzwɪθhɜɹfɹəmðəstɔːɹ	53	bafang1	æksɦɜːtubrĩŋɖisʈĩnʐwɪʇhəfrɔ̃mɖəstɔː	21	20.5	
english	æskhətubɹɪŋðiːzθĩnʐwɪðhəˈfɹʌ̃mðəstɔɹ	29	hungarian1	æskhətubrĩŋɖisʈĩnʐwiʇhərfrɔ̃mɖəstɔr	18.5	17.5	
english	eskərəbɹĩŋŋɪɪzθĩnːzwɪðʒfɹʌ̃mməstɔː	84	korean1	ɛskətubɹĩŋdissĩɳswiθhɜɹfɹɔ̃mɖəstɔ	24	21	
english	æskəˈtübɹĩŋðɪːzθĩnzwɪθəˈfɹɔ̃mðəstɔɹ	121	punjabi1	askhəɹtubriŋɖiʐtiŋʐvɪdhɑːfɹʌmdistɔɹ	24	24	
english	æʂkhəˈrəbɹĩðiːzθĩnzwɪθhɛɹfɹʌ̃mðəstoːɹ	5	portugese1	ʔʔəskəhɛ̃ʇubrĩŋdɛfɪŋkwizhɛ̃fɔ̃mistõrĩ	27	28	yes

Figure 36

Ball, 2005 Transcription Exercises 19.1, 19.2, 19.3	transcription 1	transcription 2	orig hand calc	orig APTct calc			revis hand calc	revised APTct calc	
creaking crane	ˈkɹikɪŋˈkɹeɪn	ˈtwitɪnˈtweɪn	6	6	y		6	6	y
stripey tiger	ˈstɹaɪpiˈtaɪgə	ˈdaɪpiˈdaɪdə	6	6	y		6	6	y
five fishes	ˈfaɪvˈfɪʃɪz	ˈpaɪbˈpɪtɪd	5	5	y		5	5	y
roaring lion	ˈɹɔɹɪŋˈlaɪən	ˈwɔwɪnˈjaɪən	4	4	y		4	4	y
yellow lily	ˈjɛloʊˈlɪli	ˈjɛjoʊˈjɪji	3	3	y		3	3	y
red robin	ˈɹɛdˈɹɑbɪn	ˈʊɛdˈʊɑbɪn	2	2	y		2	2	y
shoe shop	ˈʃuˈʃɑp	ˈçuˈçɑp	2	2	y		2	2	y
sizzling	ˈsɪzlɪŋ	ˈɬɪlʒɪŋ	3	4	n	user input error (ligature, n for eng)	3	4	n
sister	ˈsɪstə˞	ˈθɪθtə˞	2	2	y		2	2	y
slush	ˈslʌʃ	ˈʃlʌç	2.5	2.5	y		2.5	2.5	y
Susie	ˈsuzi	ˈɬuʒi	2	2	y		2	2	y
zebras	ˈzibɹəz	ˈðibɹəð	2	2	y		2	2	y
snowman	ˈsnoʊmæn	ˈn̥ʊʊmæ̃	3	3	y		3	3	y
elephant	ˈɛləfənt	ˈɛzəɸəmp	4	4	y		4	4	y
christmas tree	ˈkɹɪsməsˈtɹi	ˈcɥɪçməçˈcɥi	7	6	n	hand calculation error	7	6	n
dolly house	ˈdɑliˈhaʊs	ˈd̪ɒʌiˈʔaʊʔ	6	5	n	hand calculation error	5	5	y
television	ˈtɛlɐvɪʒən	ˈʔɛləʊɪçə̃	4.5	4.5	y		4.5	4.5	y
strawberry jam	ˈstɹabɛɹiˈd͡ʒæm	ˈt̪ɸɔbwiˈɖ̪ðæm	11	8.5	n	hand calculation error	10.5	8.5	n
happy snake	ˈhæpiˈsneɪk	ˈxæʔiˈŋeɪʔ	4.5	4.5	y		4.5	4.5	y
magic dragon	ˈmæd͡ʒɪkˈd͡ʒɹægɪn	ˈmæɻɪcˈdɮæɻən	7	7	y		7	7	y
jigsaw puzzle	ˈd͡ʒɪgsəˈpʌzl̩	ˈdɪɖθɔˈpʌɖ̪ðəl	10	7	n	hand calculation error	9	8	n
nasty noise	ˈnæstiˈnɔɪz	ˈnæñɪˈnɔɪ̃ʔ	8.5	5.5	n	hand calculation error	8.5	5.5	n

Figure 37

It is advantageous to use APTct instead of comparing phonetic transcriptions by hand, as APTct can save a researcher's valuable time and is much less prone to errors. Dr. Speights Atkins and Dr. Bailey also carried out usability tests on APTct, and the results indicated that usability was increased manifold from the previous versions of the tool. The new version offers a much cleaner interface, and since the on-screen keyboard closely resembles the official IPA chart, it becomes easier for a researcher to transcribe speech sounds using the tool and compare them against each other.

Additionally, APTct also underwent security testing to check the tool's response against invalid/illegal characters and other possible attacks. By default, invalid characters cannot be entered through the on-screen keyboard or the physical keyboard as the tool is programmed to ignore those inputs if they occur. The website's JavaScript code was modified (as an attacker would) to include a few illegal characters as input. When the data is sent to the backend, the server detects those illegal inputs and returns a relevant error message immediately, preventing any

bypass. A few vulnerability scanners like Burp Suite, Nikto, etc., were also run on APTct, but no serious threat or security loophole was detected.

Furthermore, there is faster performance on the front end due to the implementation of AJAX. The website works without reloading the page and functions seamlessly. The new codebase is also substantially cleaner than the previous versions and more succinct, making it easier for developers to work on the tool in the future.

Overall, Dr. Bailey and Dr. Speights Atkins were quite satisfied with the final version of APTct, and even more so with it being published in the Clinical Linguistics & Phonetics journal.

CHAPTER 6: CONCLUSION AND FUTURE WORK

It is through technology that human beings transcend the past and enter the future. APTct was designed and developed with a similar goal. It was to leverage the power of computing technology to automate phonetic transcription comparisons and eliminate the need to perform such arduous computations via hand, which had been the norm until now.

APTct provides a clean user interface and versatile functionality for comparing transcriptions. It hosts an exquisite visualization feature that allows the user to see exactly how two transcriptions compare against each other. Apart from transcription utilities, it also enables a user to export their comparison results for research and other valuable purposes. Currently, APTct supports the complete IPA symbol set for transcription purposes. There is support for extIPA symbols as well for transcription of disordered speeches. It is available for free at the following link – https://aptct.auburn.edu.

The development of APTct led to the invention of an innovative and unique priority control system. This system was instrumental in allowing APTct to feature the nucleus alignment principle in its calculations. However, this priority control design goes way beyond just APTct. It can be applied to a plethora of tabular algorithms, dynamic programming ones being the best candidates. Although APTct is a comprehensive tool, it still has a few limitations and needs future work to improve further. Speech transcriptions prepared elsewhere cannot be copied into APTct; therefore, a copy-paste functionality remains to be implemented. The general screen layout is still far from

perfect. Ideally, all the components should be modular and freely movable on the screen so that the user can adjust the layout according to their convenience for a better transcription experience. In more advanced future versions of APTct, a user account system can also be added, allowing individual users to save and track their progress online.

The automation framework presented in this research and the novel priority control design can hopefully fuel the momentum for incorporating more technology into linguistics and other related sciences. After all, APTct is one small step in computer science, one giant leap for phonetics.

REFERENCES

[1] https://en.wikipedia.org/wiki/Linguistics

[2] https://en.wikipedia.org/wiki/International_Phonetic_Alphabet

[3] Ball, M., Muller, N., Klopfenstein, M., & Rutter, B. (2009). The importance of narrow phonetic transcription for highly unintelligible speech: some examples. *Logopedics Phoniatrics Vocology*, *34*(2), 84-90. (https://doi.org/10.1080/14015430902913535)

[4] Kondrak, G. Phonetic Alignment and Similarity. Computers and the Humanities 37, 273–291 (2003). https://doi.org/10.1023/A:1025071200644

[5] Thorlacius, L. (2007). The Role of Aesthetics in Web Design. Nordicom Review. 28. 10.1515/nor-2017-0201.

[6] Ping Zhang, R. V. Small, G. M. von Dran and S. Barcellos, "Websites that satisfy users: a theoretical framework for Web user interface design and evaluation," Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences. 1999. HICSS-32. Abstracts and CD-ROM of Full Papers, Maui, HI, USA, 1999, pp. 8 pp.-, doi: 10.1109/HICSS.1999.772668.

[7] https://en.wikipedia.org/wiki/Two-factor_theory

[8] Nebeling, M. & Norrie, M.C. (2013) Responsive Design and Development: Methods, Technologies and Current Issues. In: Daniel F., Dolog P., Li Q. (eds) Web Engineering. ICWE

2013. Lecture Notes in Computer Science, vol 7977. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-39200-9_47

[9] I., Waseem & Hammouri, A. (2016). Responsive Web Design Techniques. International Journal of Computer Applications. 150. 18-27. 10.5120/ijca2016911463.

[10] Armstrong, E. (2015) IPA keyboards for iOS, Voice and Speech Review, 9:1, 96-102, DOI: 10.1080/23268263.2015.1062598

[11] Dobrowolski, P. (2015). Complete IPA Keyboard for iOS devices. In ICPhS.

[12] Bae, B., Kang, S. S., & Hwang, B. Y. (2012). Edit distance calculation by phonetic rules and word-length normalization. In Proceedings of the European Computing Conference (ECC'12) (pp. 315-319).

[13] Karadimitriou, K. & Kraft, D. (2000). Genetic Algorithms And The Multiple Sequence Alignment Problem In Biology.

[14] Singla, N. & Garg, D. (2012). String Matching Algorithms and their Applicability in various Applications. International Journal of Soft Computing and Engineering (IJSCE). 1.

[15] Themistocleous, C., Neophytou, K., Rapp, B., Tsapkini, K. A Tool for Automatic Scoring of Spelling Performance. J Speech Lang Hear Res. 2020 Dec 14;63(12):4179-4192. doi: 10.1044/2020_JSLHR-20-00177. Epub 2020 Nov 5. PMID: 33151810.

[16] Kunath, S. & Weinberger, S. (2009). STAT: speech transcription analysis tool. 10.3115/1620959.1620962.

[17] https://en.wikipedia.org/wiki/Edit_distance

[18] Vitevitch, M. S., & Luce, P. A. (2004). A web-based interface to calculate phonotactic probability for words and nonwords in English. Behavior research methods, instruments, & computers: a journal of the Psychonomic Society, Inc, 36(3), 481–487. https://doi.org/10.3758/bf03195594

[19] https://en.wikipedia.org/wiki/Scrum_(software_development)

[20] https://tomcat.apache.org/connectors-doc/

[21] https://tomcat.apache.org/tomcat-9.0-doc/virtual-hosting-howto.html

[22] https://jquery.com/

[23] https://software.sil.org/doulos/

[24] https://en.wikipedia.org/wiki/Edit_distance

[25] https://en.wikipedia.org/wiki/Extensions_to_the_International_Phonetic_Alphabet

[26] https://www.tandfonline.com/toc/iclp20/current

[27] Bailey, D.J., Speights Atkins, M., Mishra, I., Li, S., Luan, Y. & Seals, C. (2021) An automated tool for comparing phonetic transcriptions, Clinical Linguistics & Phonetics, DOI: 10.1080/02699206.2021.1896783