SAPIENZA
UNIVERSITÀ DI ROMA

Sapienza University of Rome

Department of Information Engineering, Electronics and Telecommunications (DIET)
PhD in Information and Communication Technology (ICT)

Submitted in partial fulfillment of the requirements for the degree of Doctor Of Philosophy in Information and Communication Technology (PhD in ICT)

From detailed acoustic analysis to AI: designing and developing advanced speech analysis tools

Thesis supervisor
Prof. Maria-Gabriella Di Benedetto

Candidate
Kaleem Kashif
1994085

Academic Year 2023-2024 (37th cycle)

"In the world of sound, even the smallest details can reveal big truths. Through innovation and relentless inquiry, we unlock the secrets of human speech and transform the art of acoustic analysis."

(Kaleem Kashif & Tayyaba Anam)

Dedication and Acknowledgments

I would like to dedicate this thesis to my parents, whose unwavering and unconditional support has been the cornerstone of my journey. I am deeply grateful for their encouragement and belief in me. I am profoundly thankful and dedicate to my Dear wife Tayyaba Anam, for her unwavering support, love, and patience throughout this journey. I also express my sincere gratitude to Sapienza University Rome for providing a fellowship and to the International Office of Sapienza University of Rome for their assistance, which has been instrumental in the completion of my studies. My deepest appreciation goes to my supervisor, Professor Maria-Gabriella Di Benedetto, whose guidance, insights, and expertise have been crucial in shaping this thesis. Also, I would like to thank Professor Luca De Nardis for his valuable support. Also thanks to my lab mate, Usman Ali, and a special thanks to my dear friend, Tang Wen Qiong, whose constant support and friendship have been invaluable. This achievement would not have been possible without each of you.

Kaleem Kashif, Rome, October 2024

Abstract

The modernization of the xkl software, originally developed by Dennis Klatt at MIT in the 1980s, was a major goal of this research work. The introduction of a new Graphical User Interface (GUI), using GTK libraries, simplified the installation process but most importantly made the software accessible and user-friendly on various platforms, including Windows, Linux, and MacOS. The xkl refurbishment also addressed the inclusion, in the spectrum processing tools of the so-called reassigned spectrogram, allowing thus for improved detailed examination of speech spectra. In the current xkl version, formant values are now automatically saved in a text file, which facilitates large-scale analysis, especially for vowels studies. The development of a modern xkl speech analysis tool was part of the LaMIT project [1], that has the goal of applying Stevens lexical access model [2] to the Italian language. One major innovation introduced by Stevens is the concept of landmark, that is, the presence of privileged regions in the time domain at which a primary phase of the perceptual process would take place, the landmark positions. In this work, an automatic vowel landmark detector was developed. This landmark recognition system was developed and implemented based on a Convolutional Neural Network combined with a Recurrent Neural Network, i.e. a CNN-RNN hybrid model. The CNN-RNN recognizer used a set of parameters that combined energy measurements and Mel-spectrum descriptors, and was run on the above sentences. The recognizer was tested on sentences of the LaMIT database [3], a corpus formed by 800 spoken utterances (4 native italian speakers) that were manually analyzed by examining the corresponding speech waveforms but most importantly using the xkl tool that provided invaluable information on spectral general properties and time-varying spectral properties. It is thanks to this analysis that the corpus was manually labeled and contains now information about landmark presence, landmark type, and landmark position in time. The output of the recognizer produced an estimation of detected vowel landmarks. This output was compared against the manually estimated vowel landmark presence. The overall recognition rate was 74.98%. For individual speakers the recognition rate ranged from about 72% to about 77%. Artificial intelligence methods were also applied to automatic foreign accent identification [4]. A Multi-Kernel Extreme Learning Machine (MK-ELM) model, along with a weighted scheme, was proposed for application to the recognition of 5 different accents (Arabic, Chinese, Korean, Spanish, French) in American English. The recognition was based on Mel-frequency cepstral coefficients (MFCC) and prosodic features (Pitch, Energy). The proposed model achieved an accuracy rate of 84.72% using a paired weighting scheme. In contrast, the accuracy rate dropoped to 66.5% when employing the traditional non-weighted multi-classification scheme. A comparison against other other state-of-the-art classification methods showed significant advantages of the proposed model.

Keywords: Speech Processing, Reassigned Spectrogram, Speech Analysis Tool (xkl), Speech Spectral Analysis, GTK User Interface (GUI), Automatic Landmark Detection, Speech Recognition Systems, Automatic Identification of Foreign Accents (AIFA)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The research reported in this thesis was conducted in part within the xkl project, a collaborative initiative between Sapienza University, Rome, Italy, and the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, with a primary focus on enhancing the xkl tool, originally developed by Dennis Klatt in the 1980s. In parallel, the research extended to the automatic detection of vowel landmarks, and the detection of foreign accents.

Speech analysis, and in particular the accurate estimation of its temporal and spectral characteristics, is a major research challenge because of the time-varying nature of speech. Over the past 50 years, researchers have focused on refining techniques for instantaneous spectral estimation. The xkl software, despite being a groundbreaking tool for addressing these issues, faced limited adoption in recent decades due to its outdated graphical user interface (GUI) and lack of compatibility with modern computing platforms. Updating the tool offers the potential to unlock new discoveries by providing researchers with advanced capabilities for precise speech analysis.

This research also explores the development of an automatic vowel landmark detection system using modern deep learning techniques and based on the use of Mel- spectra and energy-based features. This study was framed in a wider context (the LaMIT project), which aims to apply Stevens' Lexical Access Model [2] to the Italian language.

Deep learning techniques were applied to automatic detection of foreign accents, an important element for improving automatic speech recognition (ASR), e-learning platforms, and speaker identification (SI) systems. Foreign accent detection presents challenges due to the influence of the speaker's native language on their pronunciation. To address these challenges, this research introduces a Multi-Kernel Extreme Learning Machine (MK-ELM) architecture.

1.1 Motivation and significance

Although advances in Automatic Speech Recognition (ASR) and machine learning have introduced powerful techniques also for speech processing, these methods often overlook the importance of understanding the acoustic cues on which human listeners rely. In recent years, researchers have recognized the value of explicitly analyzing phonological contrasts, contextual variations, and speaker differences, that are vital for both human and machine speech processing. This gap in the field motivates the enhancement and modernization of the xkl software, which was once a revolutionary tool for detailed acoustic analysis, but has since become outdated and underutilized due to

the lack of updates and public accessibility. Another key motivation is the need for a more robust and user-friendly tool that can handle large-scale data with high precision. Although tools like Praat [7] and Wavesurfer [8] are widely used, they do not offer the same granularity in acoustic measurements, particularly in spectral analysis, as xkl does. The ability of xkl to compute high-quality spectrograms, to easily provide time-varying signal spectra, and perform precise formant frequency estimation, makes it invaluable for studies that require fine-tuned acoustic measurements. By updating xkl to run on modern platforms and enhancing its capabilities, such as adding the reassigned spectrogram and improving the user interface, this project aims to fill the current void in acoustic speech analysis software. In addition, there is a demand for tools that can bridge the gap between traditional linguistic analysis and modern machine learning approaches. Modernizing xkl will enable researchers to extract critical acoustic information from speech signals that could inform both acoustic-phonetic theories and the development of more robust ASR systems. This project also seeks to support studies on diverse speaker populations and dialects, which are increasingly recognized as essential for building speech technologies that work across different languages and communities. Finally, the potential of xkl to reveal previously undetectable acoustic phenomena, such as subtle formant variations in vowels or fine-grained acoustic cues, highlights the importance of making this tool more accessible to the broader research community.

The focus of the research also expands to the detection of landmarks, that led to the development of a hybrid deep learning model that combines CNN and RNN to automatically detect vowel landmarks, using features such as Mel spectrograms and energy-based measures to increase detection accuracy and efficiency. By improving the ability to detect key acoustic events, the project not only advances speech technology but also opens up new research possibilities in the study of patterns of systematic phonetic variation and speech production across different speaker demographics.

Finally, the work focuses on identifying foreign accents. As voice technologies become more integrated into daily life, accurately recognizing accented speech is crucial to creating inclusive and effective systems. Conventional Automatic Speech Recognition (ASR) systems, primarily designed for native speakers, often struggle to interpret accented speech accurately, resulting in reduced performance. This issue is becoming increasingly important as voice-activated systems, such as voice assistants, telecommunications, e-learning platforms, and banking services, are widely adopted. The limitations of ASR systems in processing non-native accents underscore the need to develop more robust methods that can handle speech from diverse linguistic backgrounds. Automatic Identification of foreign accents aims to address this gap by establishing a means to detect a speaker's accent, thus opening the door to developing ways in which ASR systems can accommodate the phonetic variations introduced by non-native speakers and improve accuracy for accented speech. Beyond speech recognition, the project has applications in fields like security, intelligence, and immigration, where identifying a speaker's native language through their accent can provide valuable insights for decision making, such as in immigration screening and border control. It also has significant potential in language learning and speech therapy. By identifying and analyzing the phonetic details of their foreign accent, language learners can improve their pronunciation by pinpointing deviations from native speech patterns. In speech therapy, it offers tailored interventions for non-native speakers. Furthermore, it contributes to linguistic research by providing insights into how accents are shaped by a speaker's native language and how non-native speakers modify their

speech. Ultimately, this research aims to create more inclusive speech technologies, ensuring that voice-based systems are accessible and effective for all users, regardless of their accent or native language.

1.2 Literature Review

Analyzing speech signals, particularly their temporal and spectral properties, plays a key role in speech science research. Studies in areas like automatic speech recognition (ASR) systems [9–12], speech disorders [13–15], language acquisition in children and second language learners [16], and sociolinguistics area [17] all rely on precise acoustic signal measurements. Given the dynamic nature of speech, accurately estimating the instantaneous spectrum of speech has been a longstanding challenge, with research ongoing for over 50 years. Consequently, the advancement of speech processing tools has played a crucial role in progressing this domain. Praat, a popular tool created by Paul Boersma and David Weenink, is frequently cited [7] for its compatibility with multiple platforms such as Windows, MacOS, and Linux. Wavesurfer [8] and VoiceSauce [18], the latter of which operates in Matlab, are also significant tools worth mentioning. Additionally, the xkl tool, created in the 1980s by Dennis Klatt at MIT, offers superior capabilities for extracting signal measurements [19, 20]. When xkl was first introduced in the 1980s, it revolutionized the field with its ability to represent linguistically relevant aspects of speech acoustics in detail. This innovation led to widespread adoption among researchers, who customized it for their local computing environments. Klatt's synthesis module, KlSyn, embedded in xkl, also had a significant impact. Approximately 30 years ago, it was ported to the C programming language [21] and has since been released in other languages such as Python [22] and JavaScript [23]. The integration of a C++ based framework for synthesizer of Dennis H. Klatt into Praat has enabled its accessibility on modern platforms as of today [7]. However, the broader xkl software, particularly its spectral analysis modules, did not follow the same trajectory. Its source code was never publicly released, limiting its distribution and use. The decline in xkl's use can be attributed to two factors. First, research priorities shifted from understanding the acoustic cues that human listeners rely on, to optimizing ASR system performance. This shift de-emphasized the detailed analysis of phonological contrasts and how these cues vary across speakers, contexts, and dialects. Second, the rise of machine learning (ML) in ASR demonstrated the ability to leverage detailed acoustic patterns without making this information transparent to researchers. This led to an explosion of ML-based ASR systems, further sidelining the importance of explicit cue analysis. Despite these trends, recent research has emphasized the importance of identifying the types of information that are most beneficial for speech recognition [24]. We believe that now is the opportune time to update xkl to function on modern operating systems. By enabling detailed acoustic analysis, this tool can provide insights into the systematic differences in phonological categories, their contextual realizations, and variations among speakers. These capabilities are crucial not only for developing ASR systems that function across different populations, but also for understanding how humans process speech. Updating xkl will provide researchers with a valuable resource that aligns with the growing recognition of the need for explicit acoustic cue analysis. Several recent studies have demonstrated the effectiveness of such tools in speech analysis [25–30]. By updating and modernizing xkl's source code and documentation for compatibility with current platforms, a newly accessible version of xkl

could pave the way for important advancements in understanding the critical acoustic properties relevant to both speakers and listeners. This thesis marks an initial move towards achieving this objective. In xkl, the procedure for submitting speech samples for analysis is simple. Users have the option to either directly capture speech using the software or upload audio files they have recorded beforehand. All available processing features can be accessed via a single interface, details of which will be outlined in section 2. One of xkl's main strengths is its ability to perform high-quality acoustic analysis. Specifically, it generates exceptionally detailed spectrograms and allows users to manipulate spectrum slices at any point along the time axis, making it an ideal tool for research that requires precise frequency-domain measurements. With its updated functionality, xkl opens new possibilities for identifying acoustic cues within speech signals. It also offers precise formant frequency measurements, which are particularly useful in addressing complex questions such as those involving rounding or nasalization, where small frequency variations can be significant [12]. Moreover, xkl is capable of calculating the Discrete Fourier Transform (DFT) for as many as 4096 points, enabling detailed spectrum analysis at standard speech sampling frequencies. This functionality allows for the identification of subtle acoustic features that other software tools may miss, especially when analyzing vowels, where accurate formant estimation is crucial. [12]. The User Interface of xkl, which facilitates both time-domain and frequency-domain analysis, makes it highly valuable for large-scale data processing. Researchers can quickly compute spectral slices, generate spectrograms, and measure acoustic parameters with fine resolution, all with a few clicks. For instance, xkl was instrumental in collecting data for a study on lexical gemination [31], which was described in [31] and further analyzed in [25, 26]. Additionally, it was used to investigate lexical and syntactic gemination in the LaMIT database [3], and to study the phenomenon of double bursts in Italian geminated stop consonants [32].

After careful consideration of the factors that inspired the development of xkl, the existing literature was thoroughly examined to identify gaps in knowledge and research concerning the analysis of acoustic cues. In this thesis, xkl's superior qualities for formant estimation were utilized to validate and compare the results of automatic vowel landmark detection. Manual inspections using xkl allowed us to assess and enhance the reliability of our proposed detection framework. The foundation of modern speech recognition and processing lies in acoustic landmark theory. The Lexical Access Model proposed by Stevens [2] is one of the most significant contributions to this field, focusing on how listeners derive intended words from speech signals by identifying acoustic landmarks, distinct points in the speech signal that correspond to phonologically-significant articulatory events. The model emphasizes the extraction of distinctive features from acoustic cues and compares these with a stored lexicon, allowing listeners to hypothesize the word sequence. This model was pivotal in enhancing the precision of speech recognition systems, especially in identifying key phonetic events such as vowels and consonants [33]. Subsequent research built on Stevens' model by integrating it into automatic speech recognition (ASR) systems. Early approaches to landmark-based ASR showed promise, especially in handling noisy environments [34]. However, most research efforts focused on English, leaving a gap in understanding how the Lexical Access Model and landmark detection methods could be applied to other languages like Italian, which presents unique phonological challenges, such as gemination (the doubling of consonants). In addition, the rise of deep learning models has transformed the field of speech recognition. Researchers such as Wang et al. [35] showed that CNN models significantly improved ASR systems by learning both local

spectral characteristics and temporal dependencies. However, these models were not specifically designed for the detection of acoustic landmarks. Most studies focused on general ASR tasks, leaving a gap in research specifically targeting the automatic detection of acoustic landmarks in non-English languages, particularly in Italian phonology.

Subsequently, we investigate existing research concerning the Automated Identification of Foreign Accents (AIFA). Tasks designed to recognize foreign accents concentrate on identifying the primary language of people who communicate in a non-native secondary language. [36]. Studies in the area of accent recognition have garnered considerable interest among speech experts, as accents negatively affect the performance of traditional Speech Recognition systems [37]. These systems are typically created for native language speakers, and their effectiveness can be significantly reduced when processing other accented speech [38]. Variations in foreign accents negatively impact automatic speaker and language identification systems [39, 40]. Moreover, the identification of foreign accents proves to be exceptionally useful in intelligence and security applications. AIFA can also be used for voice transformation, soft biometric verification, telephonic services, e-learning, voicemail, voice authentication, and online banking. [41]. Understanding that accent differences can arise from both regional and dialectal distinctions is essential. Regional accents contribute in particular to variations in pronunciation and speech patterns [42, 43], particularly among those who are native speakers of the language. In addition, dialects introduce additional complexity to linguistic identification through variations in vocabulary and grammar. Studies reveal that foreign accents can embody regional or dialectal characteristics, deviating from conventional patterns of language in the articulation of consonants and vowels, rhythm, prosody, and overall speaking style.[44–47]. Foreign accents, unlike local dialects, are different from the original style or standards due to the impact of the speaker's native language on their ability to speak a non-native language [48]. When speakers have different accents, Manner in which individuals communicate may vary significantly depending on their native language. and their skill level in the second language, thus increasing the complexity of the matter. Speakers, while speaking a second language, often modify or substitute certain phonetic elements, as they may not fully adopt its speaking pattern or sound. For example, native Arabic speakers generally have difficulty pronouncing bilabial sounds like /b/ and its unvoiced counterpart, /p/. Similarly, there is a tendency to confuse alveolar sounds, such as /d/ and /t/, when speaking English [49]. Furthermore, speakers of second language commonly replace unfamiliar L2 phonemes with the closest equivalent of their native language sound system [50]. The strength of a foreign accent can differ between speakers of the same L1 based on their fluency in L2 [41, 51]. This research focuses on the Identification of Foreign Accents, a prominent area within speech processing that has gained significant attention due to its importance in the field of linguistics, language learning, and other different systems, including ASR system. This review seeks to explore the model architectures utilized in AIFA, following their progression from machine-based learning (ML) methods to the latest advances in deep learning (DL) approaches.

Early research in AIFA used mainly classic machine learning methods, including Linear Discriminant Analysis (LDA) [52], Hidden Markov Models (HMM) [53], and Gaussian Mixture Models (GMM) [54]. These methodologies were chosen because of their reliability in managing sequential data and their efficiency in classification tasks. Due to their ability to represent chronological patterns in speech signals, hidden Markov models played a crucial role in AIFA research. In the context of detecting accents, prosodic features, such as rhythm, pitch, and intonation, are crucial.

For instance, Kat and Fung [53] integrated these prosodic features with Hidden Markov Models (HMMs) to detect accents in English spoken by native Cantonese speakers. Based on related studies, Kumpf and King [55] used phonotactic characteristics combined with HMM to distinguish Arabic vs. English accents. Hansen et al. [36] improved this technique by including prosodic characteristics obtained from source generators, allowing differentiation of German, Turkish, and Chinese accents. This demonstrated the adaptability of Hidden Markov Models to diverse linguistic scenarios. Gaussian Mixture Models were chosen for their probabilistic approach, which models the distribution of speech traits by assuming that the data is generated from a mixture of multiple Gaussian distributions. Each distribution represents a different characteristic of the speech data, and probability is used to determine the likelihood of a data point belonging to each distribution. Phapatanaburi et al. [54] used GMMs with MFCC to detect English spoken by Japanese individuals, which often features distinctive accents. Fohr and Illina [56] used prosodic characteristics such as energy and pitch with GMM to differentiate accents between the French, Greek, and Italian language while speaking English. GMMs leverage a probabilistic approach to efficiently model the variations in speech signals. Choueiter et al. [52] integrated Maximum Mutual Information (MMI) and LDA for accent classification within the FAE dataset, implementing PLP-based features. Their method highlighted that LDA could improve the ability of AIFA systems to distinguish features effectively. The researchers tackled the constraints of traditional models by investigating the use of SVM for AIFA. Support Vector Machines have demonstrated outstanding effectiveness in managing high-dimensional feature spaces and are proficient in conducting binary and multiclass classifications. Kashif et al. [49] utilized the MFCC features in combination with an SVM to distinguish between the accents of native Arabic speakers when speaking English, thereby highlighting the robust discriminative potency of the SVM models in identifying accents. Bahari et al. [57] expanded this method to encompass various languages, using a multiple-dimensional feature vector that incorporates energy to distinguish Cantonese, Hindi language, Thai language, Russian, and Vietnamese accents. This research highlighted the scalability of SVMs for multiclass AIFA tasks. The representation of the i-vector model has emerged as an essential technique to identify both speakers and accents or dialects because of its concise and distinguishing characteristics. Behravan et al. [58] employed the i-vector model to distinguish seven distinct accents in which English serves as a non-native or second language (L2). This method marks a notable progress in Automatic Identification of Foreign Accents (AIFA), providing an efficient and robust depiction of features of speech used for recognizing accents.

Recent progress in deep learning (DL) has profoundly transformed AIFA. Architectures like DBN, ANN, RNN, CNN, LSTM, and DNN have achieved significant performance improvements. These architectures are utilized with high-dimensional datasets and sophisticated features to identify complex patterns in speech data. Sheng and Edmund [59] utilized the MFCC and CNN features to differentiate between Korean and Chinese accents when they spoke English. Complex models like CNNs have proven to be highly accurate in areas of dialect or accent recognition. Jiao et al. [60] merged RNN and DNN to create an AIFA model, leveraging the advantages of both architectures for managing sequential data. This comprehensive approach demonstrates the benefits of implementing DL models to enhance performance. Purwar et al. [61] combined LSTM with CNN to address the multiclass accent classification problem, encompassing many native languages such as Hindi, French, Arabic, Dutch, Korean, Spanish, Mandarin, Russian and Portuguese. Their re-

search highlights the effective combination of different features and modeling techniques for AIFA. Upadhyay and Lui [62] combined DBNs with MFCC features to detect accents in speakers from six distinct native languages, demonstrating the benefits of hierarchical feature learning in DBNs for AIFA applications.

In addition, AIFA methods have been successfully adapted to identify regional dialect within a language, demonstrating their versatility. Chen et al. [63] utilized Gaussian Mixture Models (GMMs) in combination with prosodic features to distinguish between various regional accents of Chinese Mandarin. Rizwan et al. [64] applied the ELM plus the MFCC attributes to recognize the regional accents of the United States (RA). [65] conducted a study on Deep Neural Networks (DNNs) and Support Vector Machines (SVM) with a focus on identifying Mandarin regional accents, utilizing i-vectors. These investigations highlight the flexibility of AIFA methods in tackling the categorization of various dialects and local accents. Recent studies have expanded the variety of attributes used in AIFA to include speech prosody attributes, spectrograms, mel-based spectrograms, fundamental frequency, and zero-crossing features. These features have been used in various datasets to improve the robustness and accuracy of automatic accent identification systems [66–72]. Incorporating diverse attributes encompasses a range of acoustic properties, thereby enhancing the model's effectiveness. The progression of AIFA methodologies from traditional ML-based algorithms to complex DL-based architectures highlights ongoing improvements in accuracy and robustness. The distinctive strengths and particular uses of each approach have collectively enhanced the understanding and automated detection of accents. As the field evolves, the integration of a variety of models and feature sets, combined with the use of extensive and diverse datasets, is expected to drive further advancements in AIFA. This, in turn, will improve the efficiency of automated systems in accurately identifying and analyzing foreign accents. This review highlights major advancements in AIFA research and lays the groundwork for future innovations and investigations.

Recently, the Extreme Learning Machine (ELM) and Multi Kernel Extreme Learning Machine (MK-ELM) models have achieved notable success in various classification domains. Specifically, Zhang and colleagues [73] proposed Multi Kernels based approach for EEG data classification in brain-computer interfaces (BCI). In a similar vein, Zhao and Guo [74] proposed an MK-ELM model aimed at predicting intervals for energy systems and grids. Ahuja and Vishwakarma [75] employed the Kernal-Based Model along ELM for classifying patterns. Multi-Kernel ELM models have shown promising results in multiclass classification; however, to the best of our knowledge, they have yet to be applied to Automatic Identification of Foreign Accents (AIFA).

1.3 Aims and objectives

To address the challenges outlined in the previous sections, specifically concerning advances in xkl, this study has two primary objectives. Firstly, it intends to transition the graphical user interface of the xkl software from the obsolete Open Motif libraries to contemporary UI libraries like GTK. This transition will involve a complete redesign of the GUI components to ensure that they are compatible and user-friendly on modern operating systems. Secondly, the study aims to incorporate a new algorithm for the Reassigned Spectrogram method to improve the precision of formant estimation. Moreover, by introducing features for formant saving that were previously

missing in xkl, the study aims to enhance its analytical functionality and make it a more effective tool for researchers.

Concerning research on the automatic landmark detection, we implement and assess automatic landmark detection methods using deep learning techniques. Due to advances in deep learning, these methods are expected to provide better accuracy and efficiency compared to traditional approaches. The study will investigate different neural network architectures, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), to identify the most effective model for formant detection. The research in automatic identification of foreign accents has two primary objectives. The first is to identify and propose an improved version of the Extreme Learning Machine Model based on multiple kernels called MK-ELM for the automatic identification of foreign accents (AIFA). An additional part of this goal is to assess its effectiveness in comparison with leading cutting-edge models. Secondly, rather than utilizing a conventional multi-classification model, we propose implementing a novel weighted scheme classification (WSC) approach to address the multi-classification challenges in accents, especially when handling high-dimensional AIFA data. This architectural innovation seeks to improve the model's accuracy and efficacy, particularly in addressing multi-classification challenges. Furthermore, this study will examine how the proposed MK-ELM-based AIFA model overcomes typical computational limitations, shortens training duration, and improves stability, which are recurrent challenges in traditional classification methods.

1.4 Thesis Contributions

This thesis, along with the related published article, makes the following contributions.

- In relation to the xkl project, there are three significant contributions. Firstly, a modern user interface for the xkl tool was created using GTK, which enhanced its compatibility with current computing platforms. Secondly, the advanced reassigned spectrogram algorithm was incorporated into xkl, improving its capabilities for spectrum analysis. Lastly, an extra module was added to xkl, allowing automatic saving of formants into text files, which aids in the analysis of large datasets. The updated version of xkl offers a comprehensive platform for communities focused on speech to conduct in-depth acoustic analysis, making it an excellent tool.

- This research enhances vowel landmark detection accuracy and computational efficiency by introducing a CNN-RNN hybrid model and incorporating energy and mel-spectrogram parameters, using the LaMIT Italian speech dataset.

- For Automatic Identification of Foreign Accents (AIFA), this study enhances the field by introducing a model based on Multi-Kernel Extreme Learning Machine (MK-ELM). Additionally, a pairwise weighted strategy is employed to tackle the multi-classification issues inherent in high-dimensional foreign accent datasets. This approach addresses computational challenges concerning training duration and model robustness, surpassing conventional methods like SVM, ANN, LSTM, ELM, MLELM, and KELM.

1.5 Outline of the Thesis

This thesis is structured into four chapters. Each chapter builds on the previous one, creating a cohesive narrative that guides the reader through the research journey.

The first chapter sets the stage by introducing the significance of speech signal processing and its applications in various fields, such as automatic speech recognition (ASR), speech synthesis, and clinical diagnostics. It provides a thorough background and motivation for the research, highlighting key research questions and objectives. This chapter also includes a detailed literature review that examines existing research, theories, methodologies, and technologies relevant to the thesis. It identifies gaps in current knowledge and discusses the limitations of existing approaches, setting the context for the innovative solutions proposed in the thesis. Finally, it highlights the contributions of the thesis and provides an overview of the thesis organization, guiding the reader through the subsequent chapters.

Chapter two centers on updating the xkl tool for acoustic analysis, which had become obsolete. It outlines the initiatives taken to modernize the tool, such as incorporating new algorithms and updating to the latest user interface libraries. These developments are intended to improve the tool's effectiveness in contemporary speech research by overcoming past limitations and boosting its compatibility with modern operating systems.

Chapter three delves into the automatic detection of vowel landmarks through a hybrid CNN-RNN model, while also addressing the identification of foreign accents in speech recognition systems. It details the creation and utilization of an MK-ELM model accompanied by a weighted scheme, demonstrating their efficacy in enhancing ASR performance for speech with accents. Comprehensive results and analyses are presented, underscoring the advantages of the MK-ELM model compared to traditional classification methods.

The concluding chapter encapsulates the main discoveries of the study, considering the field's advancements in speech signal processing. It proposes avenues for forthcoming research and highlights the crucial role of ongoing innovation in acoustic analysis methods. This chapter connects the insights from earlier chapters and explores their practical impacts on the progress of speech technology.

Chapter 2

Enhancements in xkl: Design and development of tools for advanced acoustic analysis

The xkl software has been an essential tool for acoustic analysis for a long time, but earlier versions faced challenges related to usability, compatibility, and efficiency, which limited its effectiveness for extensive speech analysis and broader applications. This chapter introduces major updates that rectify these issues to enhance the software's usability, compatibility, and overall analytical capabilities. A notable improvement is the creation of a modern graphical user interface using GTK libraries, which addresses installation problems associated with the outdated Open Motif libraries and offers a more user-friendly experience. Additionally, the inclusion of a reassigned spectrogram algorithm enhances the clarity of time-frequency speech signal representations, resulting in a richer analysisnvital for speech science research. Another significant enhancement is the formant-saving module, which automates the extraction of formants from speech recordings by processing TextGrid files and exporting the results into text format, thus minimizing manual effort and enabling large-scale data processing. These updates not only streamline the workflow but also boost the xkl software's effectiveness as a comprehensive tool for acoustic analysis. Collectively, these enhancements transform the updated xkl software into a robust and user-friendly solution that caters to the evolving needs of speech researchers and practitioners.

2.1 xkl Software Architecture

The xkl software provides an integrated platform that unifies several tools for speech signal creation, analysis, and synthesis. Initially developed by Dennis Klatt as individual command-line applications, these tools can now be accessed via a Graphical User Interface (GUI) utilizing Motif's open-source libraries [76]. The most recent version, xkl v3.2, continues to be coded in C, maintaining the original programming language, and is now compatible with Windows, Linux, and MacOS operating systems [16]. Significant modifications were made to the original code to ensure compatibility with the latest gcc and clang compilers and the newest OpenMotif library. The xkl software is available to the research community under the GPL 2.0 open source license, with possible exceptions upon request. Additional information can be found in [77].

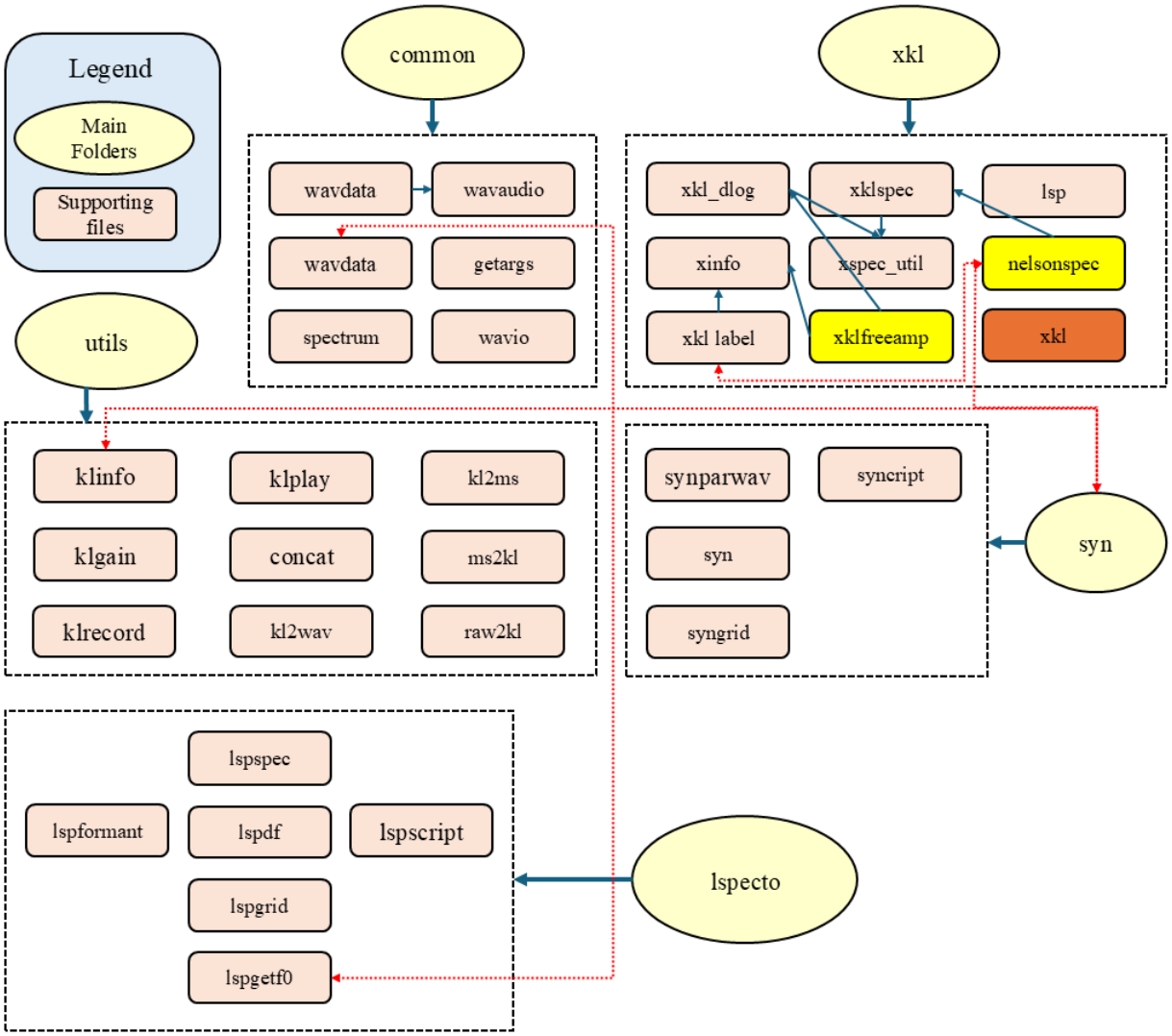


Figure 2.1: Overall architecture of xkl, outlining folders and supporting. It is divided into a number of primary folders: (**Common**, **xkl**, **Utils**, and **syn**), which make up various modules and files interacting to provide the functionality of the tool.

The xkl source code is organized into multiple C files distributed across five directories.

- **utils**: This directory includes utilities for converting file formats to and from the Klatt (.wav) format utilized by the software.
- **common**: This directory holds files for playing, recording, and handling waveform (.wav) files.
- **syn**: Dedicated to files associated with the Klatt synthesizer, KlSyn.
- **lspecto**: Contains files related to the Klatt spectrogram analysis tool, lspecto.

- **xkl**: This directory encompasses files tied to the GUI and how it integrates with functions from files in other directories.

Figure 2.1 illustrates the structure of the xkl source code.

The diagram presented in Figure 2.1 displays C files situated within each directory, depicted as orange rectangles. Dependencies are denoted by arrows, where an arrow from file A to file B signifies that A utilizes functions declared in B. As shown in Figure 2.1, the software includes executables in every directory except for the common folder. Within the **utils** folder, the executables offer utilities for handling (.wav) files, allowing users to extract data, play audio, and convert between multiple formats. The **synmain** executable in the **syn** directory serves as a command-line version of the KlSyn synthesizer, operable without the graphical interface; this variant corresponds to version 2.1 of KlSyn93. Similarly, the **lsp** executable acts as a command-line form of the lspecto tool, intended for calculating and visualizing spectrograms of waveform data in (.wav) files. The functionalities provided by the command-line tools *synmain* and **lsp** are fully integrated into the **xkl** executable, located within the **xkl** directory, which is centrally examined in this work. Detailed explanations of the xkl functionalities are provided in the subsequent section.

2.2 Reassigned Spectrogram Theory

The reassigned spectrogram is an advanced technique designed to improve the time-frequency representation of signals. This method builds upon the traditional Short-Time Fourier Transform (STFT) by utilizing the phase information of the STFT to relocate energy to more precise time-frequency coordinates. The key theoretical foundation of the reassigned spectrogram involves calculating the instantaneous frequency and group delay, which provides a clearer depiction of the spectral components of the signal. The process of reassignment involves computing the partial derivatives of the STFT phase with respect to time and frequency. These derivatives are used to reassign each energy point in the spectrogram to its correct location in the time-frequency plane. This method enhances the resolution and clarity of the spectrogram, allowing for a more accurate visualization of the signal's spectral content.

Mathematically, the STFT of a signal $f(t)$ with a window function $g(t)$ is defined as [78]:

$$\mathrm{STFT}_g(\omega, \tau) = \int_{-\infty}^{\infty} f(t+\tau) g(-t) e^{i\omega t}\, dt \tag{2.1}$$

The reassignment is achieved by using the phase derivatives of the STFT. The instantaneous frequency (IF) and local group delay (LGD) are given by [78]:

$$\mathrm{IF}(\omega, \tau) = \frac{\partial}{\partial \tau} \arg(\mathrm{STFT}_g(\omega, \tau)) \tag{2.2}$$

$$\mathrm{LGD}(\omega, \tau) = -\frac{\partial}{\partial \omega} \arg(\mathrm{STFT}_g(\omega, \tau)) \tag{2.3}$$

These values are used to relocate the energy points in the spectrogram, providing a sharper and more precise representation. The implementation of the reassigned spectrogram involves several computational steps that enhance its time-frequency resolution. Initially, the process begins with computing the Short-Time Fourier Transform (STFT) of the signal using an appropriate window

function, which provides the initial time-frequency representation of the signal. Following this, the partial derivatives of the STFT phase are calculated with respect to both time and frequency, which are critical for determining the instantaneous frequency and local group delay of the signal. Utilizing these derivatives, each energy point in the spectrogram is then reassigned to its correct location, significantly enhancing resolution by relocating smeared energy to more precise time-frequency coordinates. This reassignment provides a clearer representation of transient events and closely spaced spectral features. To ensure that the resulting spectrogram highlights the most pertinent details of the signal, pruning techniques are applied to isolate relevant components such as formants, while ignoring noise and artifacts. This involves setting thresholds for the second-order mixed partial derivatives of the STFT phase to distinguish significant components from background noise. By remapping all spectrographic magnitudes to the instantaneous frequency of the nearest signal component and correcting time smearing by reassigning to the nearest times where the group delay is concentrated, the reassigned spectrogram offers a highly detailed and accurate time-frequency representation, effectively addressing the limitations of traditional spectrograms and providing enhanced resolution and clarity essential for detailed speech analysis.

The reassigned spectrogram has numerous applications in speech analysis due to its enhanced resolution and clarity. The improved resolution of reassigned spectrograms allows for more accurate identification and tracking of formants, which are crucial for understanding the resonant frequencies of the vocal tract. Additionally, the technique provides a clearer depiction of harmonics and other spectral features, facilitating detailed analysis of the harmonic structure of speech sounds. Reassigned spectrograms excel at detecting transient events in speech, such as plosive sounds and rapid changes in pitch, which are often blurred in traditional spectrograms. It may be possible for clinicians to use reassigned spectrograms to diagnose and analyze speech disorders more effectively by examining detailed spectral features and their variations. Several studies have validated the effectiveness of reassigned spectrograms compared to traditional methods. Comparative studies have shown that reassigned spectrograms offer significantly higher resolution and clarity than traditional spectrograms. This improvement is particularly evident in the precise localization of spectral features and the reduced smearing of transient events. The reassigned spectrogram's ability to provide clear and precise spectral representations makes it superior for detailed acoustic analysis. Furthermore, reassigned spectrograms have been shown to maintain their effectiveness in noisy environments, where traditional spectrograms often struggle. This robustness is crucial for real-world applications where speech signals are frequently contaminated by background noise. In conclusion, the reassigned spectrogram represents a significant advancement in the visualization and analysis of speech signals. The integration of reassigned spectrograms into modern speech analysis tools holds great promise for advancing research and applications in the field of speech science.

2.3 xkl Interface and Functionalities

The xkl application sets up the graphical user interface and displays five dropdown menus, each providing access to a distinct set of functions. Figure 2.2 illustrates the interface of xkl.

The software consists of four principal menus: File, Time, Spectrum, and Audio. The File menu primarily manages reading and writing speech recordings. Initially, xkl utilized a special

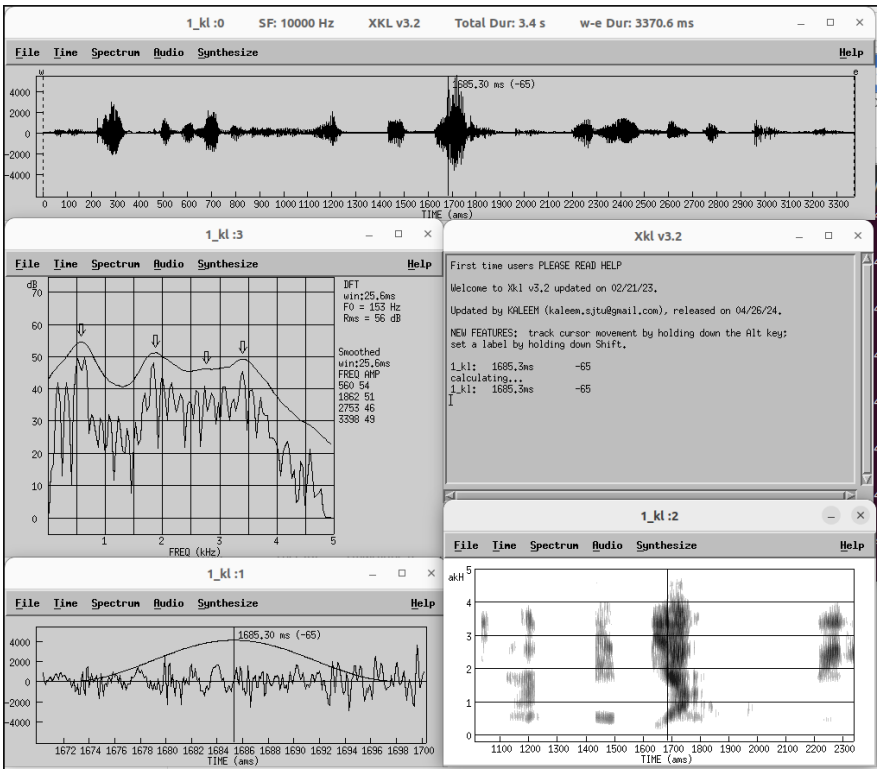


Figure 2.2: xkl windows correlated with a speech signal and a shared feedback window. Each speech signal window is identified by a suffix in the window header, where the suffix ranges from 0 to 3. Window 0 displays the signal waveform; Window 1 provides a magnified view of the waveform around the cursor position and shows the window shape for selecting a segment for spectral analysis; Window 2 presents the signal spectrogram; Window 3 offers a spectrum slice.

format called Klatt wave (.wav) for speech file encoding and decoding. This format had a unique header followed by signal samples formatted as 16-bit unsigned integers. The software's latest version still supports this format but has also added compatibility for the more common Microsoft (.wav) format, usable with most recording tools via conversion utilities in the *utils* folder. Beyond speech files, xkl can also process text label files, postscript files for printing time and frequency analysis plots, and ASCII text files for importing and exporting time and frequency data related to a waveform. The Time Analysis menu includes tools for graphically displaying and modifying the signal waveform, such as shifting time, zooming, and setting cursors, along with time-domain analysis for identifying peaks and valleys. The Frequency Analysis menu provides tools for signal representation and evaluation in the frequency domain, featuring the Digital Fourier Transform (DFT) with up to 4096-point resolution, smoothed and Linear Prediction Coding (LPC) spectra, and spectrogram generation. Analysis functions also include specific speech parameter functions, like peak detection, formant estimation, and critical band calculation.

2.4 xkl User Interface Limitations

The Motif-based user interface of the xkl software, despite being groundbreaking when first introduced, faced numerous significant drawbacks that impaired xkl usability and functionality as time progressed. One key issue was its outdated design, missing the modern features and intuitive layout that users have come to expect from current software interfaces. This resulted in challenging

navigation, particularly for new users, and contributed to a steep learning curve. Furthermore, Motif libraries experienced compatibility problems with modern operating systems, making the installation and maintenance processes more complex [79]. Users often found the software setup difficult, as the Motif-based interface required substantial modifications to operate correctly on newer systems. In addition, the limited widget set and the lack of modern accessibility options in Motif further diminished the user experience, making it less user-friendly for individuals with disabilities. Together, these limitations decreased the efficiency and accessibility of xkl [80].

2.5 Introduction of GTK-based UI

Switching from a Motif-based UI to a GTK-based UI offers various benefits, addressing the limitations of Motif and improving usability and compatibility. The open source nature of GTK allows developers full access to its source code, enabling extensive customization and fostering a collaborative community that continuously improves the toolkit [81]. This results in frequent updates, bug fixes, and enhancements, making GTK a robust and evolving platform. Developers also benefit from a vast array of community-created plugins and extensions, accelerating development and expanding functionality. The modern design principles of GTK emphasize simplicity and ease of use, significantly enhancing user experience with a comprehensive set of widgets and controls [82]. Applications built with GTK maintain a consistent look and feel across platforms, reducing the learning curve and boosting productivity. Furthermore, GTK supports accessibility features and adheres to modern standards, ensuring that applications are usable by people with various disabilities and comply with regulations such as ADA (Americans with Disabilities Act) and WCAG (Web Content Accessibility Guidelines) are important standards that GTK adheres to to provide accessible applications for users with disabilities [83]. GTK's cross-platform capability ensures that xkl software can be easily installed and used on Windows, macOS, and Linux without extensive modifications, providing consistent performance across different platforms. The active development community ensures that applications stay up-to-date with the latest UI advancements, supported by extensive documentation and tutorials. Technically, GTK applications generally offer better performance and resource management compared to Motif, with more efficient use of system resources leading to faster and more responsive applications. The integration with Cairo provides advanced graphics capabilities for high-quality text and graphic rendering, essential for detailed visualizations. GTK also supports smooth animations and transitions, enhancing visual appeal and ensuring applications scale effectively across different screen sizes and resolutions. Installing and maintaining GTK-based applications is more straightforward, with minimal dependencies and readily available binaries and packages for all major platforms. On Linux, robust package management systems like apt and yum further simplify deployment and maintenance. The consistent and stable API (Application Programming Interface), which is a set of tools and protocols that allows software components to communicate with each other, reduces the need for frequent code changes in GTK. This ensures that long-term maintenance is manageable and that applications remain compatible even as the toolkit evolves. This transition from a Motif-based UI to a GTK-based UI significantly benefits xkl software by addressing the limitations of the older interface. The enhanced design, usability, development efficiency, and technical capabilities, along with simplified installation and maintenance, ensure that the software remains accessible, efficient, and user-friendly, ultimately

improving the overall functionality and user experience of xkl.

2.6 Advancing xkl: Moving from Motif to GTK UI

Creating the new xkl GUI with GTK encompassed various crucial stages, each designed to ensure the software remained robust, user-friendly, and functional across diverse platforms. The subsections below describe the specific steps in the development process:

2.6.1 Key Libraries of GTK

The xkl GUI has been crafted utilizing the GTK software libraries (GIMP Toolkit), a choice motivated by the need for a robust, flexible, and cross-platform solution. GTK has traditionally been used in developing numerous Linux applications, its functionalities transcend the Linux environment. It supports all major operating systems, including Windows and MacOS, ensuring seamless performance of GTK-built applications across different platforms [84]. This cross-platform compatibility was a critical consideration in choosing GTK for the new xkl GUI, as it facilitates broader audience reach without the constraint of operating system restrictions. One of GTK's primary benefits is its extensive collection of widgets, ranging from basic buttons and labels to advanced tree views and text editors. This expansive widget library empowers developers to create highly interactive and visually appealing interfaces. For xkl, this translates to a new GUI with enhanced visualization tools, better layout options, and customizable features that align with the specific needs of speech analysis researchers. Furthermore, GTK is well-known for its robust community support and active development. As an open-source project, it benefits from ongoing contributions from developers worldwide, ensuring it remains current with the latest technological advancements [83].

For the xkl development team, this meant that creating a new, modern GUI could be achieved efficiently without compromising on quality or functionality. Specifically, the new xkl GUI makes use of several key GTK libraries, including GTK+ for creating graphical user interfaces, GDK (GIMP Drawing Kit) for windowing and graphics, GLib as the low-level core library that forms the basis for GTK+ and GNOME, Pango for layout and rendering of text, crucial for displaying spectrogram labels and annotations, and Cairo as a 2D graphics library used for drawing and rendering images and graphics in the GUI.

2.6.2 xkl Development Steps Using GTK

The process of creating the new GTK-based xkl GUI included several crucial steps, each designed to ensure that the software was reliable, easy to use, and compatible with various platforms. The following are the specific steps taken during development, also shown below in the flow chart 2.3.

Requirements Analysis

The first step in the development process was a thorough requirements analysis. This involved understanding the limitations of the original xkl software, collecting user feedback, and identifying the key features and improvements needed in the new GUI. The goal was to create a modern and

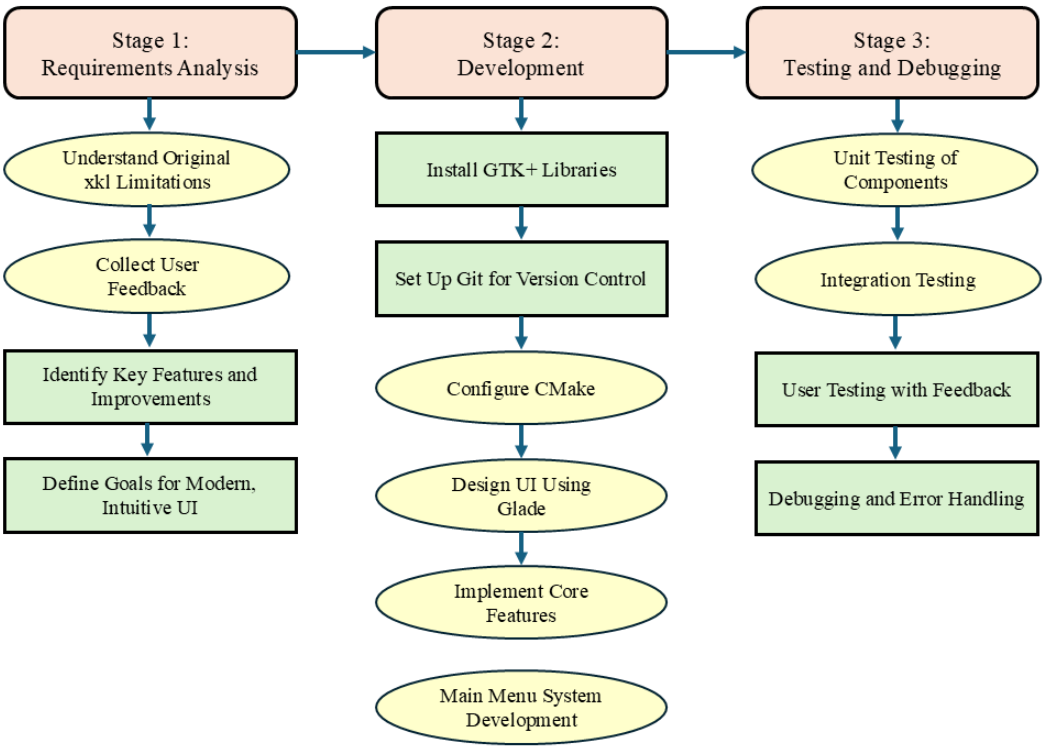


Figure 2.3: Development stages of the xkl modern UI system, showing the flow from requirements analysis, through development, to testing and debugging. Each stage is broken down into specific tasks necessary for successfully upgrading and modernizing the tool's interface and functionality.

intuitive interface that addressed usability issues and added new functionalities for enhanced speech analysis.

Development Environment Setup

The selection of the right development environment was crucial for the project. We chose to use the GTK libraries because of their flexibility, cross-platform compatibility, and extensive widget set. Development was carried out in a Windows environment, with tools and libraries that ensured the application could be seamlessly ported to other operating systems such as Linux and macOS. Setting up the development environment involved installing the necessary GTK libraries and tools. The following steps were taken:

- Installing GTK+ libraries: Downloading and installing the GTK development bundle for Windows from the official GTK website [85].
- Setting up a version control system using Git: Installing Git for Windows, which includes Git Bash and a graphical user interface for repository management.
- Configuring the build system: Downloading and installing CMake for Windows to manage the compilation process [86].

Design and Implementation

The design phase focused on creating an intuitive and user-friendly interface. Wireframes (basic blueprints outlining the structure and layout of a user interface) and prototypes were developed using tools like Glade, a user interface designer for GTK. The core features were implemented in iterative cycles, starting with basic functionalities and progressively adding more complex features. Key implementations included:

- Developing the main window and the menu system with the following menus:
  - **File Menu:** Options for reading and saving various file types, including (.wav) files, spectrum files, parameter files, and label files. It also includes options for editing and managing labels as well as for saving and opening PostScript files for spectra.
  - **Time Menu:** Tools for precise control of navigation and selection within the audio data, including moving markers, setting regions, zooming in and out, and adjusting the view of the waveform.
  - **Spectrum Menu:** A range of tools for spectral analysis, such as computing the DFT magnitude, smoothing spectra, critical-bands, and adjusting spectrogram and reassigned spectrogram parameters. Users can also include or exclude specific spectral features and recalculate spectrograms.
  - **Audio Menu:** Playback and recording functions, allowing users to play specific segments or entire audio files and to record new (.wav) files with customizable parameters.

Testing and Debugging

Extensive testing and debugging were critical to ensure that the software was reliable and user-friendly. The testing process was conducted in several stages to cover all aspects of the software's functionality and performance. Firstly, unit testing was used to evaluate the functionality of individual components. Each module, including file handling, waveform visualization, spectral analysis, and audio playback, was rigorously tested in isolation. This step ensured that each component performed its intended function correctly and efficiently, laying a solid foundation for future testing stages.

Following successful unit tests, integration testing was conducted to verify the seamless operation of the combined components. This stage involved testing interactions between the main window, the four primary menus (File, Time, Spectrum, and Audio), and their respective functionalities. The integration tests aimed to ensure that the modules worked together without conflicts, maintaining the integrity of the data flow and user interactions across the software. User testing was a pivotal part of the process, with target users selected from the LaMIT dataset [3]. This dataset includes recordings from four speakers, each providing two utterances (V1 and V2). The controlled selection ensured that the users were representative of the software's intended audience, enabling consistent and reliable performance evaluation. These users were tasked with performing typical operations within the software, such as loading and analyzing audio files, manipulating time markers, and conducting spectral analysis. Their feedback provided valuable insight into the usability and practical performance of the software. This stage identified usability issues and areas for improvement that may not have been apparent during the developer-centric testing phases.

Throughout the testing phases, debugging was an ongoing process. Any issues identified during unit, integration, or user testing were meticulously documented, analyzed, and resolved. The debugging process used automated debugging tools and manual code inspection to locate and fix bugs. Special attention was paid to edge cases and potential user errors to ensure robust error handling and a smooth user experience.

2.7 UI Development for Key Analytical Modules

The development of the GTK-based xkl GUI involved a comprehensive and iterative process aimed at creating a modern, user-friendly interface. We integrated all essential xkl modules to boost the software functionality and user-friendliness, transforming it into a robust tool for comprehensive acoustic analysis. To organize xkl's functionalities, we categorized it according to the available algorithms, which are:

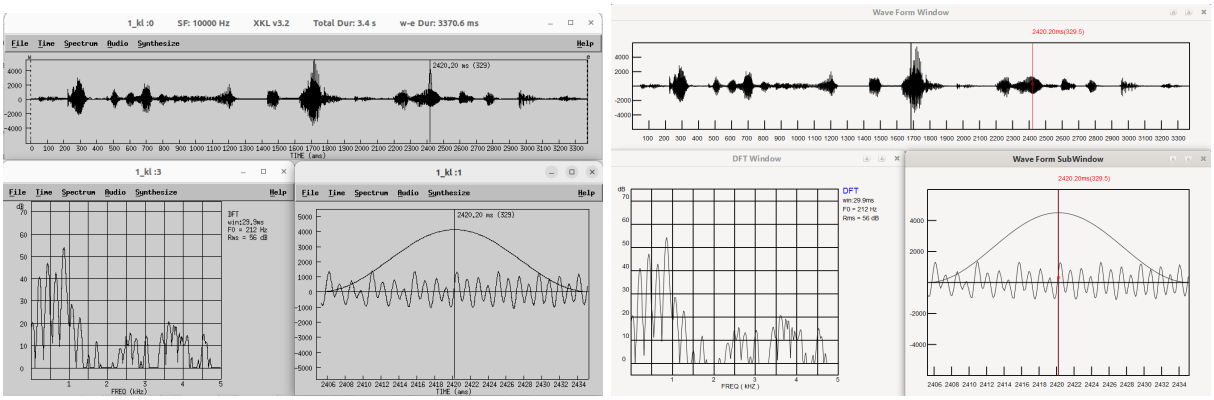
- **DFT Magnitude**
- **Critical Band**
- **Smoothed Spectrum**
- **Average Spectrum**
- **Linear Prediction**
- **Spectrogram**
- **Reassigned Spectrogram**
- **Spectrum Parameters**
- **Spectrogram Parameters Configuration**

In this section, we will describe the UI of each segment and compare the legacy motif-based xkl with our newly developed GTK-based UI for each module.

DFT Magnitude: Initially, the DFT magnitude module was added, enabling users to perform the DFT. This module calculates the signal's frequency spectrum, offering significant insights into its harmonic structure and highlighting key frequency components. As demonstrated in Figure 2.4 (a), the original DFT spectrum module based on the xkl motif, and Figure 2.4 (b) displays the xkl-developed GTK-based DFT module. The DFT spectrum is crucial for comprehending the spectral content of speech and various other acoustic signals.

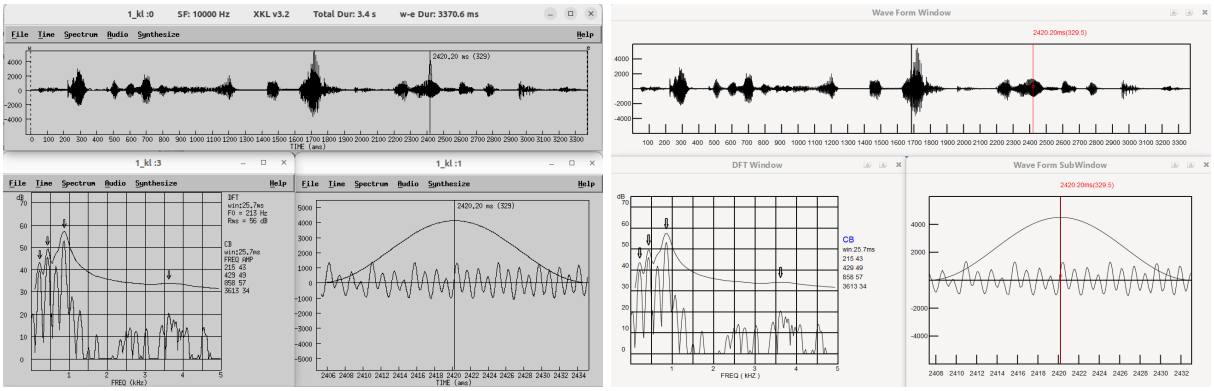
Critical Band: the Critical Band (CB) module was integrated. Users can customize the center frequencies for the first two CB filters and the transition frequencies from linear to logarithmic scales. As illustrated in Figure 2.5 (a), the unaltered xkl motif-based CB module is shown, and Figure 2.5(b) presents the newly developed GTK-based CB module of xkl.

Smoothed Spectrum: Subsequently, we integrated the smoothed-spectrum module. This module extends the fundamental DFT analysis by implementing smoothing methods in the frequency spectrum. These smoothing processes help reduce noise and highlight important spectral features, making data interpretation more straightforward. A low-pass filter is applied to the DFT



(a) (b)

Figure 2.4: Figure (a) presents the DFT spectrum analysis of a signal at a specific moment using the xkl v3.2 software, while figure (b) illustrates the DFT magnitude using GTK. The DFT window is 29.9 ms in size, with a fundamental frequency (F0) of 212 Hz and an RMS value of 56 dB. A particular time point is marked at 2420.20 ms, with a corresponding marker value of 329. These parameters are essential for examining the signal's frequency components at this precise time..

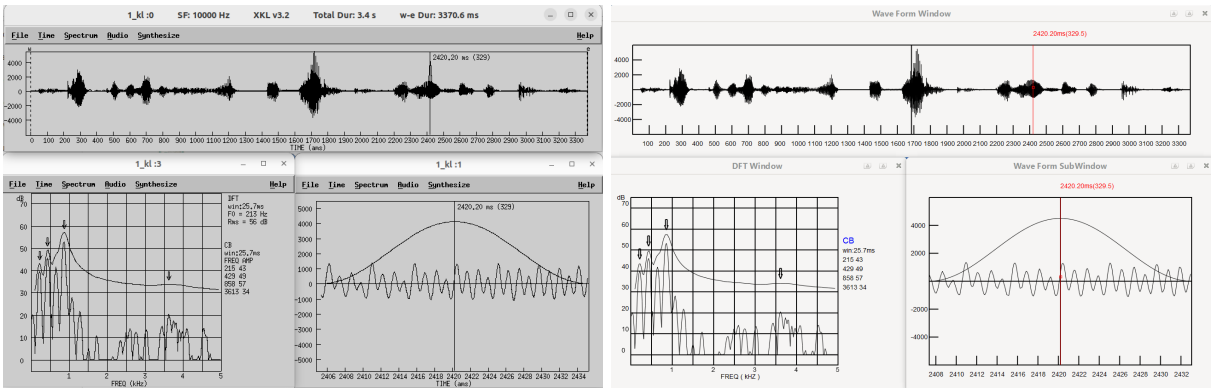


(a) (b)

Figure 2.5: Figure (a) presents the DFT spectrum analysis of a signal with CB at a particular instant, employing xkl based on motif, whereas figure (b) illustrates CB using the GTK-based xkl software. The window demonstrates the DFT spectrum analysis of a signal at a specific moment using the xkl v3.2 software, showing a DFT window size of 25.7 ms and formant frequencies of 215, 429, 858, and 3613 kHz with their respective amplitude values of 43, 49, 57, and 34 dB.

to generate the smoothed spectrum. This allows maxima to be seen more clearly. The bandwidth of the filter can be configured. This refined spectrum is especially beneficial for studying speech signals where accurate frequency resolution is essential, as illustrated in Figure 2.6. Here, Figure2.6 (a) depicts the smoothed spectrum based on the xkl motif, while Figure 2.6 (b) shows the smoothed spectrum developed using GTK.

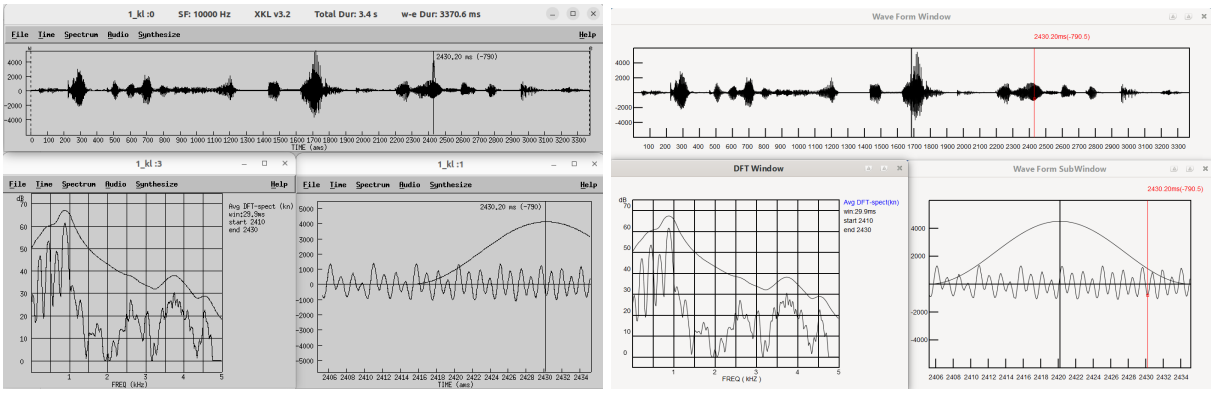
Average Spectrum: Next, we implemented a DFT average module. The DFT window, with a size of 29.9 ms, analyzes the frequency content of the signal at a specific time frame. The software averages the spectra over a time window of ±10 milliseconds from the current time instant (2430.20 ms), using 1ms time steps. This method captures the signal's frequency characteristics in short intervals, smoothing out variations to reveal the dominant frequencies within the selected window. The graph highlights peaks and troughs, where the most prominent frequency components lie between 1kHz and 5kHz. The spectrum displays the intensity (in dB) over a range of frequencies,



(a) (b)

Figure 2.6: Figure (a) presents the smoothed DFT spectrum analysis of a signal at a certain moment using the xkl v3.2 software, while Figure (b) shows a smoothed spectrum obtained via GTK. The DFT window is 25.6 ms. The fundamental frequency (F0) and its amplitude at this specific instant (2420.20 ms) are presented. The identified formant frequencies and their amplitude values are as follows: Frequency: 850 Hz, Amplitude: 54 dB; Frequency: 3652 Hz, Amplitude: 25 dB; Frequency: 4531 Hz, Amplitude: 18 dB.

showing how different harmonics contribute to the overall structure of the signal, providing insight into its spectral density. Here, Figure2.7 (a) depicts the smoothed spectrum based on the xkl motif, while Figure 2.7 (b) shows the smoothed spectrum developed using GTK.

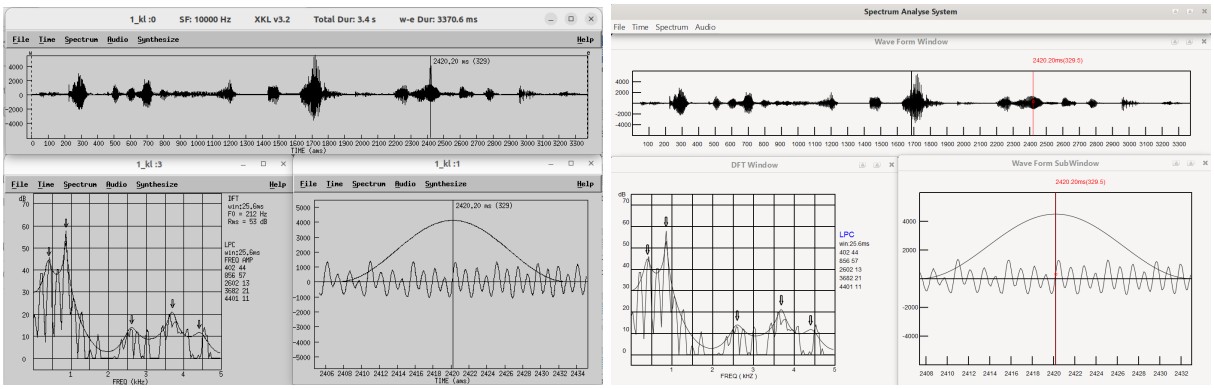


(a) (b)

Figure 2.7: Figure (a) displays spectra within a ±10 millisecond window centered at the current time (2430.20 ms), with 1 ms intervals, whereas figure (b) demonstrates the same using gtk.

Linear Prediction: Subsequently, we introduced the Linear Predictive Coding (LPC) module. LPC is a fundamental method in digital signal processing, especially for speech compression and synthesis. Future speech samples are predicted using previous samples, providing a compact representation of the speech signal. Our integrated module features a configurable number of coefficients, enabling users to adjust the analysis granularity to meet specific needs. LPC analysis identifies essential features of the speech waveform, including formants, which are the resonant frequencies of the vocal tract and critical to differentiating speech sounds. Using LPC, we can efficiently model these features, facilitating effective speech encoding, synthesis, and even speaker recognition. The window size used in the analysis (such as the 25.6-ms window shown in the spectrum plot) determines the portion of the signal analyzed at a time, balancing time and frequency resolution. By adjusting these parameters, the module can be tailored for various applications, such as speech

analysis in real time, voice synthesis, or other advanced audio processing tasks. Figure 2.8(a) shows the LPC based on the XKL motif, while Figure 2.8 (b) illustrates the LPC developed using GTK.



(a) (b)

Figure 2.8: Figure (a), based on the xkl motif, illustrates the frequency spectrum (in dB) obtained through LPC analysis. The frequency (KHz) is displayed on the x-axis, and the dB on the y-axis. Several formant peaks are highlighted with arrows, and the formant frequencies are listed to the right under the LPC section: 402 Hz, 856 Hz, 2602 Hz, 3682 Hz, and 4401 Hz, with corresponding amplitudes of 44 dB and 57 dB. The LPC algorithm uses a window size of 25.6 ms to compute these formant frequencies. The same parameters are used in Figure (b), which was developed using GTK.

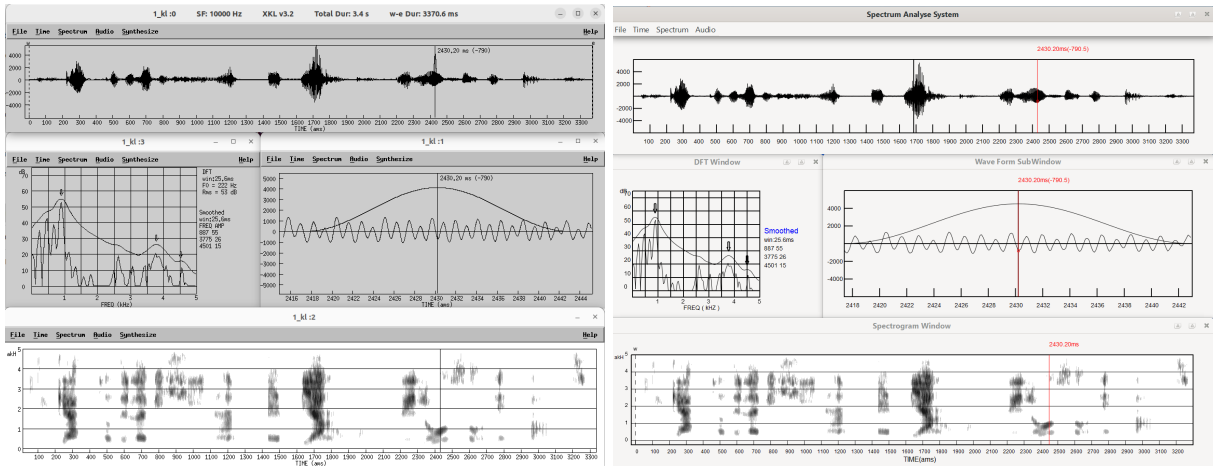
Spectrogram: Following this, the Spectrogram Development Module offers a detailed representation of the audio signal in the time-frequency domain, facilitating an in-depth analysis of its spectral elements. A spectrogram displays the intensity of frequencies over time, with dark areas signifying higher energy levels at specific frequencies. This module is crafted to provide flexibility in setting parameters like window size and overlap, allowing users to adjust the time and frequency resolution. With this tool, users can visually explore speech characteristics such as formants, harmonics, and transient events, making it indispensable for speech analysis, phonetics, and acoustic research. The spectrogram is an additional tool for LPC, offering a more intuitive and dynamic method for observing how frequency content changes over time, thus aiding users in extracting more detailed information from audio signals. Figure 2.9 (a) presents the Spectrogram based on the XKL motif, while Figure 2.9 (b) depicts the Spectrogram created with GTK.

Change Spectrogram Parameters: By default, the spectrogram is computed over 6.4 ms intervals using 128 sample FFTs, averaged in groups of 3. The user can change its parameters by initilizing its "Change Spectrum Parameters" from the Spectrum menu. Figure 2.10 (a) presents the Spectrogram based on the XKL motif, while Figure 2.10 (b) depicts the Spectrogram created with GTK.

Spectrum Configuration: Figure 2.11 illustrates the default spectrum parameter settings, which users can modify as needed. Figure 2.11 (a) depicts the parameter evolution in xkl using motif, while Figure 2.11 (b) displays the Spectrum parameter settings in GTK-based xkl.

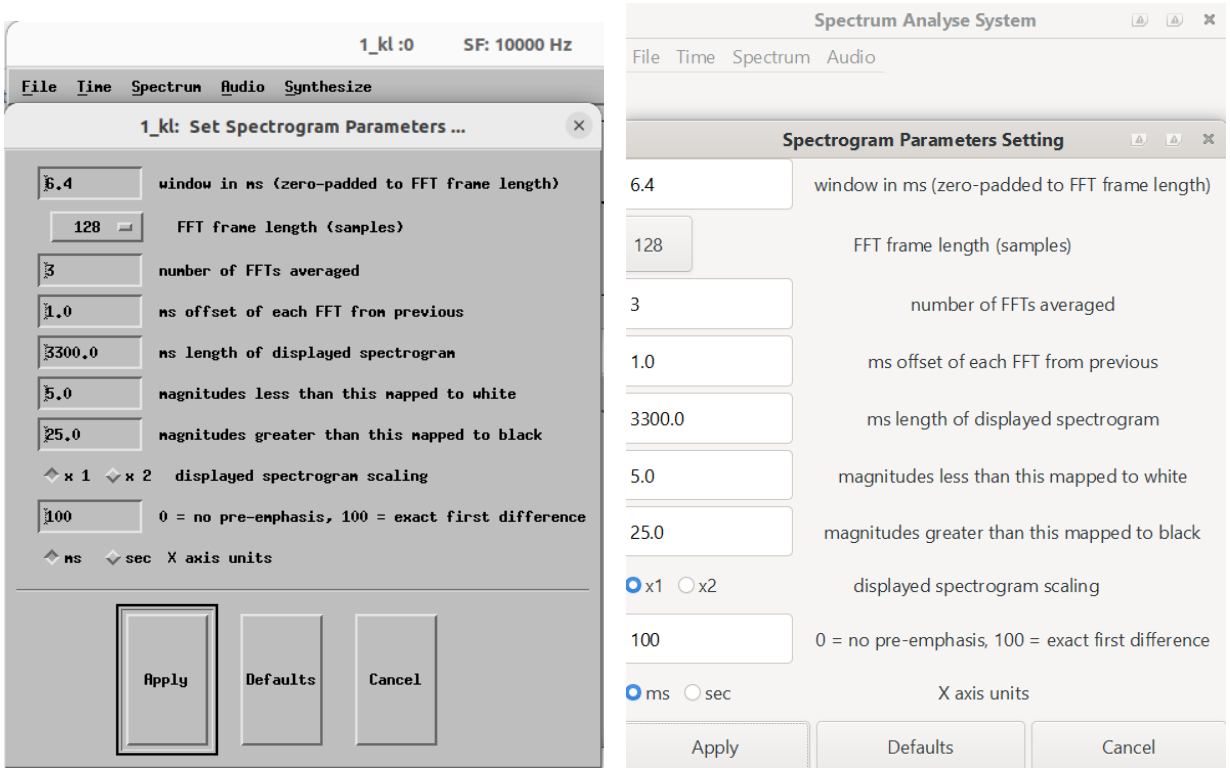
2.7.1 Reassigned Spectrogram Integration steps

To integrate Reassigned Spectrograms, advanced signal processing techniques were employed to enhance the time-frequency resolution of speech signals beyond what conventional spectrograms could achieve. Traditional spectrograms, generated using the Short-Time Fourier Transform (STFT), of-



(a) (b)

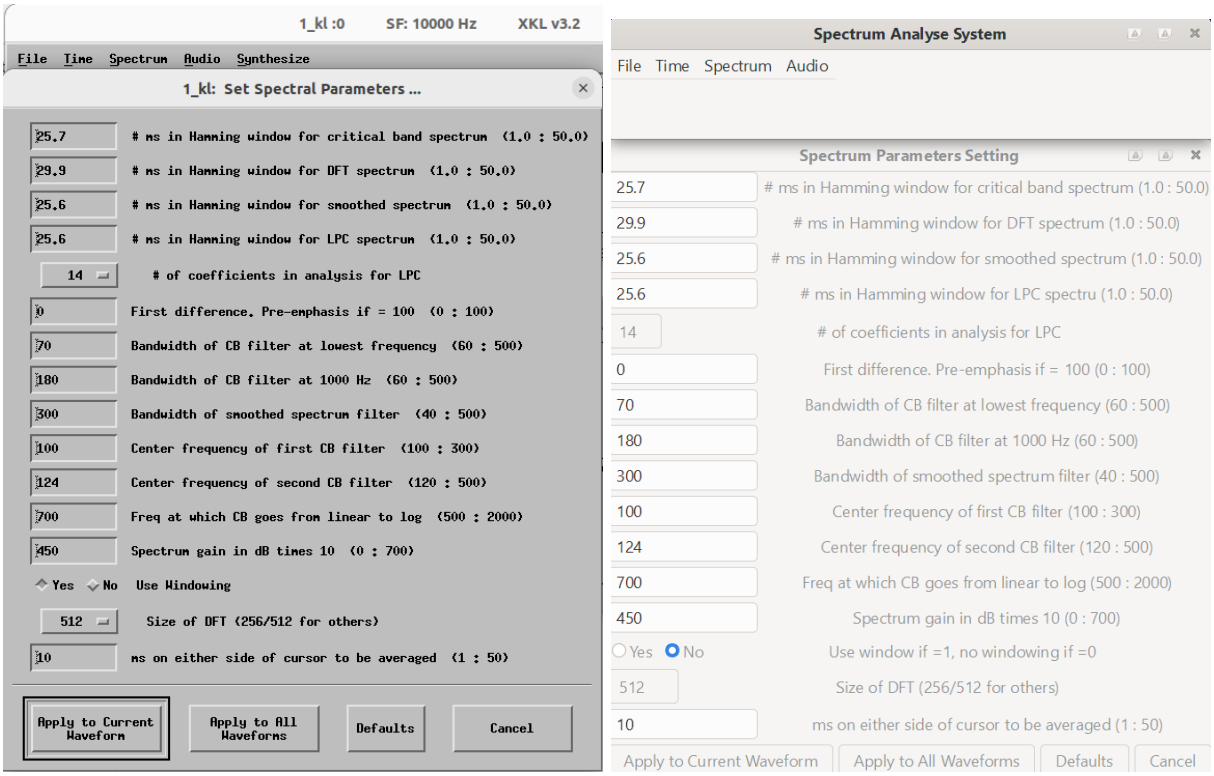
Figure 2.9: Figure (a) shows the Spectogram of the input audio signal based on the xkl motif, whereas Figure (b) displays the Spectogram created using GTK.



(a) c (b)

Figure 2.10: Figure (a) illustrates the setup of Spectrogram parameters according to the xkl motif, while Figure (b) presents the Spectrogram parameters configuration designed with GTK.

ten suffer from a fixed trade-off between time and frequency resolution. Reassigned spectrograms address this issue by redistributing energy in the time-frequency plane to more accurately reflect the instantaneous frequency and the group delay of the signal. The implementation process for Reassigned Spectrograms is depicted in the detailed diagram provided in Figure 2.12. All the functions used in this section are explained in the Appendix: Reassigned Spectrogram Code B. The process begins with reading the input speech signal from WAV files, handled by the `main()` func-



(a) (b)

Figure 2.11: Figure (a) illustrates the parameter evolution in xkl based on the motif, while Figure (b) displays the Spectrum parameter configuration in GTK-based xkl.

tion, which initializes the application. The `enterConsoleApp()` function is then called to switch the application to a console environment, bypassing the graphical user interface.

Integration Flowchart

This module enhances the spectral resolution by applying the reassigned spectrogram technique. The primary steps involved are: *Short-Time Fourier Transform (STFT)*: Computed using the `compute_stft` function, which segments the audio signal and applies FFT to each segment. *Spectral Reassignment*: Implemented in `nelsonspec.c`, this involves reassigning energy from each time-frequency bin to the center of gravity of the energy distribution within that bin. *Output Generation*: The `generate_reassigned_spectrogram` function produces a high-resolution spectrogram that provides better localization of spectral features.

Next, the `add_spectro` function is responsible for loading the WAV file and initializing the `XSPECTRO` structure, crucial for the subsequent spectral analysis. This function sets the necessary parameters for spectral analysis, such as time and index values. The command line inputs are checked to ensure that valid parameters are passed. If the parameters are valid, the process continues with setting the appropriate time and index values within the `set_time_and_index` function. This involves initializing the `saveindex` and `savetime` within the spectro structure. The core of the spectral analysis is performed by the `new_spectrum` function, which evaluates the necessary variables for the reassigned spectrogram. This function calls the `computeSTFT` function, which calculates the short-time Fourier transform, which forms the basis of the initial spectrogram. The

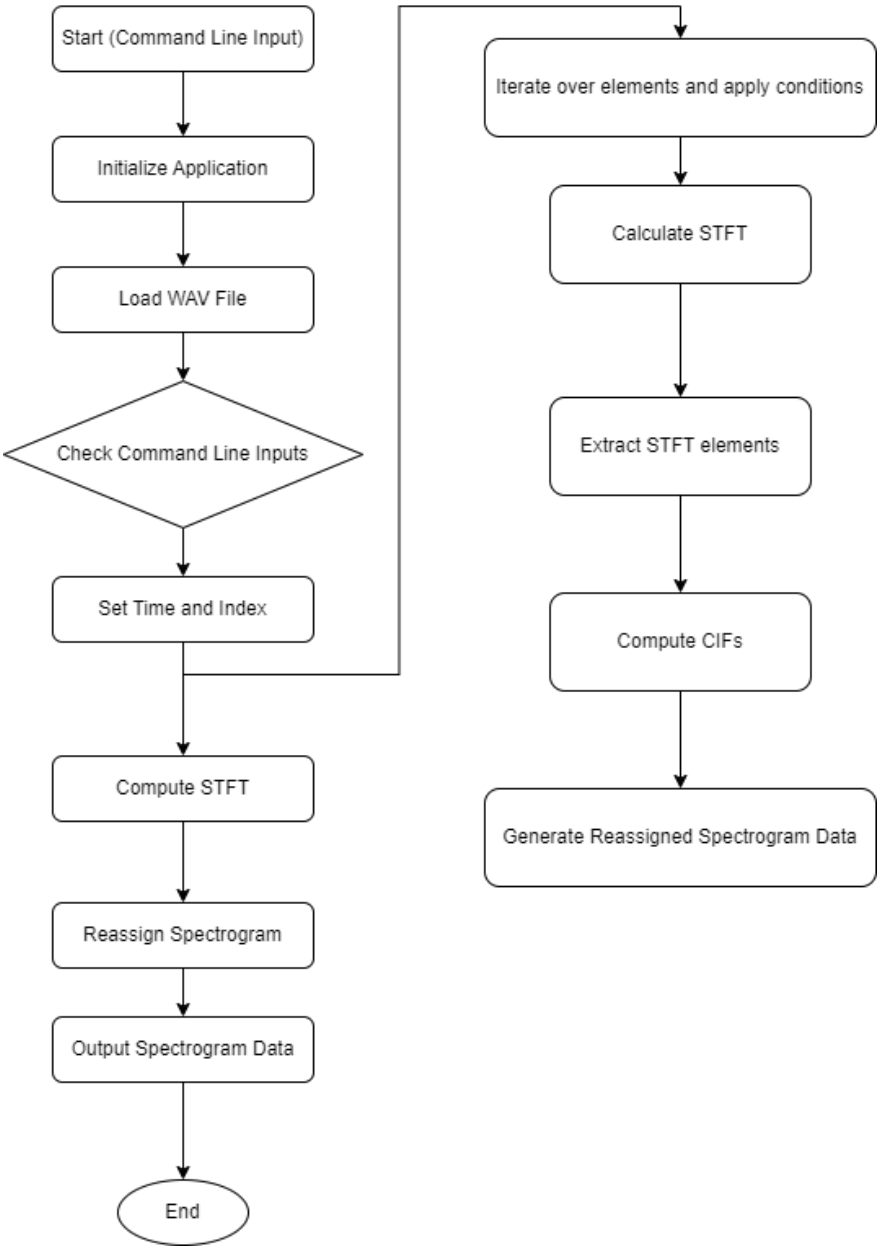
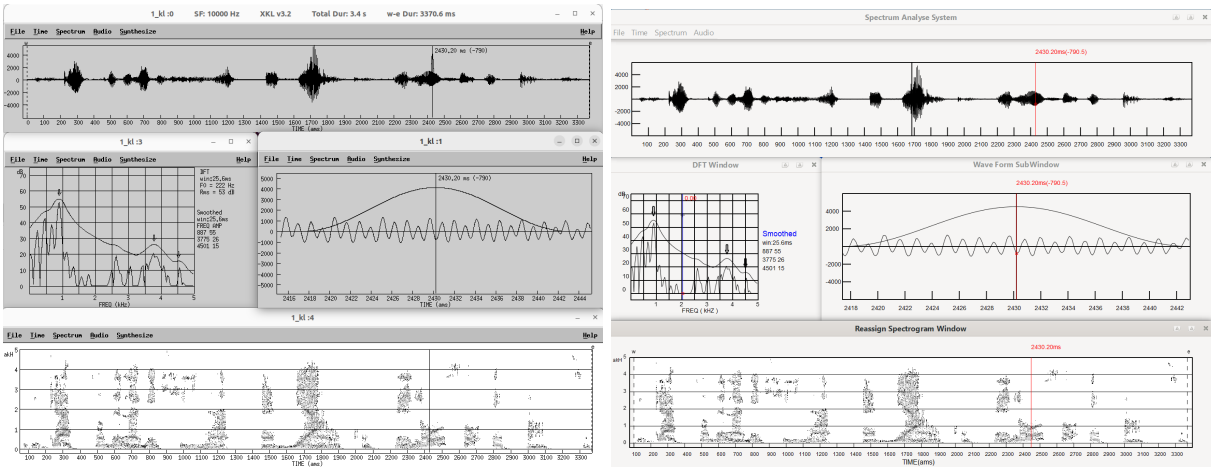


Figure 2.12: Workflow for generating a reassigned spectrogram using the xkl tool. The process begins with command-line input and proceeds through various stages of computation and data processing to output the final spectrogram data. The workflow is divided into a sequence of steps, each representing a specific action or decision point.

`computeReassignedSpectrogram` function then performs the reassignment of spectral energy based on the Channelized Instantaneous Frequency (CIF) and Local Group Delay (LGD) calculations derived from the STFT phase information.

The reassignment process involves several steps: windowing the signal using a Hanning window to minimize spectral leakage, computing the FFT of the windowed segments, and applying reassignment equations to redistribute the spectral energy. This redistribution process sharpens the spectral representation, providing higher resolution in both the time and frequency domains. The results are then visualized and analyzed, enhancing the time-frequency resolution of the spectrogram and allowing for more precise analysis of speech signals. The main libraries utilized in

this process include FFTW for efficient FFT computations, NumPy for handling numerical computations and array operations, and SciPy for providing additional signal processing capabilities. Key functions in the reassigned spectrogram calculations include those for generating the Hanning window, computing the STFT, and calculating the CIF and LGD, which are crucial for the reassignment process. Figure 2.13(a) presents the Spectrogram based on the XKL motif, while Figure 2.13 (b) depicts the Spectrogram created with GTK.



(a) (b)

Figure 2.13: Figure (a) shows the Reassign Spectogram of the input audio signal based on the xkl motif, whereas Figure (b) displays the Reassign Spectogram created using GTK.

2.7.2 Formants Saving Module Integration

Extracting formants from speech signals is vital for numerous linguistic and phonetic analyses. Formants reveal the resonant frequencies in the vocal tract, crucial for categorizing speech sounds. The xkl algorithm calculates up to the fifth formant for vowel portions, based on a method developed by Dennis Klatt, known for its precise formant estimation. In this module, we've crafted an algorithm that leverages the xkl method to process audio files and their TextGrid annotations, extract all formant data, and store it in a text file. This automated process enhances efficiency and accuracy in formant extraction, making it a valuable tool for detailed speech analysis. Landmarks or acoustic cues in speech are essential for identifying and differentiating various speech sounds. According to the Stevens model for English speech, there are eight primary landmarks: Vowel (V), Glottal (G), Nasal Closure (Nc), Nasal Release (Nr), Fricative Closure (Fc), Fricative Release (Fr), Stop Closure (Sc), and Stop Release (Sr). These landmarks correspond to distinct acoustic events and are crucial to understanding the articulatory features of speech. Vowels (V) are characterized by clear and stable resonant frequencies. Glide landmarks (G) refer to the transition points between consonantal and vocalic gestures, characterized by a smooth movement of articulators without full closure, typical in semi-vocalic consonants. Nasal closure (Nc) and nasal release (Nr) represent the closure and release of the oral constriction for nasal consonants. The velum's opening and closing, which regulates nasal airflow, is labeled on a separate tier, as it occurs independently and may not coincide with the oral constriction, especially in languages with nasalized vs. non-nasalized vowels. Fricative closure (Fc) and Fricative release (Fr) represent the constriction and subsequent

release of airflow, which produces turbulence. Stop Closure (Sc) and Stop Release (Sr) correspond to the complete blockage and sudden release of airflow. These landmarks provide a framework for articulator-free features, enabling the inference of abstract phonological features and phonemes intended by the speaker from the speech signal. Traditionally, landmarks are manually annotated using tools such as Praat and xkl. In Praat, landmarks are marked on tiers aligned with the audio speech signal and stored in a TextGrid file. Each entry in the TextGrid file includes the timestamp and corresponding landmark label. For example:

```
item [3]:
class = "TextTier"
name = "predLM"
xmin = 0
xmax = 4.2181859410430835
points: size = 51
points [1]:
number = 0.29930438370997975
mark = "V"
points [2]:
number = 0.3701303016284549
mark = "G"
points [3]:
number = 0.40454326058769247
mark = "Sc"
points [4]:
number = 0.46611805880744095
mark = "Sr"
```

This manual process involves significant labor and time, as each landmark's timestamp must be individually entered into xkl to compute formant information. This method is also prone to errors due to the repetitive and manual nature of the task, particularly when dealing with large datasets. To overcome these limitations, we have enhanced the xkl tool with an additional module that saves the formants from speech signals into a text file. The improved process allows the tool to take both the audio signal and the TextGrid file as input. The workflow is as follows: The tool validates the input arguments, ensuring that the audio file and the TextGrid file are correctly specified. The TextGrid file is read to extract all the landmark information along with their corresponding timestamps. Users can specify which landmarks to process. If no specific landmark is mentioned, the tool processes all available landmarks. The time units from the TextGrid are converted to a suitable format for further processing. The tool computes the smoothed spectrum of the audio signal, which is essential for accurate formant extraction. For each timestamp corresponding to the specified landmarks, the tool computes the formants (frequency and amplitude). The computed formant information is saved in a text file, containing the formants and their corresponding amplitudes for each landmark. By automating this process, the xkl tool significantly reduces the time and effort required for formant extraction, minimizes the possibility of human error, and improves overall efficiency. The output text file provides a comprehensive dataset of formants, facilitating

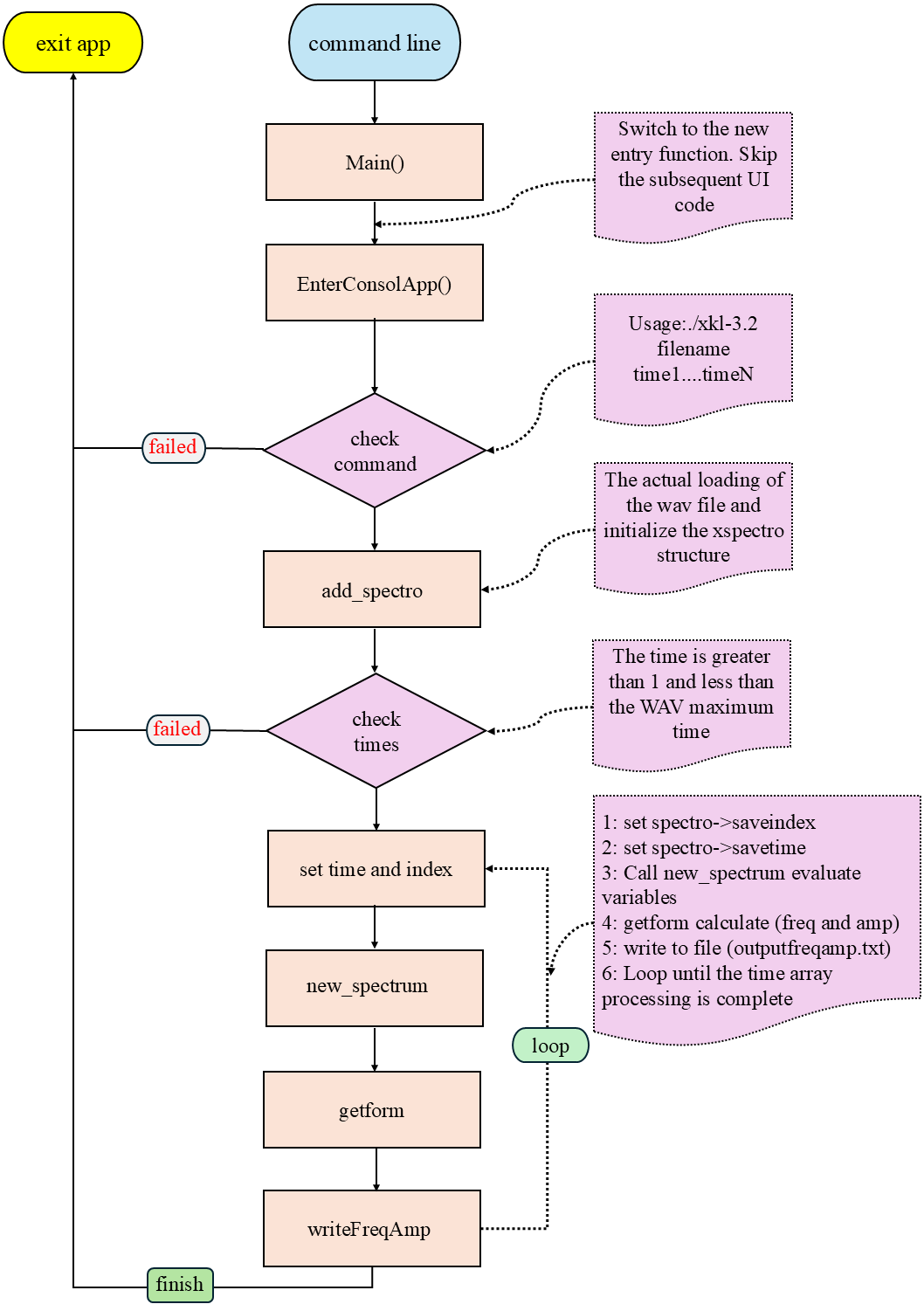


Figure 2.14: Workflow of the xkl tool for formants saving module, detailing the sequence of operations from the initial command line input to the final frequency and amplitude output. The process is divided into several key steps, each represented by different shapes indicating specific actions, decisions, and subprocesses involved in the workflow

further phonetic analysis and research. The enhanced xkl algorithm also offers flexibility. Users can specify particular landmarks for which formant information is required, or they can allow the tool to process all landmarks present in the TextGrid file. This flexibility ensures that the tool

can be tailored to specific research needs, whether for focused studies on particular speech sounds or comprehensive analyses of entire datasets. The formant saving process using the xkl tool automates and enhances formant extraction, improving efficiency, accuracy, and reliability for phonetic analysis.

Workflow of the Formant Saving Module:

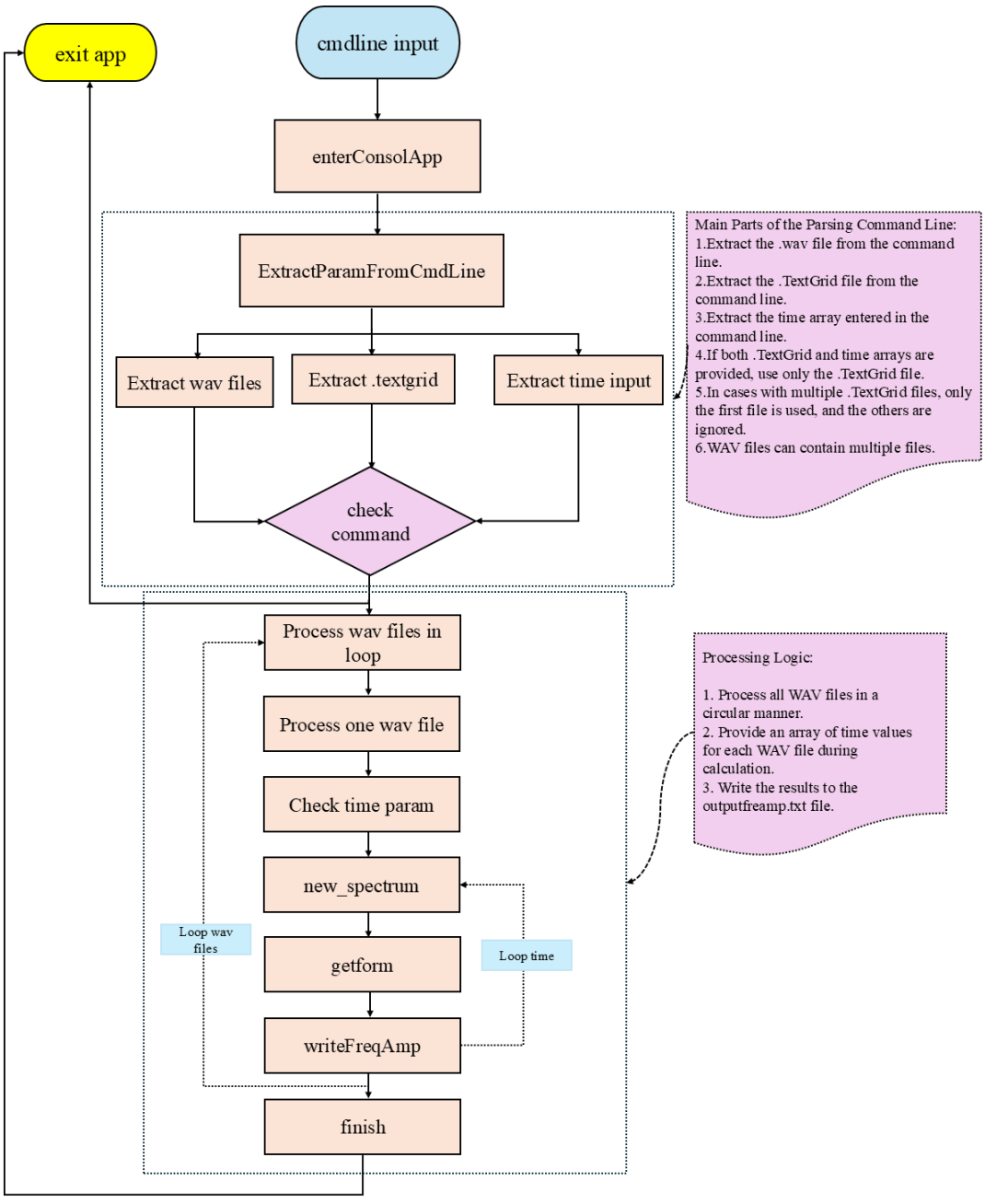


Figure 2.15: Comprehensive workflow of the xkl tool for extracting whole landmarks from audio files, detailing the sequence of operations from the initial command line input to the final output of frequency and amplitude data. The process is outlined through a series of interconnected steps and decisions, ensuring clarity in the tool's operation.

The Formant Saving Module is responsible for reading the input speech signal from WAV files, computing the spectrogram using Fast Fourier Transform (FFT), and applying the existing Linear Predictive Coding (LPC) algorithm to estimate formant frequencies. The primary functions include: *Reading WAV files*: Functions in **`wavdata.c`** and **`wavio.c`** are utilized. *Computing Spectrogram*: The **`compute_spectrogram`** function in **`spectrum.c`** leverages FFT libraries for efficient computation. *LPC Analysis*: The **`lpc_analysis`** function in **`xspec_util.c`** applies the LPC algorithm to estimate formant frequencies.

Formant Extraction: The main file handling this is **`xklfreeamp.c`**, which integrates these functionalities to automate the process. The implementation of formant saving module involved a comprehensive process, as depicted in the accompanyingFigure 2.15. The process begins with reading the input speech signal from WAV files. This step is handled by the function **`enterConsoleApp()`**, which ensures that the application runs in a console environment. The function **`add_spectro`** is responsible for loading the WAV file and initializing the **`XSPECTRO`** structure, crucial for subsequent spectral analysis.

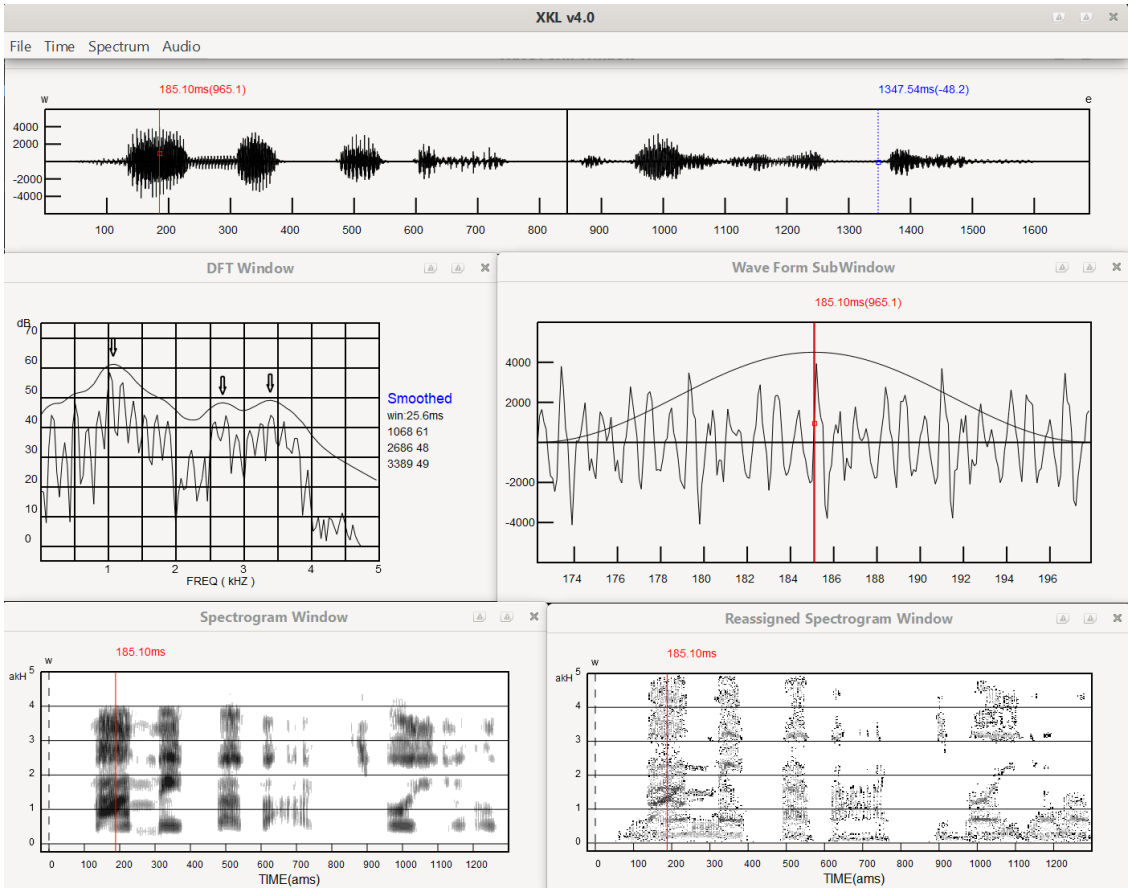


Figure 2.16: GTK based xkl 4.0

The command line inputs are checked to ensure valid parameters are passed. If the parameters are valid, the **`set_time_and_index`** step sets the appropriate time and index values for the spectral analysis. This involves initializing the **`saveindex`** and **`savetime`** within the spectro structure. The function **`new_spectrum`** then evaluates the necessary variables for spectral analysis. It calls the **`getform`** function, which calculates the formant frequencies and amplitudes using the LPC algorithm. The results are then written to an output file using the **`writeFreqAmp`** function. This loop

continues until the entire time array is processed, ensuring a comprehensive analysis of the speech signal. Key functions in the automated formant extraction code include **`printSpectro`**, which prints spectral information including filter response and discrete Fourier transform magnitudes, and **`writeFreqAmp`**, which writes frequency and amplitude information to a file. The process involves reading WAV files, computing the spectrogram, and applying formant extraction algorithms. The function **`ProcessOneWavFile`** processes individual WAV files, computes the spectrogram, and extracts formant frequencies using the **`getform`** function. The extracted formant frequencies are then written to an output file for further analysis. The main libraries utilized in this process were **`xspec_util`** for handling spectral data, **`xinfo`** for managing input and output information, **`xklspec`** for core spectral analysis functions specific to the xkl framework, **`spectrum`** for spectral analysis including the computation of spectrograms and LPC, and **`textgrid`** for handling text grid files used in annotating phonetic events. The overall view of xkl based on GTK is shown in Figure 2.16.

2.8 Conclusion

This chapter highlights substantial advancements in the xkl software, enhancing its usability, compatibility, and efficiency for comprehensive acoustic analysis. Key enhancements include the introduction of a new GTK-based graphical user interface, the incorporation of the reassigned spectrogram algorithm for improved time-frequency clarity, and a new formant-saving module that simplifies formant extraction using TextGrid files. These updates resolve the shortcomings of previous versions, making xkl more user-friendly and efficient for researchers. The updated software architecture and advanced features ensure that xkl continues to be a robust tool for speech analysis, benefiting both researchers and practitioners.

Chapter 3

Applications of AI in Speech Analysis

The goal of this chapter is to explore landmark properties for application to automatic vowel landmark detection and classification, based on the analysis of speech data derived from a database that has been specifically created toward application of Stevens Lexical Access model to the Italian language (the LaMIT database). In particular, an automated vowel landmark detection system that harnesses contemporary deep learning techniques, including a CNN-RNN hybrid model, is presented. Furthermore, this chapter introduces a novel approach based on an improved Multi-Kernel Extreme Learning Machine (MK-ELM) model, combined with the pairwise weighted scheme, to solve the multi-classification problems of foreign accents. The proposed method merges prosodic and Mel-frequency cepstral coefficient (MFCC) characteristics and employs a weighted classification scheme (WCS) to improve the identification of various non-native English accents. This framework aims to improve classification accuracy while lowering computational demands, thereby contributing to the advancement of AIFA.

3.1 Automatic Vowel Landmark Detection

Automatic landmark detection plays a critical role in implementing Stevens' (2002) model of lexical access, since landmarks are those points in time at which a primary phase of speech perception occurs. By identifying these specific points of interest in a speech signal which are particularly rich in linguistically significant information, researchers can better understand speech dynamics, improve speech synthesis, and improve speech recognition systems. Vowel landmarks are of particular relevance to our study.

3.2 What are Landmarks? (in brief)

To effectively describe Stevens' (2002) Lexical Access model, it is essential to introduce the distinction between articulator-free and articulator-bound features [2]. Articulator-free features provide a framework for classifying speech sounds based on their manner of articulation, distinguishing between vowels, glides, and various types of consonants. In contrast, articulator-bound features focus on the specific oral and non-oral articulators involved in sound production, detailing aspects such as tongue position, lip configuration, and vocal fold tension. Together, this information contributes to a comprehensive understanding of acoustic cues, enhancing our ability to analyze and

characterize speech sounds in both theoretical and applied contexts.

The landmarks are acoustic cues to articulator-free features. They refer to events at specific time points in a speech signal that convey particularly important information, because they reflect significant articulatory events that have robust and highly informative acoustic consequences [2]. These cues are critical for both speech perception and production, as they help identify and distinguish different phonetic units, such as vowels and consonants. According to Stevens' lexical access model, by focusing on these landmarks, speakers and listeners may effectively navigate the complexities of speech, enhancing communication and understanding across diverse linguistic contexts. There are eight specific landmark acoustic cues (V, G, Nc, Nr, Fc, Fr, Sc, Sr); these correspond to articulator-free features and provide critical insight into the speech signal structure. The list of landamrks is as follows:

- **Vowels (V)**: These are characterized by relatively steady-state portions of the speech signal where the vocal tract is open, allowing for the unimpeded flow of air. Vowels are critical for the formation of syllables and provide essential information about the phonetic content of speech.

- **Glides (G)**: Glides are speech sounds characterized by a smooth, gradual movement of the articulators as they transition from one position to another, typically functioning as semivowels. Unlike vowels, glides are non-syllabic, and their articulatory transitions are more rapid than those of vowels but slower and more continuous than those of consonants. This gradual movement allows glides to capture the dynamic nature of speech articulation, often occurring between vowels or as onsets to syllables.

- **Stop closure (Sc)**: This landmark marks the complete closure of the vocal tract, temporarily stopping airflow. Stop closures are key components of plosive consonants (e.g., /p/, /t/, /k/) and are crucial for identifying these sounds.

- **Stop release (Sr)**: The stop release follows the closure phase, where the articulators rapidly split, releasing the built-up air pressure. This release burst is a significant acoustic cue to identify plosive sounds.

- **Nasal closure (Nc)**: This landmark indicates the closure of the oral cavity in coordination with the lowering of the velum, allowing air to pass through the nasal cavity. It is essential to note that the velum often begins to lower during the preceding vowel before the oral constriction is fully closed. Nasal closures are crucial for producing nasal consonants (e.g., /m/, /n/).

- **Nasal release (Nr)**: Following nasal closure, nasal release marks the point where the oral cavity opens while airflow through the nasal cavity often ceases, usually due to the closing of the velum. This transition is important for distinguishing nasal sounds.

- **Fricative closure (Fc)**: This landmark represents the narrowing of the vocal tract to create turbulent airflow, characteristic of fricative consonants (e.g., /f/, /s/). Fricative closures are crucial to identifying these consonant sounds.

- **Fricative release (Fr)**: The fricative release occurs when the narrow constriction in the vocal tract is released, so that the airflow is no longer turbulent. This landmark helps to distinguish fricative sounds from other types of consonants.

3.3 Experimentation

3.3.1 Description of the Reference LaMIT Database

The analysis uses the LaMIT database described in Lexical Access Based on Detection of Landmarks and Other Acoustic Cues to Features [3]. This data set comprises 100 Italian spoken sentences recorded by two male and two female speakers and their corresponding text grid files that contains Landmark tier (LM) information. It is a comprehensive corpus of read speech crafted specifically for acoustic research on the Italian language. Notable features of the data set include:

Key Characteristics of Audio Files:

- **Source and Collection Method:** Recordings were made using a Samson Meteor Mic USB microphone in an Amplisilence recording booth by Amplifon, ensuring high-quality sound capture with minimal noise interference. The recordings were managed with Audacity software, set to a sampling rate of 44.1 kHz and 16-bit quantization.
- **Speakers:** The dataset includes contributions from four native speakers of Standard Italian (two males and two females), all of whom were born and live in Rome, Italy. This selection ensures a balanced representation of standard Italian speech patterns.
- **Sentences:** The corpus comprises 100 Italian sentences carefully selected to include all phonemes of the language, reflecting their typical frequency in written Italian. Each sentence was recorded twice by each speaker in two different sessions, resulting in 800 unique recordings.
- **File Formats:** Audio recordings are stored in WAV format with accompanying TextGrid files that contain detailed cue labels. The labeling follows a multi-tiered approach, adapted from the Lexical Access model proposed by Ken Stevens for American English and modified for Italian.

Key Characteristics of TextGrid Files:

- **Labeling Tiers:**
  - **Words**: Represents the boundaries between individual words in each sentence.
  - **LEXI phonemes**: Predicted phonemes derived from word phonetic representations, which are used to predict the acoustic cues. These labels are not time aligned with the signal, but instead the LEXI phonemes of each word are temporarily assigned equal durations, while the predicted cues are later time aligned with the signal.
  - **LM (Landmarks)**: Significant acoustic events or features in the speech signal serve as landmarks that provide information about articulator-free features of the speaker's

intended phonological segments and highlight regions rich in cues to articulator-bound features, rather than explicitly marking phonetic or prosodic boundaries.

- **LMMods**: Modifications or adjustments made to predicted LMs
- **vgplace**: Refers to the place of articulation for vowels in speech, indicating where in the vocal tract the vowel sound is produced.
- **vgplace mods**: Adjustments or refinements to the vowel place of articulation predictions based on waveform characteristics.
- **cplace**: Refers to the place of articulation for consonants, specifying where in the vocal tract the consonant sound is articulated.
- **cplace mods**: Modifications to the consonant place of articulation labels to better match the acoustic data.
- **nasal**: Identification of nasal sounds, which are produced by airflow through the nose during speech.
- **nasal mods**: Adjustments or corrections to nasal sound detection in the labeling process.
- **glottal**: refers to glottal sounds, which involve the vocal folds, like the glottal stop.
- **glottal mods**: Modifications to the predicted activity of the vocal folds

Speakers	Gender	Native language	v1	v2	Total Utterances
SPEAKER_JV	Male	Italian	100	100	200
SPEAKER_LDN	Male	Italian	100	100	200
SPEAKER_MGDB	Female	Italian	100	100	200
SPEAKER_SB	Female	Italian	100	100	200
Total					**800**

Table 3.1: Details of the LaMIT corpus recordings

Table 3.1 provides an overview of the speakers in the LaMIT corpus recordings. The dataset consists of four native Italian speakers, two males (SPEAKER_JV and SPEAKER_LDN) and two females (SPEAKER_MGDB and SPEAKER_SB). Each speaker contributed 200 utterances, equally split into two sets labeled v1 and v2, with 100 utterances in each set. The total number of utterances across all speakers is 800, ensuring balanced representation between male and female speakers, as well as consistency in the number of utterances per individual. This data set serves as a foundational element for analyzing acoustic features and conducting statistical evaluations in the present study.

3.3.2 Analysis Tools and Software

The experimental process begins with data preparation, where TextGrid files are obtained from the LaMIT database. These files contain detailed phonetic annotations and the positions of various landmarks are extracted. These landmarks are used to segment the data, allowing for a structured analysis of the intervals between phonetic events. In the landmark labeling stage, eight types of

landmark are identified: vowels (V), glides (G), stop closure (Sc), stop release (Sr), nasal closure (Nc), nasal release (Nr), fricative closure (Fc) and fricative release (Fr). After identification, the landmarks are manually verified to ensure their accuracy and consistency across all recordings, a crucial step to maintain the integrity of the data. All codes, and all Excel files containing all the acoustic measurements that were used in the landmark classifier are reported in Appendix C.

3.3.3 Extraction of Parameters

Parameter extraction is a first step in the automatic landmark detection process, as it transforms raw speech data into a set of measurable characteristics that can be used for model training. In this study, we focus on two types of features: Energy parameters and Mel-spectrogram parameters.

Energy

Energy parameters may play an important role in the detection of landmarks in speech signals. Vowel sounds, for example, typically have higher energy compared to other phonemes. Energy parameters are derived from the speech signal by calculating the short-term energy over a defined window. The energy E of a signal frame $x[n]$ is calculated using the formula given by:

$$E = \sum_{n=1}^{N} x[n]^2, \tag{3.1}$$

where $x[n]$ represents the amplitude of the signal in sample n and N is the total number of samples in the frame.

Mel-Spectrogram

The Mel-spectrogram is a technique that captures the detailed spectral characteristics of speech. It may be useful for vowel landmark detection thanks to its ability to represent harmonic structures and formant frequencies.The Mel-spectrogram aligns with human auditory perception [87], making it effective for phonetic analysis.

The extraction process begins with applying the Short-Time Fourier Transform (STFT) to obtain the magnitude spectrum, using the sampling rate of 10,000 Hz, and a frame length of 10 ms. Frequencies are then mapped to the Mel scale using triangular filters, the way in which the human auditory system perceives fundamental frequency as pitch. This scale emphasizes perceptually meaningful frequency components, crucial for identifying formant structures in vowels. The resulting Mel-spectrogram is further processed by converting it to a logarithmic scale, which compresses the dynamic range and highlights spectral variations. In this study, the Mel-spectrogram was parameterized with 128 Mel frequency bins.

The STFT can be expressed as follows:

$$X[n,k] = \sum_{m=0}^{N-1} x[m] \cdot w[n-m] \cdot e^{-j\frac{2\pi}{N}km}, \tag{3.2}$$

where:

- $X[n,k]$ represents the STFT at time frame n and frequency bin k,

- $x[m]$ is the input audio signal,
- $w[n-m]$ is the window function applied to the signal to manage frame transitions (usually Hamming or Hann window),
- N is the FFT size (frame length in samples),
- k is the frequency bin index.

Following this, the frequencies are mapped to the Mel scale using a filter bank of 128 Mel filters. The mapping can be described using triangular filters, where each filter $H_m(k)$ emphasizes a particular range of frequencies. The Mel-spectrogram is computed by applying these filters to the magnitude spectrum:

$$M[n,m] = \sum_{k=1}^{K} H_m(k) \cdot |X[n,k]|, \tag{3.3}$$

where:

- $M[n,m]$ is the Mel-spectrogram for time frame n and Mel filter m,
- $H_m(k)$ is the triangular Mel filter applied at frequency bin k,
- $|X[n,k]|$ is the magnitude spectrum of the STFT.

To make the representation more perceptually relevant, the logarithm of the Mel-spectrogram is taken, transforming it into the log-Mel spectrogram, which compresses the dynamic range and emphasizes weaker components that may be crucial for distinguishing vowel sounds:

$$\text{Log-Mel}[n,m] = \log(M[n,m] + \epsilon), \tag{3.4}$$

where:

- ϵ is a small constant added to avoid taking the logarithm of zero.

3.3.4 Convolutional and Recurrent Neural Networks

The model architecture used for automatic landmark detection in this study is a hybrid of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). This hybrid model effectively leverages the strengths of both CNNs and RNNs to capture both spatial and temporal features of the speech signal.

The CNN component excels at extracting local features and patterns from the input data, particularly from the Mel-spectrogram representation. Conversely, the RNN component is designed to model the temporal dependencies and sequence dynamics inherent in speech, making it ideal for processing sequential data like speech signals. In this model, Long Short-Term Memory (LSTM) [88] units are employed to retain information over time, allowing for effective handling of variations in speech.

A critical step in preparing the input for the hybrid model involves the concatenation of Energy and Mel-spectrogram parameters. Energy parameters capture the amplitude dynamics of the speech signal, providing insights into the power associated with different phonetic components, particularly

vowels. The Mel-spectrogram, with its perceptually meaningful frequency components, captures the harmonic structures and formant frequencies essential for distinguishing vowels. The concatenated feature vector $F[n]$ for each time frame n combines both sets of parameters as follows:

$$F[n] = [E[n], M[n, 1], M[n, 2], \ldots, M[n, 128]] \tag{3.5}$$

where:

- $E[n]$ represents the Energy parameter for frame n,
- $M[n, m]$ denotes the Mel-spectrogram values across the 128 Mel frequency bins for frame n.

This unified feature representation enriches the input to the model, enabling it to learn complex patterns that are indicative of vowel landmarks. The integration of Energy and Mel-spectrogram parameters allows the model to first capture local spectral features through convolutional operations, followed by modeling their temporal dependencies using LSTMs. After processing through the LSTM layers, the output is passed through dense layers for final classification into vowel landmarks or non-landmarks. The implementation of the model for automatic landmark detection involved several key steps, from data loading and preparation to model training and evaluation. The dataset used for this task included recordings from multiple speakers and versions, specifically JV (JV_V1, JV_V2), LDN (LDN_V1, LDN_V2), MGDB (MG_V1, MG_V2), and SB (SB_V1, SB_V2).

The model was implemented in Python, and the corresponding code can be found in Appendix C. The first step in the process was to load and prepare the data. This involved reading the CSV files that contain energy parameters and Mel spectrogram parameters for each speaker and merging them into a single file . The parameters used for model training included energy and Mel-spectrogram parameters, while labels indicating the presence of vowel landmarks were extracted from the "LM Tier Info" column. To ensure consistency in the training process, the parameters were normalized using a StandardScaler (built-in function in Python) to have zero mean and unit variance. The labels were encoded using a LabelEncoder (built-in function in Python) and converted to categorical format from "V" and "NV" to pre-defined numerical values for the classification task. The CNN-RNN hybrid model was then constructed using Keras (Python built-library). The implementation of the model architecture comprised a convolutional layer with 64 filters, followed by a max-pooling layer and dropout for regularization. This was followed by two LSTM layers, each with 100 units, and additional dropout layers to prevent overfitting. The final dense layer with a softmax activation function was used to classify the input frames into vowel landmarks or non-vowel landamrks. The model was compiled using the Adam optimizer and categorical cross-entropy loss function, and accuracy was used as the evaluation metric (see code in Appendix C). For training, the dataset was split into training (80%) and testing 20%) sets. Each input sequence was reshaped to have 10 timesteps, with the number of features per timestep adjusted accordingly. The model was trained for 20 epochs with a batch size of 64, and validation data was used to monitor the training process and prevent overfitting. The performance of the model was evaluated individually for each speaker as well as for the combined dataset. Key metrics such as accuracy, precision, recall, and F1 score were calculated to assess the model's effectiveness. The results were summarized in a performance table, showing the evaluation metrics for each speaker and for the combined dataset. This comprehensive

evaluation provided insights into the model's ability to generalize across different speakers and versions, highlighting its strengths and areas for improvement. Figure 3.1 illustrates the architecture of the CNN-RNN hybrid model used for automatic landmark detection.

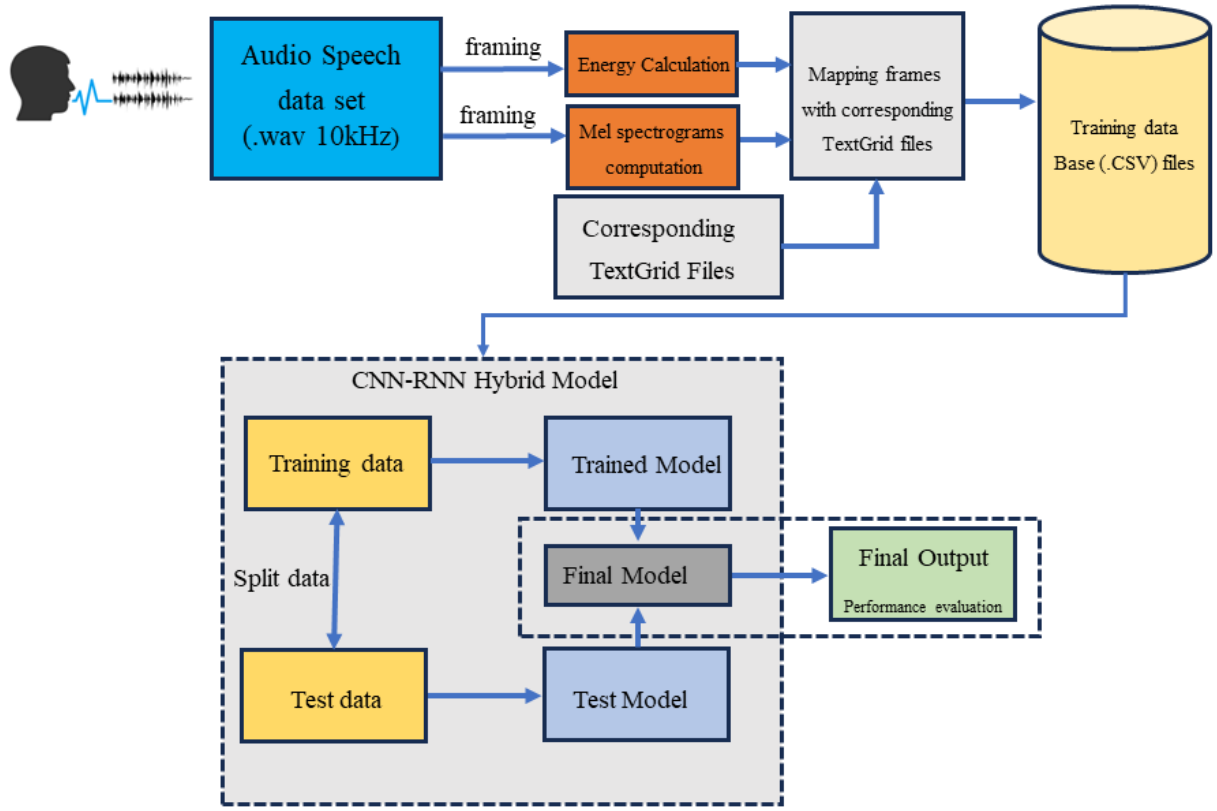


Figure 3.1: Automatic Landmarks Detection process from an audio speech dataset (.wav 10kHz). The system performs energy calculation and Mel spectrograms computation, followed by mapping frames with corresponding TextGrid files to create a training database. A CNN-RNN hybrid model is trained and tested on split data to produce a final output for performance evaluation.

3.3.5 Automatic Vowel Landmark Detection: Results and Discussion

Performance is evaluated based on accuracy, and a confusion matrix is generated to assess classification errors, while a classification report provides precision, recall, and F1 scores as summarized in Table 3.2.

Speaker	Accuracy	Precision	Recall	F1 Score
JV	74.15%	66.45%	74.15%	65.07%
LDN	72.00%	71.21%	72.00%	71.54%
MGDB	75.46%	70.45%	75.46%	69.40%
SB	77.65%	67.55%	77.65%	68.50%
Combined	74.98%	69.85%	74.98%	67.88%

Table 3.2: Performance for Combined Dataset

Results indicate that the model performs reasonably well across all speakers, with accuracy ranging from 72.00% to 77.65%. The combined dataset, which includes data from all speakers,

shows an overall accuracy of 74.98%. For speaker JV, the model achieved an accuracy of 74.15%. The precision was 66.45%, recall was 74.15%, and the F1 score was 65.07%. These results suggest that while the model is relatively good at identifying vowel landmarks for this speaker, there is room for improvement in terms of precision and F1 score, indicating some false positives in the detection process. The model's performance with speaker LDN yielded an accuracy of 72.00%, the lowest among the individual speakers. However, the precision was 71.21%, recall was 72.00%, and the F1 score was 71.54%. The close values of precision, recall, and F1 score suggest a balanced performance, though the overall accuracy indicates that the model may struggle with this particular speaker's data, possibly due to unique speech patterns or variability. For speaker MGDB, the model achieved an accuracy of 75.46%, with a precision of 70.45%, recall of 75.46%, and F1 score of 69.40%. These results highlight a strong performance with a good balance between precision and recall, suggesting that the model can effectively detect vowel landmarks in MGDB's speech with fewer false positives and negatives compared to other speakers. The highest performance was observed with speaker SB, where the model achieved an accuracy of 77.65%. The precision was 67.55%, recall was 77.65%, and the F1 score was 68.50%. This indicates a high recall rate, meaning the model is very effective at identifying most of the vowel landmarks in SB's speech, though there is still some room for improvement in precision. The combined dataset, which aggregates data from all speakers, shows an overall accuracy of 74.98%. The precision was 69.85%, recall was 74.98%, and the F1 score was 67.88%.

These results show that the model maintains robust performance across diverse speech patterns and can generalize well to different speakers. The combined performance metrics are an indication of the model's capability to handle variability in speech data, making it suitable for broader applications in speech processing. The variability in performance across different speakers suggests that individual speech characteristics, such as pitch, tone, and speaking style, can influence the model's effectiveness. The high recall rates, especially for speakers SB and MGDB, indicate that the model is proficient in detecting vowel landmarks but might produce some false positives, as reflected in the precision scores. This robustness is crucial for practical applications, where the model will need to handle a variety of speech inputs. The relatively lower precision scores indicate that further refinement may be needed to reduce false positives, possibly through more sophisticated feature extraction techniques or advanced model architectures. In conclusion, the CNN-RNN hybrid model demonstrates strong potential for automatic landmark detection in speech processing, with consistent performance across individual speakers and the combined dataset. Future work could focus on enhancing precision and exploring the impact of different model configurations to further improve detection accuracy and reliability.

3.4 Automatic Identification of Foreign Accents

Automatic Identification of Foreign Accents (AIFA) focuses on identifying the native language of a speaker when they converse in a non-native tongue. This field has attracted extensive research attention, driven by its relevance to applications such as automatic speech recognition (ASR), speaker identification, e-learning, and security measures such as immigration checks. Foreign accents create specific variations in pronunciation, prosody, and vocal traits, often reducing the performance of ASR systems optimized for native speakers. AIFA poses significant challenges, particularly in mul-

ticlass classification contexts. The difficulties stem from the need to manage diverse accents and imbalanced datasets, as well as selecting effective classification features. Traditional machine learning techniques, such as Gaussian Mixture Models (GMMs) and Hidden Markov Models (HMMs), have offered some solutions, but struggle with the variability and complexity of foreign accents. Recent progress has been made using advanced methods, such as deep learning, although these approaches still face issues with regard to computational efficiency and classification accuracy. This study introduces a novel approach based on an improved Multi Kernels Extreme Learning Machine model (MK-ELM), which addresses these challenges through a weighted classification strategy. The proposed method merges prosodic and Mel cepstral coefficient MFCC characteristics and employs a weighted classification scheme (WCS) to improve the identification of various non-native English accents. This framework aims to improve classification accuracy while lowering computational demands, thereby contributing to the advancement of AIFA.

3.5 Multi Kernals Extreme Learning Machine (MK-ELM)

In this section, we provide an overview of the fundamental principles behind the Extreme Learning Machine (ELM) algorithm and its kernelized version, known as KELM. Building upon the strengths of KELM, we further enhance its capabilities by introducing the Linear Combination of Kernels (KLC) technique. This technique is based on the Multi-Kernel ELM algorithm (MK-ELM) and is designed to address multi-classification problems effectively. The kernel learning method was introduced into the ELM to enhance the stability and generalization capability [89]. The kernel matrix Ω_{ELM} , constructed to replace HH^T , can be defined as shown in equation (3.6):

$$\Omega_{ELM} = HH^T : \Omega_{ELM_{i,j}} = h(x_i)h(x_j) = K(x_i, x_j) \tag{3.6}$$

where $h(x)$ represents the hidden layer mapping. The output of KELM can be expressed as follows in equation (3.7) and (3.8):

$$f(x) = h(x)H^T\left(I/C + HH^T\right)^{-1} Y \tag{3.7}$$

$$= \begin{bmatrix} K(x, x_1) \\ \vdots \\ K(x, x_N) \end{bmatrix}^T (I/C + \Omega_{ELM})^{-1} Y \tag{3.8}$$

3.6 Multi-Kernel ELM combined with Kernel Linear Combination (KLC)

To address the challenge of multi-classification, we extend the Kernel Extreme Learning Machine (KELM) algorithm and propose the Kernels Linear Combination (KLC) approach within the framework of the Multi-Kernel ELM (MK-ELM) algorithm. Although the single-kernel learning approach is retained, the KELM algorithm combines the advantages of ELM and the generalization capabil-

ity of the Support Vector Machine (SVM) methods. The performance of a classification algorithm is influenced by the choice of kernel parameters and the type of kernel function used. Since KELM relies on a single kernel function, it has limitations in terms of detection accuracy, robustness, and the ability to select the most suitable kernel function for a given multi-classification scenario. Multi-kernel functions offer advantages, as they improve mapping performance by combining kernel functions with diverse features [90]. Mercer's theorem [90] provides a sufficient condition for constructing kernel functions, stating that any semi-positive definite symmetric function can be used as a kernel function. Different kernel functions yield varying effects on the performance of the constructed MK-ELM model. Fig. 3.2 illustrates a schematic diagram of this algorithm.

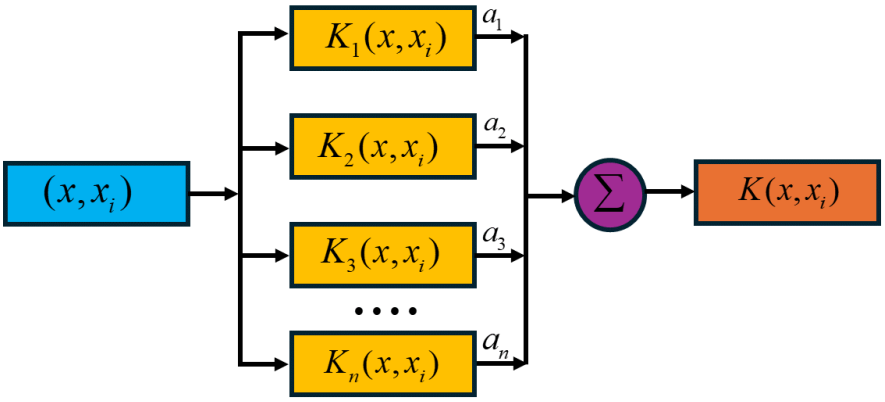


Figure 3.2: Illustration of the Kernel Linear Combined methods implemented to create multi-kernel functions [5]

.

Consider $K(x, x_i)$ as a given kernel function and $\hat{K}(x, x_i)$ as its normalized version. The normalization of the kernel function is defined as $\sqrt{K(x,x)K(x_i,x_i)}$. The algorithm is developed based on the Mercer theorem for kernel functions, integrating the multikernal (MK) learning approach with the ELM model and the proposed MK-ELM model. The derivation of the algorithm follows. Generally, the linear combination of MK functions employs various kernels. For instance:

- Linear kernels: $K(x, x_i) = x \cdot x_i$
- Gaussian kernels: $K(x, x_i) = \exp\left(-\frac{\|x-x_i\|^2}{\delta^2}\right)$
- Polynomial kernels: $K(x, x_i) = (x \cdot x_i + 1)^d$

To address the limitations inherent in individual kernel functions, the MK function can be adopted, and its representation is provided as follows:

$$K(x,x_i) = a_1K_1(x,x_i) + a_2K_2(x,x_i) + a_3K_3(x,x_i) + \cdots + a_nK_n(x,x_i) \quad (3.9)$$

$$\text{s.t.} \quad \sum_{k=1}^{n} a_k = 1, \quad \forall\, a_k \geq 0 \tag{3.10}$$

The optimization problem of the MK ELM can be described as shown in equation (3.11), (3.12) and (3.13):

$$\min L_{MK-ELM} = \frac{1}{2} \sum_{k} \frac{1}{a_k} \|w_k\|^2 + C \frac{1}{2} \sum_{i=1}^{N} \zeta_i^2 \tag{3.11}$$

$$\text{s.t.} \quad \sum_{k} K(x, x_i) w_k = t_i - \xi_i, \quad i = 1, \ldots, N, \tag{3.12}$$

$$\sum_{k} \frac{1}{a_k} = 1, \tag{3.13}$$

where w_k is the feature weight corresponding to the adopted kernel function $K(x, x_i)$, ξ_i is the predicted error of sample i, and C is the regularization parameter to balance model complexity and predictive performance. By replacing the kernel matrix of Equation 3.9 in the ELM with the newly constructed multi-kernel function, the multi-kernel extreme learning machine (MK-ELM) Algorithm is obtained as defined in Algorithm (1).

Algorithm 1 based on Multiple Kernals of ELM.

Step1: Initialize sample set N,

$$N = \{(x_i, t_i) \,|x_i \in R^n \,,\ t_i \in R^n \,, i = 1, 2, ..., N\},$$

Step2: The best MK-ELM is created by combining various single-kernel functions according to the multi-kernel formula 3.9, choosing the best kernel function combination, and figuring out the regularization parameter C and kernel parameters.
Step 3: Weights w and bias b between the input layer and hidden layer that was produced at random using the MK-ELM algorithm,
Step 4: With training samples as input, hidden layer output matrix H and layer weight matrix β are calculated using equations 3.9 and 3.10,
Step 5: Comparing the performance of MK-ELM made up of various single-kernel functions, evaluating the effectiveness of MK-ELM using a number of experiments,
Step 6: Return the MK-ELM classifier along with the classification outcome.

3.7 Experimentation

This section provides an overview of the corpus, along with the proposed Foreign Accent Identification (AIFA) model and the weighted scheme. Additionally, it outlines the methodology for combining features and describes the attributes of the selected features.

3.7.1 Speech Dataset

The "Speech Accent Archive," which is kept up to date by George Mason University (GMU) [6], is a collection of English spoken by people with different first languages. Every speaker gives a

paragraph in English that includes both difficult-to-pronounce sounds and commonly used words. This paragraph contains 69 English words. The format of these audio recordings is (.mp3). The text is as follows:

"Please call Stella. Ask her to bring these things with her from the store. Six spoons of fresh snow peas, five thick slabs of blue cheese, and maybe a snack for her brother Bob. We also need a small plastic snake and a big toy frog for the kids. She can scoop these things into three red bags, and we will go meet her Wednesday at the train station".

The GMU Speech Accent Archive that formed the basis of this analysis includes 580 participants, speaking the following languages (Table 3.3)

Table 3.3: The GMU (Speech Accent Archive) [6] offers a collection featuring samples of English spoken by subjects with six distinct native languages.[6]

Native language	Total speakers	Males	Females	Birthplace	Time duration (hh:mm:ss)
English	100	46	54	USA	38:47
Arabic	100	55	45	Saudi Arabia: 97 (54M, 43F) U.A.E: 3 (1M, 2F)	56:20
Chinese	100	34	66	China	53:10
Korean	100	38	62	South Korea	51:20
French	80	41	39	France: 27 (13M, 14F) Canada: 8 (5 M, 3 F) Belgium: 20 (12M, 8F) Switzerland: 15 (8M, 7F) Portugal: 10 (3M, 7F)	37:35
Spanish	100	57	43	Spain: 36 (20M, 16F) Argentina: 17 (12M, 5F) el Salvador: 25 (15M, 10F) Mexico: 22 (10M, 12F)	48:20
Total	**580**	**271**	**309**		**4:45:32**

3.7.2 Acoustic Attributes

The characteristics of speech included prosodic and MFCCs elements, as detailed in the subsections below.

Mel-frequency cepstrum (MFCC)

The procedures involved in calculating MFCCs (Mel-frequency cepstral coefficients) included: 1) pre-emphasis; 2) windowing; 3) DFTM; and 4) application of a Mel-filter bank, logarithmic processing, and discrete cosine transform. A Python library is eused to extract MFCC characteristics [91]. For spectrum processing from the DFT, a series of triangular filters is constructed, with the filters logarithmically spaced above 1000 Hz, and uniformly spaced below 1000 Hz. The feature set consists of the first 39 cepstral coefficients. HTK (Hidden Markov Model Toolkit) is a software toolkit used to build and analyze hidden Markov models, primarily for the recognition of speech and the concatenation of features [92]. In our work, HTK is used to concatenate the MFCC features with their derivatives.

Prosodic Features

As well known, suprasegmental speech components, i.e.prosodic features, are essential for differentiating among accents and dialects [93]. In this study, pitch and energy were the selected parameters

to describe prosodic information.

- Pitch

When spoken sounds are articulated, the frequency at which the vocal cords vibrate is known as pitch, or fundamental frequency (F0). The pitch contour of an uttered word is important for several applications, including emotional state recognition, speaker identification, and voice activity detection. The pitch can be determined through several techniques; a widely used method is the auto-correlation technique [94].The pitch (F0) of a speech signal is computed using Praat's [7] short-time autocorrelation method and pitch stylization functions, utilizing a frame size of 25 ms.

- Energy:

As shown in the equation, normalized frame-level energy serves as a prosodic feature of accents (3.14):

$$E_{norm}(i) = \frac{1}{f_L} \sum_{n=1}^{f_L} |s_i(n)|^2 \tag{3.14}$$

where, $s_i(n),\ n = 1, ...f_L$ are the audio samples in the i^{th} frame and f_L is the length of the frame.

3.7.3 Feature Combination

The paramaters that were considered in this study were the first 13MFCC, along with their first and second derivatives, and also prosodic elements such as F0 (fundamental frequency) and energy, together with their respective derivatives. The study separately considered the max., min. and mean values of F0 as distinct attributes. A performance comparison was carried out to investigate how these feature sets performed both individually and collectively. This resulted in four comparative conditions: MFCCs by themselves, MFCCs with F0, MFCCs with energy, and MFCCs incorporating both F0 and energy. The features were concatenated on a per-frame basis for analysis.

3.7.4 System Architecture

The block diagram of the proposed system is shown in Figure 3.3. Multiclass classification was based on the Multi-Kernel ELM Model, as detailed in Section (3.6). The input to the model is the set of features as defined in previous paragraphs. To assess the performance of the model, the data were divided into training (80%) and testing (20%.) All parameters were tuned to increase the model efficiency.

3.7.5 Weighted Scheme Architecture

We propose a novel weighted scheme for multi-classification tasks, which utilizes pairwise input data. The block diagram in Figure 3.4 illustrates the integration of this weighted scheme with our proposed MK-ELM model. The WCS algorithm consists in three steps: 1) we train multiple models using a pairwise classification approach to differentiate main boundaries [58]. The main parameters for each model are well tuned using K-fold cross-validated testing; 2) the outcome of

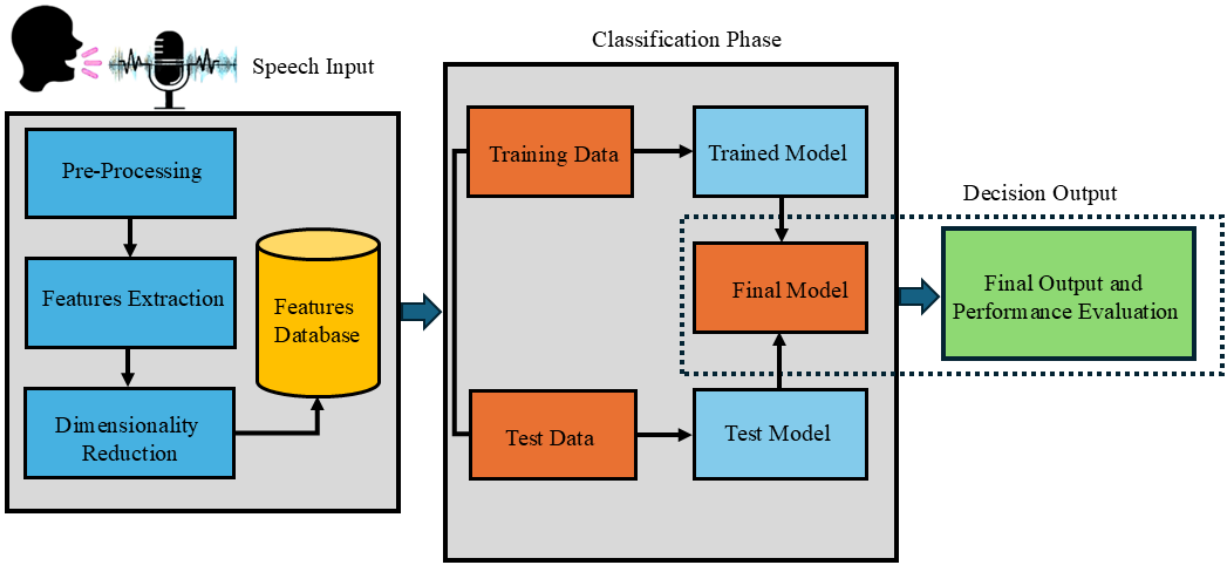


Figure 3.3: System model: pre-processing, feature extraction, classification, performance evaluation [5].

the models are collected to get the classification result by counting the number each class selected. Based on these count results, each class assigned a weight; 3) the overall accuracy is observed and the data class having the highest score is chosen as the output. The maximum count a class can achieve correlates with the number of pairwise combinations in which it is involved, as detailed in Algorithm 2. This algorithm, when applied to various accent classification models, has shown superior performance over other methods. Detailed results and analysis will be discussed in the following sections.

3.7.6 Pre-processing

The "Speech Accent Archive" [6] contains audio files in (.mp3) format that were converted to (.wav) for this research. Multiple preprocessing steps were performed for consistency. The sampling rate was reduced from 44.1 KHz to 16 KHz, and the audio was converted to mono. In the last step of preprocessing, the version 2.1.2 of the Audacity tool was used to normalize the amplitude across all audio files to ensure a consistent signal energy level. This step was performed to minimize any variations in recording volume, enabling fair comparisons of acoustic features across different audio samples during analysis.

3.7.7 Hardware and Software Tools

The experiments took place on a hardware setup featuring a Core™ i7-7500U CPU (2.70 GHz - 2.90 GHz). The entire codebase was developed using Python version 3.10.

3.7.8 Implementation of the Models

The results of a series of experiments carried out in two different phases are presented in this section. In the first phase, we used a conventional one-vs-all multi-classification technique to evaluate the

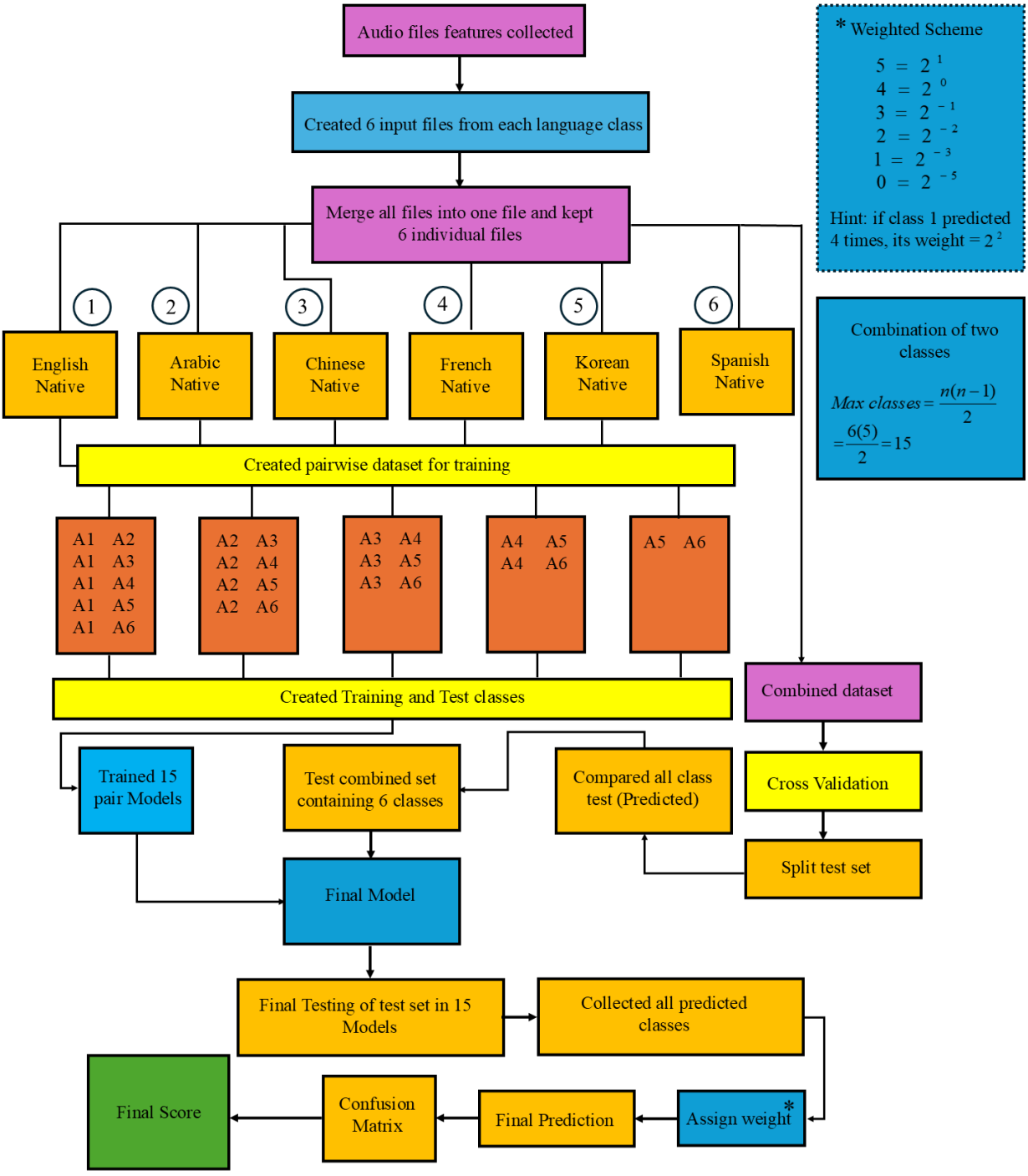


Figure 3.4: A weighted scheme methodology is illustrated in this model for multi-classification utilizing the proposed MK-ELM model [5].

Algorithm 2 Weighted Scheme Classification (WSC) Algorithm

Require: Input data with n instances and k classes
Ensure: Predicted class for each instance
Divide the data into p pairwise subsets
for $i = 1$ to p **do**
    Train a binary classifier on the i^{th} pairwise subset
    Tune the hyper-parameters of each classifier using cross-validation
end for
Initialize an array W to store weights for each pairwise classifier
for $i = 1$ to p **do**
    Predict the class probabilities for each instance using the i^{th} binary classifier
    Calculate the accuracy of the binary classifier on the validation set
    Calculate the weight w_i for the i^{th} binary classifier based on its accuracy
    $W[i] \leftarrow w_i$
end for
Normalize the weights: $W \leftarrow \frac{W}{\sum_{i=1}^{p} W[i]}$
Initialize an array P to store the combined class probabilities for each instance
for $i = 1$ to p **do**
    Predict the class probabilities for each instance using the i^{th} binary classifier
    Combine the class probabilities using the pairwise weighted scheme: $P \leftarrow P + W[i] \times$ Class Probabilities from i^{th} classifier
end for
Choose the class with the highest score in P as the predicted class for each instance

performance of our proposed MK-ELM model. Using a dataset with six categories, the MK-ELM model's performance was compared to numerous industry-leading models, including LSTM, SVM, ELM, ANN, ML-ELM, and KELM. Accuracy and performance metrics for every model were gathered and assessed. Using a weighted method based on paired inputs, we improved the MK-ELM model's multi-classification capabilities in the second phase. We also applied the WCS strategy to different state-of-the-art models and noted their efficiency and precision for comparison with our proposed model.

3.7.9 Research Flowchart

The flowchart in Figure 3.5 illustrates the systematic methodology of the research framework, detailing each stage from sample set initialization to final classification results. Through parameter optimization and rigorous model training, the MK-ELM model effectively captures accent variations, aided by the Weighted Scheme Algorithm for precise weight computation, ensuring robust and standardized accent identification. The diagram delineates each stage, showcasing the integration of the Multi-Kernel Extreme Learning Machine (MK-ELM) model and the Weighted Scheme Algorithm with Weight Computation for robust accent identification.

3.8 Results and Discussion

This section provides the results and explores the efficiency of the proposed model by using cross-validation based on K-Fold, highlighting the resilience of the model after its assessment. The success of the model is demonstrated by the debate, which highlights its quick training times and

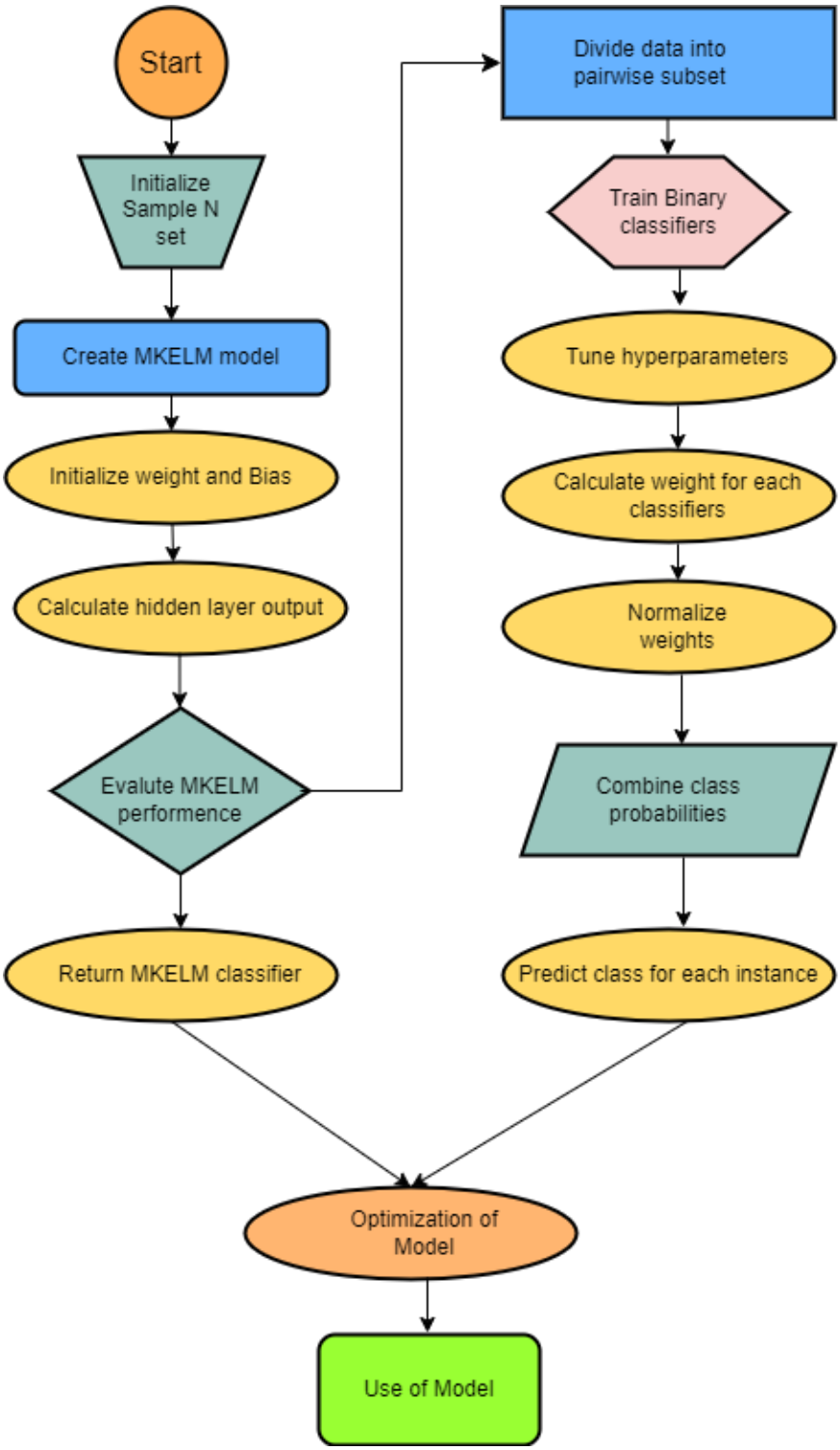


Figure 3.5: A flowchart outlines each phase, from theinitialization of the sample set to the final classification results. It highlights the application of the Multi Kernels ELM model and the WCS Computation to guarantee precise accent detection [5].

accurate accent identification. A comparison with previous research shows where the model has progressed and points out possible areas for improvement.

The experiments were conducted in four steps, each focusing on a different combination of features: (1) baseline features consisting of only MFCCs, (2) MFCCs combined with Pitch, (3) MFCCs combined with Energy, and (4) all features combined (MFCCs, Pitch, Energy). The results of implementing SVM, ANN, LSTM, ELM, ML-ELM, and KELM models on the six-class dataset are presented and summarized in Table 3.4. When the SVM model utilized solely MFCCs

Table 3.4: Evaluation of classification accuracy across various models with different feature combinations. All results are derived from the standard multiclass classification methodology [5].

	Accuracy achieved (%)				
Models	MFCCs	MFCCs + Pitch	MFCCs + Energy	MFCCs + Pitch + Energy	(%) Increment due to prosodic features
SVM	34.30	36.0	37.50	38.56	4.26
ANN	16.31	18.70	19.90	20.83	4.52
LSTM	22.0	24.05	25.88	26.93	4.93
ELM	27.40	28.76	29.02	31.88	4.48
ML-ELM	32.29	33.17	35.38	37.0	4.71
KELM	49.70	52.18	53.97	55.0	5.3
MK-ELM (Proposed)	60.0	62.23	63.99	65.5	5.5

as baseline features, it achieved an accuracy of 34.30%. This accuracy improved to 36.0% when MFCCs were augmented with Pitch features, 37.50% when combined with Energy features, and reached 38.56% when all features (MFCCs, Pitch, Energy) were incorporated. The overall increase in accuracy was 4.26% as prosodic features (Pitch and Energy) were added to the MFCC features. Similarly, the accuracies of other models was enhanced when MFCCs were merged with prosodic features (F0 and Energy): ANN improved from 16.31% to 20.83%, LSTM from 22.0% to 26.93%, ELM from 27.40% to 31.88%, ML-ELM from 32.29% to 37.0%, and KELM's accuracy rose from 49.70% to 55.0% with the combined features. The proposed MK-ELM model achieved an accuracy of 60.0% with only MFCCs, 62.23% when paired with Pitch features, 63.99% when combined with Energy features, and 65.5% when incorporating all features. In a second experiment, By utilizing the unique pairwise weighted scheme design, we implemented the suggested MK-ELM model in addition to existing state-of-the-art models. In this stage, various feature combinations were used to assess each model's accuracy; the outcomes are presented in Table. 3.5. Comparing the pairwise

Table 3.5: The classification accuracy for each model is assessed through the use of different feature combinations. The findings are derived using the pairwise WCS method [5].

	Accuracy achieved (%)				
Models with Weighted Scheme	MFCCs	MFCCs + Pitch	MFCCs + Energy	MFCCs + Pitch + Energy	(%) Increment due to prosodic features
SVM	65.40	67.90	68.70	69.90	4.5
ANN	37.02	39.10	40.01	41.3	4.3
LSTM	50.91	52.04	53.88	55.0	4.09
ELM	63.80	65.02	66.91	68.45	4.65
ML-ELM	68.01	70.0	71.90	73.0	4.99
KELM	72.10	75.30	76.95	77.79	5.69
MK-ELM (Proposed)	79.0	82.40	83.19	84.72	5.72

weighted system to the traditional multi-class classification technique, the models trained with the latter performed better. There was an increase in the precision of 38.56% to 69.90% for SVM, 20.83% to 41.3% for ANN, 26.93% to 55% for LSTM, 31.88% to 68.45% for ELM, 37.0% to 73.0% for ML-ELM, 55.0% to 77.79% for KELM, and 65.6% to 84.72% for MK-ELM. Overall, prosodic feature addition resulted in performance increases of about 4-5% for all systems. Notably, the suggested weighted approach produced remarkable improvements of about 20%. The suggested

MK-ELM model produced the best outcomes out of all of the models.

Table 3.6: Assessments of the accent-based effectiveness of the proposed MK-ELM model underscore the ideal blend of MFCCs and prosodic features.(%)

Accents	Tested (No. of appearance)	Predicted (No. of appearance)	Accuracy (%)	Precision	Recall	F1-score
English (A1)	26	25	96.15	0.86	0.96	0.90
Arabic (A2)	21	19	90.47	0.82	0.90	0.86
Chinese (A3)	24	21	87.50	0.84	0.87	0.85
French (A4)	14	11	78.57	0.73	0.78	0.75
Korean (A5)	16	12	75.0	1.0	0.75	0.85
Spanish (A6)	15	11	73.73	0.91	0.73	0.81
		Macro avg		0.84	0.83	0.83
		Weighted avg		0.84	0.84	0.84
		Final accuracy			**84.72**	

3.8.1 Prediction accuracy for various accents

Table 3.6 displays the accuracy and precision metrics for each accent as determined by the proposed MK-ELM model with a weighted scheme. The accuracy in recognizing a particular accent is influenced by the presence of similar accents within a region because similar accents share overlapping acoustic and phonetic features, which can simplify the model's classification task. For instance, Arabic exhibits higher accuracy compared to Spanish and French, as it includes two closely related accents (SA and UAE) that exhibit minimal variation. In contrast, French and Spanish accents originate from diverse countries with distinct linguistic and phonetic variations, making it harder for the model to generalize and recognize them accurately. The findings reveal that Accent A1 (L1 English) achieved an accuracy of 96.15%, A2 (L1 Arabic) achieved 90.47%, A3 (L1 Chinese) achieved 87.50%, A4 (L1 Korean) achieved 78.57%, A5 (L1 French) achieved 75.0%, and A6 (L1 Spanish) achieved 73.33%.

3.8.2 Model Evaluation by K-Fold Cross-Validation

A crucial element in the categorization process is evaluating a design model. The performance and capabilities of the model need to be evaluated after training. In the realm of machine learning, classification algorithms are often used to predict labels from given data. A model gains the ability to predict these unknown labels throughout the training phase. Different algorithms provide different approaches and strategies for prediction and training. However, it is imperative to confirm the efficacy of the suggested paradigm. During the validation process, the data is divided into test and train sets. Usually, 80% of the data is reserved for training, while 20% is used for testing. The model is trained in the training set and its performance is evaluated in the test set. It is an important technique because it guarantees the appropriate performance of the model and aids in the detection of overfitting.

We used a cross-validation process with varying values of K within the range (5, 10, 15, and 20) to guarantee the model's robustness and generalizability. The data set was divided into K subsets, with the model being trained and tested K times, each time using a different subset as the test set and the rest as the training set. This process continued until all subsets were used for cross-validation. Our proposed MK-ELM model showed greater stability compared to the SVM, LSTM, ANN, ELM, ML-ELM, and KELM models. Stability was assessed by creating box plots

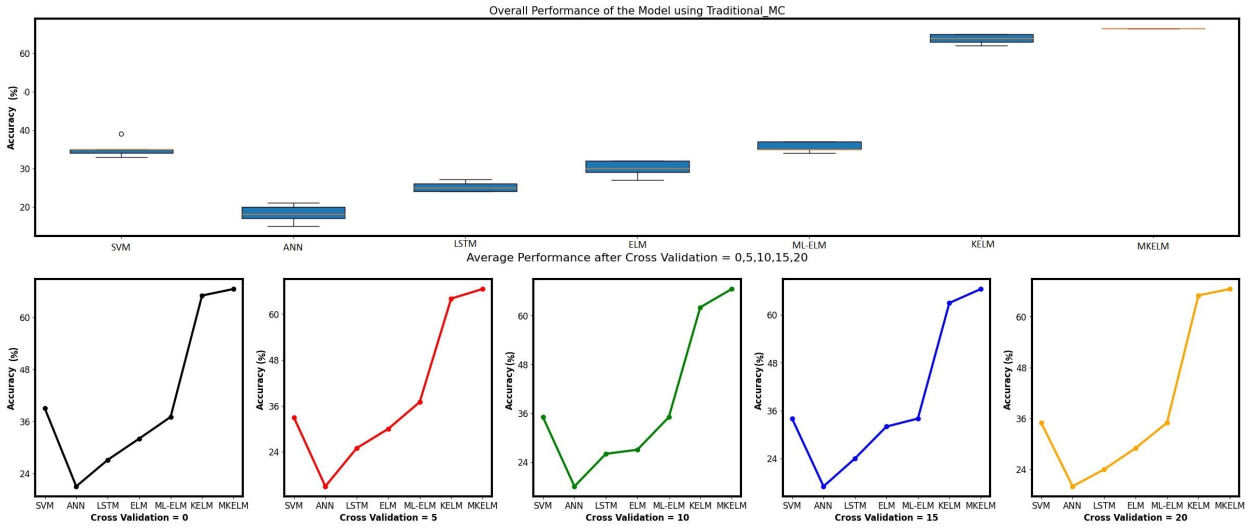


Figure 3.6: The stability of various models was assessed using the standard multiclass classification framework. For this evaluation, cross-validation was performed with iterations at different K folds values [5].

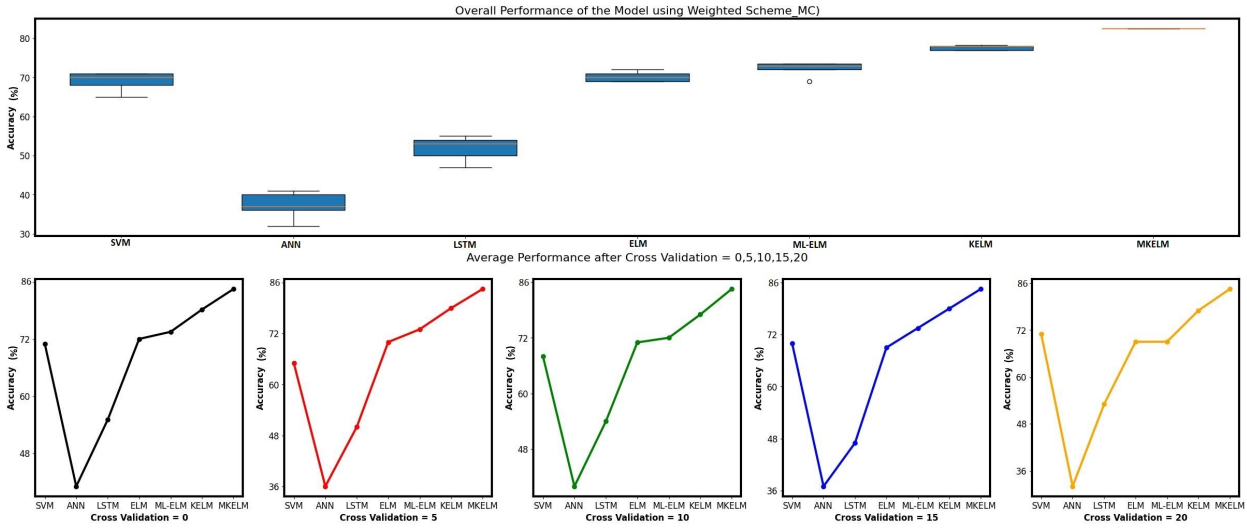


Figure 3.7: Assessing the performance stability of different models adopting a weighted multi-classification methodology. The precision observed for SVM ranged from 65% to 71%, for ANN from 32% to 41%, for LSTM from 47% to 55%, for ELM from 69% to 72%, for MLELM from 69% to 73.5%, for KELM from 77% to 78.2%, while the proposed Multiple Kernal ELM model consistently recorded an accuracy of 84.72% across all K-fold levels. [5].

that represented the "mean," "median", and "standard deviation" of the accuracies from different KFold iterations, as shown in Figures 3.6 and 3.7. Figure 3.8 illustrates the diagram of the cross-validation model. We performed a statistical analysis using K-Fold cross-validation, demonstrating

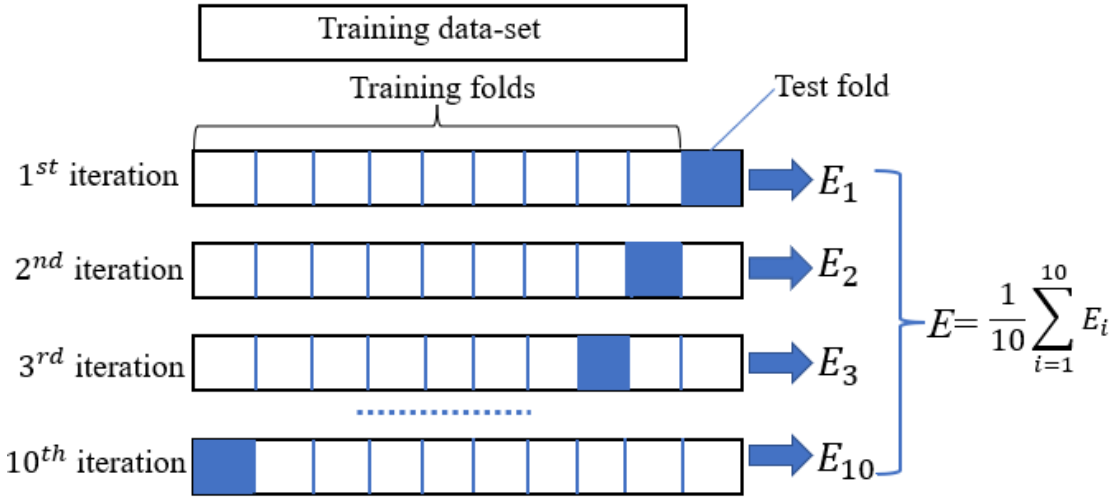


Figure 3.8: Diagram of the structure of KFold cross-validation [5].

that our proposed model surpasses other models significantly in terms of mean and accuracy, with all p-values indicating highly significant differences. The confidence intervals for our model are remarkably precise, reflecting its consistent performance over multiple iterations. These results underscore the robustness and exceptional performance of the model, especially when validated across different iterations. The zero standard deviation for MK-ELM indicates its consistent performance, unaffected by variations in K-Fold values, further reinforcing its reliability and effectiveness. The results of our statistical analysis, summarized in Table 3.7, reveal that the MK-ELM model significantly surpasses other models in mean accuracy. Paired t-tests and confidence intervals confirm that the performance differences are statistically significant. MK-ELM achieved a mean accuracy of 84.72% with a standard deviation of 0.00, demonstrating exceptional stability across different K-Fold values. In comparison, models such as SVM and ANN displayed lower mean accuracies and higher standard deviations, indicating their less consistent performance.

Table 3.7: A statistical examination of the models was performed. Paired t-tests assessed the accuracy of each model relative to the Multi Kernel KELM model [5].

Models	Mean Accuracy (%)	Standard Deviation	95% Confidence Interval	Paired t-test (vs. MK-ELM)	p-value (vs. MK-ELM)
SVM	68.00	2.10	[66.83, 69.17]	23.96	< 0.001
ANN	36.50	3.10	[34.52, 38.48]	45.14	< 0.001
LSTM	51.00	2.90	[49.17, 52.83]	36.79	< 0.001
ELM	70.50	1.50	[69.60, 71.40]	24.64	< 0.001
ML-ELM	71.75	2.10	[70.57, 72.93]	25.34	< 0.001
KELM	77.60	0.80	77.17, 78.03]	31.77	< 0.001
MK-ELM	84.72	0.00	[84.72, 84.72]	-	-

The confusion matrix for the proposed system is generated when employing both traditional multi-classification and weighted scheme-based multi-classification with the MK-ELM model. Figure 3.9 illustrates the confusion matrix obtained from multi-classification using our proposed weighted scheme with the MK-ELM model.

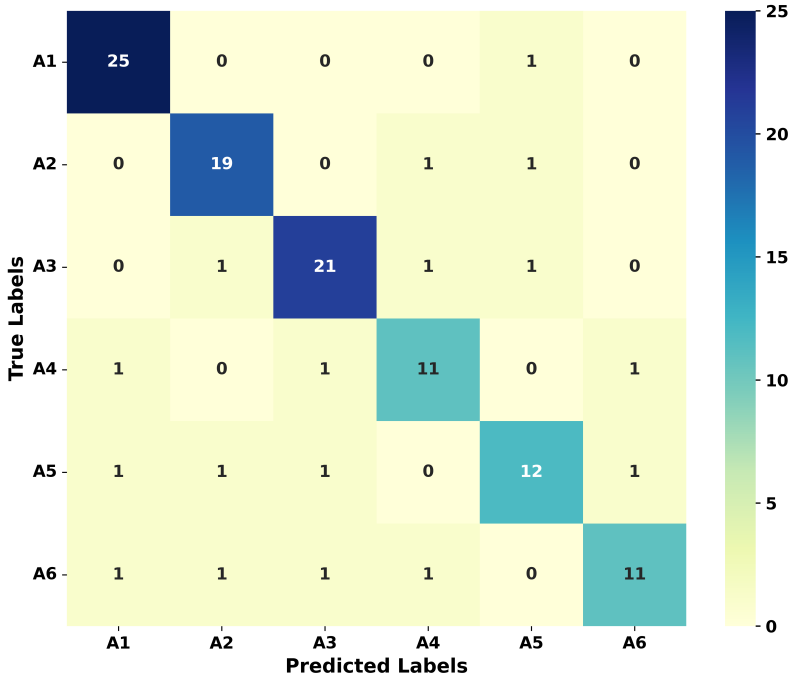


Figure 3.9: Confusion matrix demonstrating the frequency distribution of predicted classes/accents as compared to the actual classes [5].

3.8.3 Evaluation of Model Performance with Respect to Computational Time

To evaluate their training times, the proposed model was compared with other different machine learning and deep learning-based model architectures. We used two methods in our work, Initially, there is the conventional multiclass classification and then the weighted classification scheme (WCS) based approach. In the course of model implementation, the training durations for each model using the traditional method are as follows: SVM requires 740 seconds, ANN 1540 seconds, LSTM 1620 seconds, ELM 500 seconds, ML-ELM 510 seconds, KELM 495 seconds, and our proposed model takes 490 seconds. On the other hand, the WCS-based approach took 745 seconds for the proposed model, 1780 seconds for the ANN, 1767 seconds for the LSTM, 768 seconds for the ELM, 780 seconds for the ML-ELM, and 760 seconds for the KELM in the training phases. The reduced training times in these data suggest that the proposed MK-ELM model is more computationally efficient than the other models. These findings are represented graphically in Figures 3.10 and 3.11.

3.8.4 Comparison to Earlier Findings

Using the same GMU data set, we compared the MK-ELM model to previous research on the identification of accents (AID). Table 3.8 shows the findings of these previous investigations. In [66], Bryant et al. used a GMM to classify five distinct speech accents with 42% accuracyusing the speech data from the GMU Speech Accent Archive. A CNN-based approach was used by Widyowaty et al. [67] and Y. Singh et al. [68] to classify five foreign accents, with overall accuracy of 53.92% and 51.96%, respectively. Widyowaty et al. introduced a KNN model in [69], which

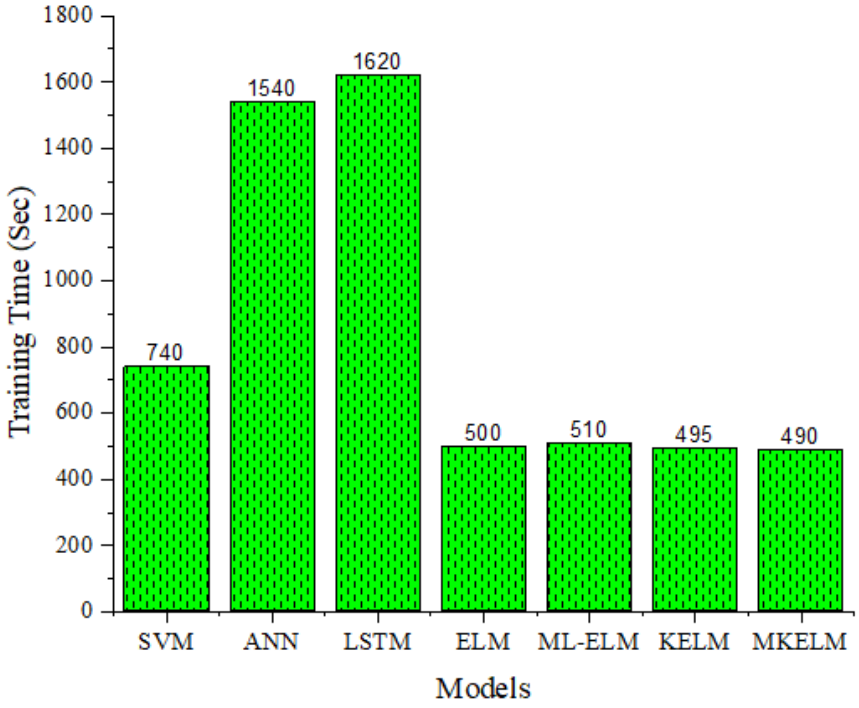


Figure 3.10: The training times (in seconds) for each model when employing standard multi-class classification indicate that the proposed MK-ELM model exhibits the briefest training period, completing in merely 490 seconds[5].

recognized six accents with 57% accuracy. A DNN was utilized by Ensslin et al. [70] to categorize three accents with 61% accuracy. Parikh et al. [71] identified three accents with 68.67% accuracy by combining CNN and LSTM techniques. With a 2-layer CNN model, Berjon et al. [72] were able to categorize five accents with an accuracy of 70.65%. A DBN model was created by Upadhyay et al. [62], and it identified six accents with a 71.09% accuracy rate. A Gaussian Mixture Model (GMM) was created by Deshpande et al. [96], and it successfully distinguished two accents with an accuracy rating of 85.00%. With an accuracy of 53.92%, Singh et al. [68] distinguished five accents using a Convolutional Neural Network (CNN). CNN was also used by Duduka et al. [97], who identified three accents with 62.81% accuracy. In order to recognize eleven accents, Jiao et al. [60] used Deep Neural Network based approaches, resulting in an accuracy of 46.66%. A hybrid technique combining CNN, RNN, and DNN was presented by Parikh et al. [71], which successfully identified three accents with an accuracy of 68.67%. Finally, Ahmed et al. [98] identified three accents with 70.33% accuracy using a CNN model. It is found that the CNN-based approach for foreign accent identification outperforms our "Multi-Kernel Extreme Learning Machine (MK-ELM)" model in terms of accuracy when Mel-spectrogram features are used [95]. However, a thorough examination highlights a number of elements. The CNN model employs amplitude Mel-spectrograms to successfully capture a variety of characteristics, whereas MK-ELM combines prosodic features with MFCCs to indicate potential improvements through inventive prosodic feature utilization. In particular, for several European accents, CNN's use of the Speech Accent Archive dataset greatly increases its efficacy. On the other hand, a closer examination of the representativeness and vari-

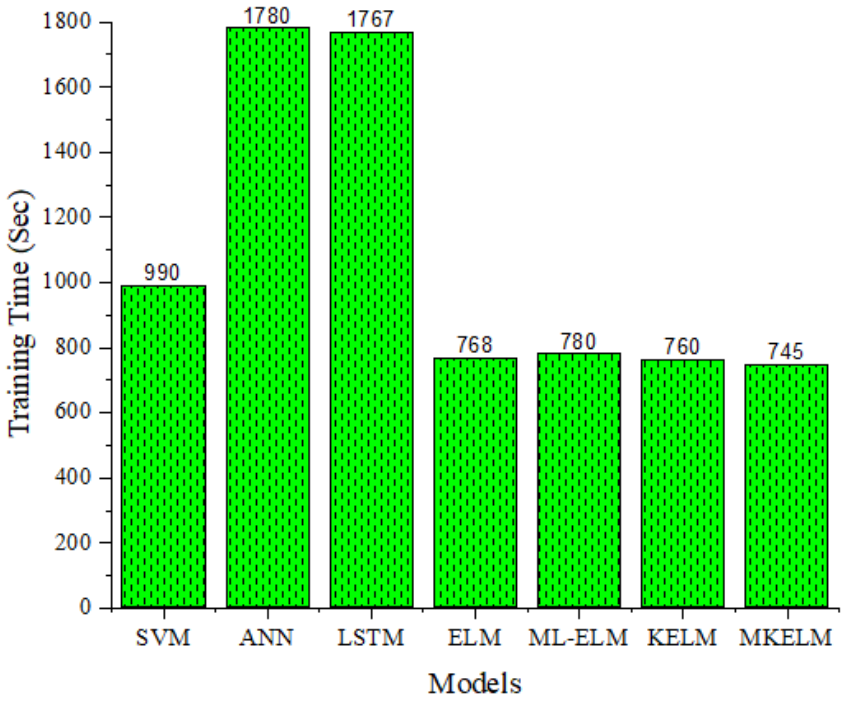


Figure 3.11: Time duration (seconds) for each model training using the WCS in multi-class classification. The presented MK-ELM model records the shortest training time at 745 seconds [5].

ety of MK-ELM dataset may highlight opportunities for improvement. The proposed model has distinct practical advantages for real-world applications with its lower computational complexity, faster learning rates, and shorter training times. For environments with limited resources, its scalability and efficiency are essential. While CNN is good at identifying certain accents, MK-ELM's paired weighting process corrects imbalances in the class, indicating a better chance of generalization over a larger range of accents. Future research seems to be headed in a promising direction: the development of hybrid models that combine the best features of both the MK-ELM and CNN architectures. This approach might lead to the development of a more reliable and accurate system for recognizing foreign accents. The practical relevance and flexibility of these models depend on evaluating them in real-world scenarios, taking into account criteria that go beyond accuracy, and continuously improving them on the basis of changing datasets. In the end, our investigation showed that while CNN was more accurate, MK-ELM had its own advantages, particularly when it comes to its attention to prosodic details, operational effectiveness, generalization capacity, and adaptability. The promise of MK-ELM in increasing foreign accent identification is highlighted by the results of continuous research and development of hybrid techniques. Using a pairwise weighted technique and the MK-ELM model, our study achieved 84.72% accuracy across 6 different dialects, yielding superior results for multi-class categorization.

3.8.5 Constraints of the Study

This study has a number of restrictions. There may be a need for larger and more varied data sets because the GMU data set may notadequately reflect the global variance in accents. To improve

Table 3.8: A comparative study of different foreign accent analyses using the same GMU dataset [5].

Study on Foreign Accents Identification	Data-set	Models	Accents Identified	Accuracy achieved (%)
M Bryant et.al [66]	Speech Accent Archive (GMU)	GDA and Naive Bayes	5	42.00
DS Widyowaty et.al [67]	Speech Accent Archive (GMU)	CNN	5	51.96
Y Singh et.al[68]	Speech Accent Archive (GMU)	CNN	5	53.92
DS Widyowaty et.al[69]	Speech Accent Archive (GMU)	KNN	6	57.00
A Ensslin et.al [70]	Speech Accent Archive (GMU)	CNN	3	61.00
P Parikh et.al[71]	Speech Accent Archive (GMU)	CNN, DNN,RNN	3	68.67
e P Berjon et.al[72]	Speech Accent Archive (GMU)	2-layer CNN	5	70.65
V Mikhailava et.al[95]	Speech Accent Archive (GMU)	CNN	9	98.70
S Deshpande et.al[96]	Speech Accent Archive (GMU)	GMM	2	85.00
Y Singh et.al[68]	Speech Accent Archive (GMU)	CNN	5	53.92
Duduka et.al[97]	Speech Accent Archive (GMU)	CNN	3	62.81
Y Jiao et.al[60]	Speech Accent Archive (GMU)	DNN RNN	11	46.66
P Parikh et.al[71]	Speech Accent Archive (GMU)	DNN RNN CNN	3	68.67
A Ahmed et.al[98]	Speech Accent Archive (GMU)	CNN	3	70.33
Current study	Speech Accent Archive (GMU)	MK-ELM (Proposed)	6	84.72

performance, further feature sets like deep learning-based embeddings and Mel spectrograms at the word and phoneme levels could be added. In this study, the initial feature sets used were prosodic features (e.g., pitch, energy) alongside MFCCs. More resilience and noise handling skills are required because the audio samples that were tested in this study do not reflect the range of noise levels in real-scenario recordings. The accuracy of the model varies with accent; certain accents, such as native French and native Spanish, have lower accuracy than others, which indicates that feature extraction and model training still need to be improved. By addressing these issues, the robustness, accuracy, and applicability of the MK-ELM model can be improved.

3.9 Conclusion

This chapter describes a CNN-RNN hybrid model developed for automatically detecting vowel landmarks, achieving a 74.98% accuracy rate with the LaMIT dataset. The results demonstrate the proficiency of advanced deep learning methods for landmark detection. Additionally, the chapter introduces an innovative approach for multi-classification of foreign accents employing a pairwise weighted strategy alongside the MK-ELM model. The proposed model incorporates prosodic features and MFCCs as inputs. We compared its effectiveness with top models like LSTM, ANN, SVM, ELM, and kernel-based ELM. Experimental findings show that our model's weighted scheme successfully differentiates accents and delivers higher accuracy with reduced computational complexity compared to the others tested.

Chapter 4

Conclusions and Future Work

4.1 Conclusions

The first focus of this thesis was to present a modernization of the xkl software, originally developed by Dennis Klatt at MIT in the 1980s. This software has historically been instrumental in digital speech processing and spectrum estimation, but its usability has been hindered by outdated Motif-based UI libraries. The development introduced by this work of a new Graphical User Interface (GUI), using GTK libraries, marked a pivotal improvement of the tool. This modern GUI not only simplified the installation process but most importantly made the software accessible and user-friendly on various platforms, including Windows, Linux, and MacOS.

The xkl refurbishment also addressed the inclusion, in the spectrum processing tools, of an analysis method, the reassigned spectrogram, that in recent years has proved to be important for fine-grained formant estimation in vowels [99], [100]. The incorporation of this advanced spectrum analysis tool also allows for a precise and detailed examination of speech dynamics. As an additional minor but useful improvement, the possibility of saving formant values automatically in a text file greatly reduced the manual effort involved in speech analysis. This development facilitates large-scale analysis, especially for vowels studies, making xkl a powerful tool for researchers.

The development of a modern xkl speech analysis tool is part of a broader research project, the LaMIT project [1], that has the goal of applying Stevens' lexical access model [2] to the Italian language. One major innovation introduced by Stevens is the concept of landmark, that is, the presence of privileged regions in the time domain, the landmark positions at which a primary phase of the perceptual process would take place, the landmark positions. In this work, an automatic vowel landmark detector was developed. This landmark recognition system was developed and implemented based on a Convolutional Neural Network combined with a Recurrent Neural Network, i.e. a CNN-RNN hybrid model. The recognizer was tested on sentences of the LaMIT database [3], a corpus formed by 800 spoken utterances (4 native italian speakers) that was specifically created within the LaMIT project. The sentences of the database were manually analyzed by examining the corresponding speech waveforms, but most importantly using the xkl tool that provided invaluable information on both general and time-varying spectral properties. It is thanks to this analysis that the 800 sentences could be manually labeled and that the corpus contains now all the information about landmark presence, landmark type, and landmark position in time. The CNN-RNN recognizer used a set of parameters that combined energy measurements and mel spectrum

descriptors, and was run on the above sentences. Its output produced an estimation of detected vowel landmarks, and this output was compared against the manually estimated vowel landmark presence. The overall recognition rate was 74.98%. For individual speakers the recognition rate ranged from about 72% to about 77%.

Further motivated by a strong interest in artificial intelligence methods, this research investigation also covered a particularly challenging problem in current AI based speech recognition systems, that is recently receiving a great deal of attention:automatic foreign accent identification [4]. In this work, we proposed a Multi-Kernel Extreme Learning Machine (MK-ELM) model, along with a weighted scheme, for application to the recognition of 5 different accents (Arabic, Chinese, Korean, Spanish, French) in American English. The recognition was based on Mel-frequency cepstral coefficients (MFCC) and prosodic features (Pitch, Energy). To evaluate the effectiveness of our model, we compared it with state-of-the-art methods, including Support Vector Machine (SVM), Artificial Neural Network (ANN), Long Short-Term Memory (LSTM), Extreme Learning Machine (ELM), Multi-Layer Extreme Learning Machine (MLELM), Kernel Extreme Learning Machine (KELM). Experimental analyses demonstrated that the MK-ELM model with the weighted classifier effectively distinguishes between different foreign accents. It outperformed all other tested models in terms of accuracy and computational complexity. The proposed model achieves an accuracy rate of 84.72% using a paired weighting scheme. In contrast, the accuracy rate drops to 66.5% when employing the traditional non-weighted multi-classification scheme. A comparison with other models demonstrates the significant advantages of the proposed model in (AIFA) multi-class classification, showcasing improved accuracy, reduced computational complexity and enhanced stability compared to state-of-the-art classification methods.

4.2 Future Work

Although significant advances have been made in the modernization of the xkl software, several areas warrant further research and development. Future research should focus on upgrading xkl to incorporate tools for structured annotation of labeling into multiple tiers. Currently, xkl only permits the insertion of phonetic symbols directly on the acoustic waveform plot, which limits its utility in modern speech processing applications that require systematic annotation. For example, the Praat tool offers tiered labeling and is widely used for such purposes. To streamline this process and enhance the functionality of xkl, it is crucial to develop features that support the insertion and management of multiple labeling levels, thus facilitating more efficient and organized annotation.

Another important direction of future work is the refinement and expansion of deep learning models for automatic landmark detection. The proposed CNN-RNN hybrid model developed for automatic vowel landmark detection showed promising results, but further refinement and exploration of other advanced architectures, such as transformers, could yield better performance. Expanding the training and validation datasets to include more diverse and larger samples is also crucial to improve the robustness of these models. Developing gender-specific models, as suggested by the distinct patterns observed in the current study, could further improve the precision of speech analysis. Tailoring models to account for gender-specific variations in speech could lead to more precise analysis and better outcomes in speech processing tasks.

Future work in Automatic Identification of Foreign Accents (AIFA) must recognize that vari-

ations in individual speaker characteristics can affect accent formation. Achieving a perfect classification of foreign accents may be unrealistic for a specific set of speakers or accents due to the inconsistencies in accent-sensitive features in their speech signals. Our plans for future research include addressing several key issues. Firstly, we aim to enhance classification accuracy by segmenting at the word or phoneme level instead of at the sentence level. Additionally, we will assess the proposed model on larger datasets to verify its generalization. Furthermore, we will explore the effectiveness of using multi-resolution features that combine both long- and short-term aspects and consider the integration of formant position shifts. These efforts will contribute to the ongoing advancement of the field of automatic accents and dialect identification [101].

Bibliography

[1] Maria-Gabriella Di Benedetto, Stefanie Shattuck-Hufnagel, Jeung-Yoon Choi, Luca De Nardis, Javier Arango, Ian Chan, and Alec DeCaprio. Lexical access model for italian – modeling human speech processing: identification of words in running speech toward lexical access based on the detection of landmarks and other acoustic cues to features, 2021. URL `https://arxiv.org/abs/2107.02720`.

[2] Kenneth N Stevens. Toward a model for lexical access based on acoustic landmarks and distinctive features. *The Journal of the Acoustical Society of America*, 111(4):1872–1891, 2002.

[3] Maria-Gabriella Di Benedetto, Stefanie Shattuck-Hufnagel, Jeung-Yoon Choi, Luca De Nardis, Javier Arango, Ian Chan, Alec DeCaprio, and Sara Budoni. The lamit database: A read speech corpus for acoustic studies of the italian language toward lexical access based on the detection of landmarks and other acoustic cues to features. *Data in brief*, 42:108275, 2022.

[4] A Kolesnikova and V Frolova. The impact of foreign accent on professional communication: Empirical research. *Norwegian Journal of development of the International Science No*, 118: 60, 2023.

[5] Kaleem Kashif, Abeer Alwan, Yizhi Wu, Luca De Nardis, and Maria-Gabriella Di Benedetto. Mkelm based multi-classification model for foreign accent identification. *Heliyon*, 10(16), 2024.

[6] Steven Weinberger. Speech accent archive. george mason university. *Online:¡ http://accent. gmu. edu*, 2015.

[7] Paul Boersma. Praat: doing phonetics by computer. *http://www. praat. org/*, 2007.

[8] Kåre Sjölander and Jonas Beskow. Wavesurfer-an open source speech tool. In *Sixth International Conference on Spoken Language Processing*, 2000.

[9] Frederick Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556, 1976.

[10] Li Deng and Xiao Li. Machine learning paradigms for speech recognition: An overview. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(5):1060–1089, 2013.

[11] Roberto Pieraccini. *AI assistants*. MIT Press, 2021.

[12] Kenneth N Stevens. *Acoustic phonetics*, volume 30. MIT press, 2000.

[13] Nick G Riches, Tom Loucas, Gillian Baird, Tony Charman, and Emily Simonoff. Non-word repetition in adolescents with specific language impairment and autism plus language impairments: A qualitative analysis. *Journal of communication disorders*, 44(1):23–36, 2011.

[14] Lawrence D Shriberg, Joan Kwiatkowski, and Amanda M Best. Nonword repetition in children with speech sound disorders. *Clinical Linguistics & Phonetics*, 23(12):926–947, 2009.

[15] Fiona E Gibbon and Alice Lee. Preface to the special issue on covert contrasts, 2017.

[16] Luca De Nardis, Maria-Gabriella Di Benedetto, Jeung-Yoon Choi, and Stefanie Shattuck-Hufnagel. xkl: A legacy software for detailed acoustic analysis of speech made modern. *SoftwareX*, 23:101492, 2023.

[17] William Labov. A sociolinguistic perspective on sociophonetic research. *Journal of phonetics*, 34(4):500–515, 2006.

[18] Yen-Liang Shue, Patricia Keating, Chad Vicenik, and Kristine Yu. Voicesauce. *p. Program available online at http://www. seas. ucla. edu/spapl/voicesauce/. UCLA*, 2009.

[19] Sheila E Blumstein and Kenneth N Stevens. Acoustic invariance in speech production: Evidence from measurements of the spectral characteristics of stop consonants. *The Journal of the Acoustical Society of America*, 66(4):1001–1017, 1979.

[20] David B Pisoni and Robert Ellis Remez. *The handbook of speech perception*. Wiley Online Library, 2005.

[21] Klsyn - c language port, 2023. URL `https://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/speech/systems/klatt/0.html`. Released on May 9, 1994.

[22] KlSyn - Python 3 language port. Version released on october 7, 2021. `https://github.com/rsprouse/klsyn`, 2023. Accessed on 30 June 2023.

[23] KlSyn - JavaScript language port. Version released on december 14, 2022. `https://github.com/chdh/klatt-syn`, 2023. Accessed on 30 June 2023.

[24] Di He, Boon Pang Lim, Xuesong Yang, Mark Hasegawa-Johnson, and Deming Chen. Acoustic landmarks contain more information about the phone string than other frames for automatic speech recognition with deep neural network acoustic model. *The Journal of the Acoustical Society of America*, 143(6):3207–3219, 2018.

[25] M.-G. Di Benedetto and L. De Nardis. Consonant gemination in italian: The nasal and liquid case. *Speech Communication*, 133:62–80, 2021. doi: 10.1016/j.specom.2021.07.006.

[26] M.-G. Di Benedetto and L. De Nardis. Consonant gemination in italian: The affricate and fricative case. *Speech Communication*, 134:86–108, 2021. doi: 10.1016/j.specom.2021.07.005.

[27] M.-G. Di Benedetto, S. Shattuck-Hufnagel, L. De Nardis, S. Budoni, J. Arango, I. Chan, and A. DeCaprio. Lexical and syntactic gemination in italian consonants—does a geminate

italian consonant consist of a repeated or a strengthened consonant? *Journal of the Acoustical Society of America*, 149(5):3375–3386, 2021. doi: 10.1121/10.0004987.

[28] T. Igeta and T. Arai. Dominance of lower formants of korean vowels /o/–/u/ in perceptual identification by seoul dialect listeners. *Acoustical Science and Technology*, 40:56–58, 2019. doi: 10.1250/ast.40.56.

[29] T. Igeta, M. Sonu, and T. Arai. Sound change of /o/ in modern seoul korean: Focused on relations with acoustic characteristics and perception. *Phonetics and Speech Sciences*, 6(3): 109–119, 2014. doi: 10.13064/KSSS.2014.6.3.109.

[30] K. Tomaru and T. Arai. Discrimination of /ra/ and /la/ speech continuum by native speakers of english under nonisolated conditions. *Acoustical Science and Technology*, 35(5):251–259, 2014. doi: 10.1250/ast.35.251.

[31] Maria-Gabriella Di Benedetto and Luca De Nardis. The gemma speech database: Vcv and vccv words for the acoustic analysis of consonants and lexical gemination in italian. *Data in Brief*, 43:108373, 2022.

[32] Maria-Gabriella Di Benedetto, Stefanie Shattuck-Hufnagel, Luca De Nardis, Sara Budoni, Javier Arango, Ian Chan, and Alec DeCaprio. Lexical and syntactic gemination in italian consonants—does a geminate italian consonant consist of a repeated or a strengthened consonant? *The Journal of the Acoustical Society of America*, 149(5):3375–3386, 2021.

[33] Kenneth N Stevens. Phonetic features and lexical access. In *Recent Research Towards Advanced Man-Machine Interface Through Spoken Language*, pages 267–281. Elsevier, 1996.

[34] Sharlene A Liu. Landmark detection for distinctive feature-based speech recognition. *The Journal of the Acoustical Society of America*, 100(5):3417–3430, 1996.

[35] Ruizhi Li, Xiaofei Wang, Sri Harish Mallidi, Shinji Watanabe, Takaaki Hori, and Hynek Hermansky. Multi-stream end-to-end speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:646–655, 2019.

[36] John HL Hansen and Levent M Arslan. Foreign accent classification using source generator based prosodic features. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 836–839. IEEE, 1995.

[37] V Gupta and P Mermelstein. Effects of speaker accent on the performance of a speaker-independent, isolated-word recognizer. *The Journal of the Acoustical Society of America*, 71 (6):1581–1587, 1982.

[38] Silke Goronzy, Stefan Rapp, and Ralf Kompe. Generating non-native pronunciation variants for lexicon adaptation. *Speech Communication*, 42(1):109–123, 2004.

[39] Levent M Arslan and John HL Hansen. Language accent classification in american english. *Speech Communication*, 18(4):353–367, 1996.

[40] Pongtep Angkititrakul and John HL Hansen. Advances in phone-based modeling for automatic accent classification. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(2):634–646, 2006.

[41] Hamid Behravan, Ville Hautamäki, and Tomi Kinnunen. Factors affecting i-vector based foreign accent recognition: A case study in spoken finnish. *Speech Communication*, 66:118–129, 2015.

[42] Cécile Woehrling and Philippe Boula de Mareüil. Identification of regional accents in french: perception and categorization. In *Ninth International Conference on Spoken Language Processing*, 2006.

[43] Adrian Leemann. *Comparative analysis of voice fundamental frequency behavior of four swiss german dialects*. Selbstverl., 2009.

[44] Cynthia G Clopper, David B Pisoni, and Kenneth De Jong. Acoustic characteristics of the vowel systems of six regional varieties of american english. *The Journal of the Acoustical society of America*, 118(3):1661–1676, 2005.

[45] Larry M Hyman. In defense of prosodic typology: A response to beckman and venditti. *Linguistic Typology*, 16(3):341–385, 2012.

[46] Helene N Andreassen and Isabelle Racine. Schwa et variation inter-régionale: une analyse de trois points d'enquête suisses. *Journées PFC 2013 «Regards croisés sur les corpus oraux*, 2013.

[47] Peter Rickard. R. lodge anthony, french: from dialect to standard. london and new york: Routledge1993, x+ 285 pp. 0 415 08071 1. *Journal of French Language Studies*, 3(2):243–244, 1993.

[48] John Nerbonne. Linguistic variation and computation (invited talk). In *10th conference of the european chapter of the association for computational linguistics*, 2003.

[49] Kaleem Kashif, Yizhi Wu, and Adjeisah Michael. Consonant phoneme based extreme learning machine (elm) recognition model for foreign accent identification. In *Proceedings of the 2019 The World Symposium on Software Engineering*, pages 68–72, 2019.

[50] Hong You, Abeer Alwan, Abe Kazemzadeh, and Shrikanth Narayanan. Pronunciation variations of spanish-accented english spoken by young children. In *Ninth European Conference on Speech Communication and Technology*, 2005.

[51] James E Flege, Carlo Schirru, and Ian RA MacKay. Interaction between the native and second language phonetic subsystems. *Speech communication*, 40(4):467–491, 2003.

[52] Ghinwa Choueiter, Geoffrey Zweig, and Patrick Nguyen. An empirical study of automatic accent classification. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4265–4268. IEEE, 2008.

[53] Liu Wai Kat and Pascale Fung. Fast accent identification and accented speech recognition. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*, volume 1, pages 221–224. IEEE, 1999.

[54] Khomdet Phapatanaburi, Longbiao Wang, Ryota Sakagami, Zhaofeng Zhang, Ximin Li, and Masahiro Iwahashi. Distant-talking accent recognition by combining gmm and dnn. *Multimedia tools and applications*, 75(9):5109–5124, 2016.

[55] Karsten Kumpf and Robin W King. Automatic accent classification of foreign accented australian english speech. In *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP'96*, volume 3, pages 1740–1743. IEEE, 1996.

[56] Dominique Fohr and Irina Illina. Text-independent foreign accent classification using statistical methods. In *2007 IEEE International Conference on Signal Processing and Communications*, pages 812–815. IEEE, 2007.

[57] Mohamad Hasan Bahari, Rahim Saeidi, David Van Leeuwen, et al. Accent recognition using i-vector, gaussian mean supervector and gaussian posterior probability supervector for spontaneous telephone speech. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 7344–7348. IEEE, 2013.

[58] Hamid Behravan, Ville Hautamäki, Sabato Marco Siniscalchi, Tomi Kinnunen, and Chin-Hui Lee. i-vector modeling of speech attributes for automatic foreign accent recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(1):29–41, 2016. doi: 10.1109/TASLP.2015.2489558.

[59] Leon Mak An Sheng and Mok Wei Xiong Edmund. Deep learning approach to accent classification. *CS229*, 2017.

[60] Yishan Jiao, Ming Tu, Visar Berisha, and Julie M Liss. Accent identification by combining deep neural networks and recurrent neural networks trained on long and short term features. In *Interspeech*, pages 2388–2392, 2016.

[61] Archana Purwar, Hardik Sharma, Yukti Sharma, Himanshu Gupta, and Amanpreet Kaur. Accent classification using machine learning and deep learning models. In *2022 1st International Conference on Informatics (ICI)*, pages 13–18. IEEE, 2022.

[62] Rishabh Upadhyay and Simon Lui. Foreign english accent classification using deep belief networks. In *2018 IEEE 12th international conference on semantic computing (ICSC)*, pages 290–293. IEEE, 2018.

[63] Too Chen, Chao Huang, Eric Chang, and Jingehan Wang. Automatic accent identification using gaussian mixture models. In *IEEE Workshop on Automatic Speech Recognition and Understanding, 2001. ASRU'01.*, pages 343–346. IEEE, 2001.

[64] Muhammad Rizwan, Babafemi O Odelowo, and David V Anderson. Word based dialect classification using extreme learning machines. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 2625–2629. IEEE, 2016.

[65] Felix Weninger, Yang Sun, Junho Park, Daniel Willett, and Puming Zhan. Deep learning based mandarin accent identification for accent robust asr. In *INTERSPEECH*, pages 510–514, 2019.

[66] Morgan Bryant, Amanda Chow, and Sydney Li. Classification of accents of english speakers by native language, 2014.

[67] Dwi Sari Widyowaty, Andi Sunyoto, and Hanif Al Fatta. Accent recognition using mel-frequency cepstral coefficients and convolutional neural network. In *Proceedings of the International Conference on Innovation in Science and Technology (ICIST 2020)*, pages 43–46. Atlantis Press, 2021. ISBN 978-94-6239-472-8. doi: https://doi.org/10.2991/aer.k.211129.010. URL `https://doi.org/10.2991/aer.k.211129.010`.

[68] Yuvika Singh, Anban Pillay, and Edgar Jembere. Features of speech audio for accent recognition. In *2020 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*, pages 1–6. IEEE, 2020.

[69] Dwi Sari Widyowaty and Andi Sunyoto. Accent recognition by native language using mel-frequency cepstral coefficient and k-nearest neighbor. In *2020 3rd International Conference on Information and Communications Technology (ICOIACT)*, pages 314–318. IEEE, 2020.

[70] Astrid Ensslin, Tejasvi Goorimoorthee, Shelby Carleton, Vadim Bulitko, and Sergio Poo Hernandez. Deep learning for speech accent detection in videogames. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.

[71] Pratik Parikh, Ketaki Velhal, Sanika Potdar, Aayushi Sikligar, and Ruhina Karani. English language accent classification and conversion using machine learning. In *Proceedings of the International Conference on Innovative Computing & Communications (ICICC)*, 2020.

[72] Pierre Berjon, Avishek Nag, and Soumyabrata Dev. Analysis of french phonetic idiosyncrasies for accent recognition. *Soft Computing Letters*, 3:100018, 2021.

[73] Yu Zhang, Yu Wang, Guoxu Zhou, Jing Jin, Bei Wang, Xingyu Wang, and Andrzej Cichocki. Multi-kernel extreme learning machine for eeg classification in brain-computer interfaces. *Expert Systems with Applications*, 96:302–310, 2018.

[74] Haoran Zhao and Sen Guo. Uncertain interval forecasting for combined electricity-heat-cooling-gas loads in the integrated energy system based on multi-task learning and multi-kernel extreme learning machine. *Mathematics*, 9(14):1645, 2021.

[75] Bhawna Ahuja and Virendra P Vishwakarma. Deterministic multi-kernel based extreme learning machine for pattern classification. *Expert Systems with Applications*, 183:115308, 2021.

[76] Motif libraries. Motif libraries, 2023. released on December 5, 2017, available at `https://sourceforge.net/projects/motif/`. [Accessed on 30 June 2023].

[77] XKL Licensing. Xkl licensing page. `http://newyork.ing.uniroma1.it/xkl.php`, n.d. Accessed: [insert date].

[78] S. A. Fulop and K. Fitz. Algorithms for computing the time-corrected instantaneous frequency (reassigned) spectrogram, with applications. *Journal of the Acoustical Society of America*, 119(1):360–371, 2006.

[79] Node 74: The users guide. `https://www.mi.uni-koeln.de/b/tcm/doc/usersguide/node74.html`, 2024. Accessed: 2024-08-22.

[80] Motif 2.3.4 release notes. `https://motif.ics.com/motif-234-release-notes`, 2023. Accessed: 2024-08-22.

[81] Gtk. `https://www.gtk.org`, . Accessed: 2024-08-01.

[82] Gtk features. `https://www.gtk.org/features/`, 2024. Accessed: 2024-08-22.

[83] Gtk documentation. `https://docs.gtk.org`, . Accessed: 2024-08-01.

[84] Gnome developer center. `https://developer.gnome.org`. Accessed: 2024-08-01.

[85] GTK Team. Gtk - installation on windows. `https://www.gtk.org/docs/installations/windows/`, 2023. Accessed on 7 September 2024.

[86] Inc. Kitware. Cmake: Download. `https://cmake.org/download/`, 2024. Accessed on 7 September 2024.

[87] Arda Ustubioglu, Beste Ustubioglu, and Guzin Ulutas. Mel spectrogram-based audio forgery detection using cnn. *Signal, Image and Video Processing*, 17(5):2211–2219, 2023.

[88] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[89] Guang-Bin Huang, Hongming Zhou, Xiaojian Ding, and Rui Zhang. Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):513–529, 2011.

[90] Mark Girolami. Mercer kernel-based clustering in feature space. *IEEE transactions on neural networks*, 13(3):780–784, 2002.

[91] James Lyons, Darren Yow-Bang Wang, Gianluca, Hanan Shteingart, Erik Mavrinac, Yash Gaurkar, Watcharapol Watcharawisetkul, Sam Birch, Lu Zhihe, Josef Hölzl, Janis Lesinskis, Henrik Almér, Chris Lord, and Adam Stark. jameslyons/python_speech_features: release v0.6.1, January 2020. URL `https://doi.org/10.5281/zenodo.3607820`.

[92] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, et al. The htk book. *Available on*, 1999.

[93] Najim Dehak, Pierre Dumouchel, and Patrick Kenny. Modeling prosodic features with joint factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(7):2095–2103, 2007.

[94] Lawrence Rabiner. On the use of autocorrelation analysis for pitch detection. *IEEE transactions on acoustics, speech, and signal processing*, 25(1):24–33, 1977.

[95] Veranika Mikhailava, Mariia Lesnichaia, Natalia Bogach, Iurii Lezhenin, John Blake, and Evgeny Pyshkin. Language accent detection with cnn using sparse data from a crowd-sourced speech archive. *Mathematics*, 10(16):2913, 2022.

[96] Shamalee Deshpande, Sharat Chikkerur, and Venu Govindaraju. Accent classification in speech. In *Fourth IEEE Workshop on Automatic Identification Advanced Technologies (AutoID'05)*, pages 139–143. IEEE, 2005.

[97] Saiprasad Duduka, Henil Jain, Virik Jain, Harsh Prabhu, and Pramila M Chawan. A neural network approach to accent classification. *International Research Journal of Engineering and Technology (IRJET)*, 8(03):1175–1177, 2021.

[98] Asad Ahmed, Pratham Tangri, Anirban Panda, Dhruv Ramani, and Samarjit Karmakar. Vfnet: A convolutional architecture for accent classification. In *2019 IEEE 16th India Council International Conference (INDICON)*, pages 1–4. IEEE, 2019.

[99] Sean A Fulop and Kelly Fitz. A spectrogram for the twenty-first century. *Acoustics today*, 2 (3):26–33, 2006.

[100] Christine H Shadle, Hosung Nam, and DH Whalen. Comparing measurement errors for formants in synthetic and natural vowels. *The Journal of the Acoustical Society of America*, 139(2):713–727, 2016.

[101] Alexander Johnson, Kevin Everson, Vijay Ravi, Anissa Gladney, Mari Ostendorf, and Abeer Alwan. Automatic Dialect Density Estimation for African American English. In *Proc. Interspeech 2022*, pages 1283–1287, 2022. doi: 10.21437/Interspeech.2022-796.

List of Publications

- **Kashif, K.**, Alwan, A., Wu, Y., De Nardis, L., & Di Benedetto, M. (2024). *MKELM based multi-classification model for foreign accent identification.* Heliyon, e36460. `https://doi.org/10.1016/j.heliyon.2024.e36460`

- Di Benedetto, M.-G., De Nardis, L., & **Kashif, K**. Toward understanding time patterns of landmark acoustic cues: A database of time distances between consecutive landmarks. (to be submitted in Data Journal)

- De Nardis, L., La Morgia, M., Carbone, G., Richiardi, D., Dell'Orso, F., Ali, U., **Kashif, K.**, Roshanghias, R., Simonetta, P. L., Mei, A., & Di Benedetto, M.-G. (2024). *LIFE SENSOR: a multi-technology system for workers tracking and environment safety in Industrial construction sites.* IEEE Access (to be submitted in IEEE Access).

Appendix A

Appendix: Reassigned Spectrogram Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <float.h>
#include <math.h>

#include "rt_nonfinite.h"
#include "Nelsonspec.h"
#include "fft.h"

#ifndef M_PI
#define M_PI (3.14159265358979323846)
#endif

emxArray_real_T* emxCreateND_real_T(int numDimensions, int* size)
{
    emxArray_real_T* emx;
    int numEl;
    int i;
    emxInit_real_T(&emx, numDimensions);
    numEl = 1;
    for (i = 0; i < numDimensions; i++) {
        numEl *= size[i];
        emx->size[i] = size[i];
    }

    emx->data = (double*)calloc((unsigned int)numEl, sizeof(double));
    emx->numDimensions = numDimensions;
```

```
    emx->allocatedSize = numEl;
    return emx;
}

emxArray_real_T* emxCreateWrapperND_real_T(double* data, int numDimensions, int* size)
{
    emxArray_real_T* emx;
    int numEl;
    int i;
    emxInit_real_T(&emx, numDimensions);
    numEl = 1;
    for (i = 0; i < numDimensions; i++) {
        numEl *= size[i];
        emx->size[i] = size[i];
    }

    emx->data = data;
    emx->numDimensions = numDimensions;
    emx->allocatedSize = numEl;
    emx->canFreeData = false;
    return emx;
}

emxArray_real_T* emxCreateWrapper_real_T(double* data, int rows, int cols)
{
    emxArray_real_T* emx;
    emxInit_real_T(&emx, 2);
    emx->size[0] = rows;
    emx->size[1] = cols;
    emx->data = data;
    emx->numDimensions = 2;
    emx->allocatedSize = rows * cols;
    emx->canFreeData = false;
    return emx;
}

emxArray_real_T* emxCreate_real_T(int rows, int cols)
{
    emxArray_real_T* emx;
    int numEl;
    emxInit_real_T(&emx, 2);
    emx->size[0] = rows;
    numEl = rows * cols;
```

```
    emx->size[1] = cols;
    emx->data = (double*)calloc((unsigned int)numEl, sizeof(double));
    emx->numDimensions = 2;
    emx->allocatedSize = numEl;
    return emx;
}

void emxDestroyArray_real_T(emxArray_real_T* emxArray)
{
    emxFree_real_T(&emxArray);
}

void emxInitArray_real_T(emxArray_real_T** pEmxArray, int numDimensions)
{
    emxInit_real_T(pEmxArray, numDimensions);
}

void emxEnsureCapacity_creal_T(emxArray_creal_T* emxArray, int oldNumel)
{
    int newNumel;
    int i;
    void* newData;
    if (oldNumel < 0) {
        oldNumel = 0;
    }

    newNumel = 1;
    for (i = 0; i < emxArray->numDimensions; i++) {
        newNumel *= emxArray->size[i];
    }

    if (newNumel > emxArray->allocatedSize) {
        i = emxArray->allocatedSize;
        if (i < 16) {
            i = 16;
        }

        while (i < newNumel) {
            if (i > 1073741823) {
                i = MAX_int32_T;
            }
            else {
                i <<= 1;
```

```
            }
        }

        newData = calloc((unsigned int)i, sizeof(creal_T));
        if (emxArray->data != NULL) {
            memcpy(newData, emxArray->data, sizeof(creal_T) * oldNumel);
            if (emxArray->canFreeData) {
                free(emxArray->data);
            }
        }

        emxArray->data = (creal_T*)newData;
        emxArray->allocatedSize = i;
        emxArray->canFreeData = true;
    }
}

void emxEnsureCapacity_real_T(emxArray_real_T* emxArray, int oldNumel)
{
    int newNumel;
    int i;
    void* newData;
    if (oldNumel < 0) {
        oldNumel = 0;
    }

    newNumel = 1;
    for (i = 0; i < emxArray->numDimensions; i++) {
        newNumel *= emxArray->size[i];
    }

    if (newNumel > emxArray->allocatedSize) {
        i = emxArray->allocatedSize;
        if (i < 16) {
            i = 16;
        }

        while (i < newNumel) {
            if (i > 1073741823) {
                i = MAX_int32_T;
            }
            else {
                i <<= 1;
```

```
            }
        }

        newData = calloc((unsigned int)i, sizeof(double));
        if (emxArray->data != NULL) {
            memcpy(newData, emxArray->data, sizeof(double) * oldNumel);
            if (emxArray->canFreeData) {
                free(emxArray->data);
            }
        }

        emxArray->data = (double*)newData;
        emxArray->allocatedSize = i;
        emxArray->canFreeData = true;
    }
}

void emxFree_creal_T(emxArray_creal_T** pEmxArray)
{
    if (*pEmxArray != (emxArray_creal_T*)NULL) {
        if (((*pEmxArray)->data != (creal_T*)NULL) && (*pEmxArray)->canFreeData) {
            free((*pEmxArray)->data);
        }

        free((*pEmxArray)->size);
        free(*pEmxArray);
        *pEmxArray = (emxArray_creal_T*)NULL;
    }
}

void emxFree_real_T(emxArray_real_T** pEmxArray)
{
    if (*pEmxArray != (emxArray_real_T*)NULL) {
        if (((*pEmxArray)->data != (double*)NULL) && (*pEmxArray)->canFreeData) {
            free((*pEmxArray)->data);
        }

        free((*pEmxArray)->size);
        free(*pEmxArray);
        *pEmxArray = (emxArray_real_T*)NULL;
    }
}
```

```
void emxInit_creal_T(emxArray_creal_T** pEmxArray, int numDimensions)
{
    emxArray_creal_T* emxArray;
    int i;
    *pEmxArray = (emxArray_creal_T*)malloc(sizeof(emxArray_creal_T));
    emxArray = *pEmxArray;
    emxArray->data = (creal_T*)NULL;
    emxArray->numDimensions = numDimensions;
    emxArray->size = (int*)malloc(sizeof(int) * numDimensions);
    emxArray->allocatedSize = 0;
    emxArray->canFreeData = true;
    for (i = 0; i < numDimensions; i++) {
        emxArray->size[i] = 0;
    }
}

void emxInit_real_T(emxArray_real_T** pEmxArray, int numDimensions)
{
    emxArray_real_T* emxArray;
    int i;
    *pEmxArray = (emxArray_real_T*)malloc(sizeof(emxArray_real_T));
    emxArray = *pEmxArray;
    emxArray->data = (double*)NULL;
    emxArray->numDimensions = numDimensions;
    emxArray->size = (int*)malloc(sizeof(int) * numDimensions);
    emxArray->allocatedSize = 0;
    emxArray->canFreeData = true;
    for (i = 0; i < numDimensions; i++) {
        emxArray->size[i] = 0;
    }
}

static double rt_hypotd_snf(double u0, double u1)
{
    double y;
    double a;
    double b;
    a = fabs(u0);
    b = fabs(u1);
    if (a < b) {
        a /= b;
        y = b * sqrt(a * a + 1.0);
    } else if (a > b) {
```

```
        b /= a;
        y = a * sqrt(b * b + 1.0);
    } else if (rtIsNaN(b)) {
        y = b;
    } else {
        y = a * 1.4142135623730951;
    }

    return y;
}

static double rt_remd_snf(double u0, double u1)
{
    double y;
    double b_u1;
    double q;
    if (rtIsNaN(u0) || rtIsNaN(u1) || rtIsInf(u0)) {
        y = rtNaN;
    } else if (rtIsInf(u1)) {
        y = u0;
    } else {
        if (u1 < 0.0) {
            b_u1 = ceil(u1);
        } else {
            b_u1 = floor(u1);
        }

        if ((u1 != 0.0) && (u1 != b_u1)) {
            q = fabs(u0 / u1);
            if (fabs(q - floor(q + 0.5)) <= DBL_EPSILON * q) {
                y = 0.0 * u0;
            } else {
                y = fmod(u0, u1);
            }
        } else {
            y = fmod(u0, u1);
        }
    }

    return y;
}

static double rt_roundd_snf(double u)
```

```
{
    double y;
    if (fabs(u) < 4.503599627370496E+15) {
        if (u >= 0.5) {
            y = floor(u + 0.5);
        } else if (u > -0.5) {
            y = u * 0.0;
        } else {
            y = ceil(u - 0.5);
        }
    } else {
        y = u;
    }

    return y;
}

// Display matrix datas
bool displayMatrix( emxArray_real_T* matrix, const char* title, bool isInt, int count) {

    int _i, _c, _line, _count;
    char _txtFormat[255];

    if (matrix == NULL)
        return false;

    if (matrix->data == NULL)
        return false;

    if (matrix->numDimensions < 1)
        return false;

    _count = 1;
    for (_i = 0; _i < matrix->numDimensions; _i++)
        _count *= matrix->size[_i];

    printf("-----------");
    printf("%s", title);
    printf("(%d", _count);
    printf(":%d", matrix->size[0]);
    for (_i = 1; _i < matrix->numDimensions; _i++)
        printf("x%d", matrix->size[_i]);
    printf(")%s","");
```

```
    printf("%s-----------\n"," ");

    _c = 0;
    _line = 0;
    printf("%5d:", _line * count);
    for (_i = 0; _i < _count; _i++) {


        if (isInt)
            printf(" %d", (int)(matrix->data[_i] + 0.5));
        else
            printf(" %lf", matrix->data[_i]);

        _c++;
        if (_c == count) {

            _c = 0;
            _line++;
            printf("%s\n","");

            if (_i != _count - 1)
                printf("%5d:", _line * count);
        } // end if (_c...
    } // end for (_i...

    return true;
}

// Display complex matrix datas
bool displayComplexMatrix(emxArray_creal_T* matrix, const char* title, int count) {

    int _i, _c, _line, _count;
    char _txtFormat[255];

    if (matrix == NULL)
        return false;

    if (matrix->data == NULL)
        return false;

    if (matrix->numDimensions < 1)
        return false;
```

```
    _count = 1;
    for (_i = 0; _i < matrix->numDimensions; _i++)
        _count *= matrix->size[_i];

    printf("-----------%s", " ");
    printf("%s", title);
    printf("(%d", _count);
    printf(":%d", matrix->size[0]);
    for (_i = 1; _i < matrix->numDimensions; _i++)
        printf("x%d", matrix->size[_i]);
    printf(")%s", "");
    printf("%s----------\n", " ");

    _c = 0;
    _line = 0;
    printf("%5d:", _line * count);
    for (_i = 0; _i < _count; _i++) {


        printf(" %7.4f", matrix->data[_i].re);
        if (matrix->data[_i].im < 0)
            printf("%7.4fi", matrix->data[_i].im);
        else
            printf("+%7.4fi", matrix->data[_i].im);

        _c++;
        if (_c == count) {

            _c = 0;
            _line++;
            printf("%s\n", "");

            if (_i != _count - 1)
                printf("%5d:", _line * count);
        } // end if (_c...
    } // end for (_i...

    return true;
}

// Matrix translate
bool matrixTranslate(double* inDatas, int rows, int cols, double* outDatas) {
```

```
    if (inDatas == NULL || outDatas == NULL)
        return false;

    int _i, _row, _col, _c0, _c, _pos, _count;
    double* _datas;

    _count = rows * cols;
    _datas = (double*)calloc((unsigned int)_count, sizeof(double));
    for (_i = 0; _i < _count; _i++)
        _datas[_i] = inDatas[_i];

    for (_row = 0; _row < rows; _row++) {

        _c0 = _row * cols;
        for (_col = 0; _col < cols; _col++) {

            _c = _c0 + _col;
            _pos = _col * rows + _row;
            outDatas[_pos] = _datas[_c];
        } // end for _col
    } // end for _row

    free(_datas);

    return true;
}

// Matrix complex translate
bool matrixComplexTranslateE(emxArray_creal_T* inMatrix, emxArray_creal_T* outMatrix) {

    if (inMatrix == NULL || outMatrix == NULL)
        return false;
    if (inMatrix->numDimensions != 2 || outMatrix->numDimensions != 2)
        return false;

    int _i, _row, _col, _rows, _cols, _c, _pos, _count;
    double* _reDatas, * _imDatas;

    _rows = inMatrix->size[0];
    _cols = inMatrix->size[1];

    _count = _rows * _cols;
    _reDatas = (double*)calloc((unsigned int)_count, sizeof(double));
```

```
    _imDatas = (double*)calloc((unsigned int)_count, sizeof(double));

    for (_i = 0; _i < _count; _i++) {

        _reDatas[_i] = inMatrix->data[_i].re;
        _imDatas[_i] = inMatrix->data[_i].im;
    }

    _c = 0;
    _row = 0;
    _col = 0;
    for (_i = 0; _i < _count; _i++) {

        _pos = _row * _cols + _col;
        outMatrix->data[_pos].re = _reDatas[_c];
        outMatrix->data[_pos].im = _imDatas[_c];

        _c++;
        _row++;
        if (_row == _rows) {

            _row = 0;
            _col++;
        }
    }

    free(_imDatas);
    free(_reDatas);

    return true;
}

// Matrix one row move
bool matrixComplexOneRowMove(emxArray_creal_T* inMatrix, emxArray_creal_T* outMatrix) {

    if (inMatrix == NULL || outMatrix == NULL)
        return false;

    if (inMatrix->numDimensions != 2)
        return false;

    int _rows, _cols, _i, _c, _pos, _count;
    double* _reDatas, *_imDatas;
```

```
    _rows = inMatrix->size[0];
    _cols = inMatrix->size[1];
    _count = _rows * _cols;


    _reDatas = (double*)calloc((unsigned int)_count, sizeof(double));
    _imDatas = (double*)calloc((unsigned int)_count, sizeof(double));

    for (_i = 0; _i < _count; _i++) {

        _reDatas[_i] = inMatrix->data[_i].re;
        _imDatas[_i] = inMatrix->data[_i].im;
    }

    _c = 0;
    _pos = _count - _cols;
    for (_i = 0; _i < _cols; _i++) {

        outMatrix->data[_c].re = _reDatas[_pos];
        outMatrix->data[_c].im = _imDatas[_pos];

        _c++;
        _pos++;
    }

    for (_i = 0; _i < _count - _cols; _i++) {

        outMatrix->data[_c].re = _reDatas[_i];
        outMatrix->data[_c].im = _imDatas[_i];

        _c++;
    }

    free(_reDatas);
    free(_imDatas);

    return true;
}

// Matrix complex copy
bool matrixComplexCopy(emxArray_creal_T* leftMatrix, emxArray_creal_T* rightMatrix) {
```

```
    if (leftMatrix == NULL || rightMatrix == NULL)
        return false;

    if (rightMatrix->numDimensions != 2)
        return false;

    int _i, _count;

    _count = rightMatrix->size[0] * rightMatrix->size[1];

    for (_i = 0; _i < _count; _i++) {

        leftMatrix->data[_i].re = rightMatrix->data[_i].re;
        leftMatrix->data[_i].im = rightMatrix->data[_i].im;
    }

    return true;
}

// Matrix ones multiply
bool matrixOnesMultipy(double* inDatas, int inDataCount, int oneCount, double* outDatas) {

    if (inDatas == NULL || outDatas == NULL)
        return false;

    int _c, _rows, _cols, _row, _col;
    _rows = inDataCount;
    _cols = oneCount;

    _c = 0;
    for (_row = 0; _row < _rows; _row++) {

        for (_col = 0; _col < _cols; _col++) {

            outDatas[_c] = inDatas[_row];
            _c++;
        } // end for _col
    } // end for _row

    return true;
}

// Matrix left ones multiply
```

```
bool matrixLeftOnesMultipy(double* inDatas, int inDataCount, int oneCount, double* outDatas)

    if (inDatas == NULL || outDatas == NULL)
        return false;

    int _c, _rows, _cols, _row, _col;
    _rows = oneCount;
    _cols = inDataCount;

    _c = 0;
    for (_row = 0; _row < _rows; _row++) {

        for (_col = 0; _col < _cols; _col++) {

            outDatas[_c] = inDatas[_col];
            _c++;
        } // end for _col
    } // end for _row

    return true;
}

// Matrix multiply
bool matrixMultiply(double* inLeftDatas, int leftRows, int leftCols,
    double* inRightDatas, int rightRows, int rightCols, double* outDatas) {

    if (inLeftDatas == NULL || inRightDatas == NULL || outDatas == NULL)
        return false;

    int _c, _c0, _i, _row, _col;
    double _sum, _data, _leftData, _rightData;

    for (_row = 0; _row < leftRows; _row++) {

        _c0 = _row * leftCols;
        for (_col = 0; _col < leftCols; _col++) {

            _c = _c0 + _col;
            _sum = 0.0;
            for (_i = 0; _i < leftCols; _i++) {

                _leftData = inLeftDatas[_row * leftCols + _i];
                _rightData = inRightDatas[_i * rightRows + _col];
```

```
                _data = _leftData * _rightData;
                _sum += _data;
            } // end for _i
            outDatas[_c] = _sum;
        } // end for _col...
    } // end for _row...
    return true;
}

// Matrix dot multiply
bool matrixDotMultiply(double* inLeftDatas, double* inRightDatas, int rows, int cols, double

    if (inLeftDatas == NULL || inRightDatas == NULL || outDatas == NULL
        || rows < 1 || cols < 1)
        return false;

    int _i, _count;

    _count = rows * cols;
    for (_i = 0; _i < _count; _i++){

        outDatas[_i] = inLeftDatas[_i] * inRightDatas[_i];
    } // end for _i

    return true;
}

// Matrix dot complex multiply
bool matrixDotComplexMultiply(emxArray_creal_T* leftMatrix, emxArray_creal_T* rightMatrix, e

    if (leftMatrix == NULL || rightMatrix == NULL || outMatrix == NULL)
        return false;

    if (leftMatrix->numDimensions != 2 || rightMatrix->numDimensions != 2 || outMatrix->numD
        return false;

    int _i, _rows, _cols, _count;
    double _x1, _x2, _y1, _y2;

    _rows = leftMatrix->size[0];
    _cols = leftMatrix->size[1];
    _count = _rows * _cols;
```

```
    if (_count < 1)
        return false;

    for (_i = 0; _i < _count; _i++) {

        _x1 = leftMatrix->data[_i].re;
        _x2 = rightMatrix->data[_i].re;

        _y1 = leftMatrix->data[_i].im;
        _y2 = rightMatrix->data[_i].im;

        outMatrix->data[_i].re = _x1 * _x2 - _y1 * _y2;
        outMatrix->data[_i].im = _x1 * _y2 + _x2 * _y1;
    }

    return true;
}

// Matrix dot conjugate complex multiply
bool matrixDotConjComplexMultiply(emxArray_creal_T* leftMatrix, emxArray_creal_T* rightMatri

    if (leftMatrix == NULL || rightMatrix == NULL || outMatrix == NULL)
        return false;

    if (leftMatrix->numDimensions != 2 || rightMatrix->numDimensions != 2 || outMatrix->numD
        return false;

    int _i, _rows, _cols, _count;
    double _x1, _x2, _y1, _y2;

    _rows = leftMatrix->size[0];
    _cols = leftMatrix->size[1];
    _count = _rows * _cols;

    if (_count < 1)
        return false;

    for (_i = 0; _i < _count; _i++) {

        _x1 = leftMatrix->data[_i].re;
        _x2 = rightMatrix->data[_i].re;

        _y1 = leftMatrix->data[_i].im;
```

```
        _y2 = rightMatrix->data[_i].im;

        outMatrix->data[_i].re = _x1 * _x2 + _y1 * _y2;
        outMatrix->data[_i].im = _x2 * _y1 - _x1 * _y2;
    }

    return true;
}

// Matrix dot get angle complex multiply
bool matrixDotGetAngleComplex(emxArray_creal_T* inMatrix, bool modSign, emxArray_real_T* out

    if (inMatrix == NULL || outMatrix == NULL)
        return false;

    if (inMatrix->numDimensions != 2 || outMatrix->numDimensions != 2)
        return false;

    int _i, _count, _rows, _cols;
    double _angle, _2pi;

    _rows = inMatrix->size[0];
    _cols = inMatrix->size[1];
    _count = _rows * _cols;

    _2pi = 2.0 * M_PI;

    for (_i = 0; _i < _count; _i++) {

        _angle = atan2(inMatrix->data[_i].im, inMatrix->data[_i].re);

        if (modSign) {

            if (_angle > 0)
                _angle -= _2pi;

            if (_angle <= -_2pi + 1E-5)
                _angle += _2pi;

            if (fabs(_angle) < 1E-5)
                _angle = 0.0;
        }
        else {
```

```
            if (_angle < 0)
                _angle += _2pi;

            if (_angle >= _2pi - 1E-5)
                _angle -= _2pi;

            if (fabs(_angle) < 1E-5)
                _angle = 0.0;
        }

        outMatrix->data[_i] = _angle;
    }

    return true;
}

// Matrix addition
bool matrixAddition(double* inLeftDatas, double* inRightDatas, int rows, int cols, double* o

    int _i, _count;

    _count = rows * cols;
    for (_i = 0; _i < _count; _i++)
        outDatas[_i] = inLeftDatas[_i] + inRightDatas[_i];

    return true;
}

// Matrix value addition
bool matrixValueAddition(double* inDatas, int rows, int cols, double value, double* outDatas

    int _i, _count;

    _count = rows * cols;
    for (_i = 0; _i < _count; _i++)
        outDatas[_i] = inDatas[_i] + value;

    return true;
}

// Matrix value multiply
bool matrixValueMultiply(double* inDatas, int rows, int cols, double value, double* outDatas
```

```
    int _i, _count;

    _count = rows * cols;
    for (_i = 0; _i < _count; _i++)
        outDatas[_i] = inDatas[_i] * value;

    return true;
}

// Matrix get as index
bool matrixGetAsIndex(const double* inDatas, int inDataCount, emxArray_real_T* matIndex, emx


    if (inDatas == NULL || inDataCount < 1 || matIndex == NULL || outMatrix == NULL)
        return false;

    if (matIndex->numDimensions < 2 || outMatrix->numDimensions < 2)
        return false;

    int _c, _index, _row, _col;

    _c = 0;
    for (_row = 0; _row < matIndex->size[0]; _row++) {

        for (_col = 0; _col < matIndex->size[1]; _col++) {

            _index = (int)(matIndex->data[_c] + 0.5);
            _index--;
            if (_index < 0 && _index >= inDataCount)
                return false;

            outMatrix->data[_c] = inDatas[_index];
            _c++;
        } // end for _col
    } // end for _row

    return true;
}

// Calculate the window matrix using the hanning
bool calcMatWindow(int window, emxArray_real_T* matWindow) {
```

```
    if (window < 1 || matWindow == NULL)
        return false;

    if (matWindow->numDimensions != 1)
        return false;

    int _i, _count, _count1, _size0;
    double _angle, _data, _coef;


    _size0 = matWindow->size[0];
    matWindow->size[0] = window;
    emxEnsureCapacity_real_T(matWindow, _size0);

    _count = window;
    _count1 = _count + 1;
    _coef = 2.0 * M_PI / _count1;
    for (_i = 0; _i < _count; _i++) {

        _angle = (double)(_i + 1) * _coef;
        _data = (1.0 - cos(_angle)) / 2.0;
        matWindow->data[_i] = _data;
    }

    return true;
}

// Calculate the offset matrix
bool calcMatOffset(int signalCount, int window, int overlap, emxArray_real_T* matOffset) {

    if (signalCount < 1 || window < 1 || overlap < 1 || matOffset == NULL)
        return false;

    if (matOffset->numDimensions != 1)
        return false;

    int _i, _c, _step, _count, _count0, _size0;

    _step = window - overlap;

    if (_step < 1)
        _step = 1;
```

```
    _count0 = signalCount - window - 1;
    _count = _count0 / _step;
    if (_count % _step != 0)
        _count++;

    _size0 = matOffset->size[0];
    matOffset->size[0] = _count;
    emxEnsureCapacity_real_T(matOffset, _size0);

    _c = 0;
    for (_i = 0; _i < _count; _i++) {

        if (_c > _count0)
            _c = _count0;

        matOffset->data[_i] = _c;
        _c += _step;
    }

    return true;
}

bool CalcNelsonspecFirst(const double* signals, NelsonSetting setting,
    emxArray_creal_T* STFT, emxArray_creal_T* STFTdel, emxArray_creal_T* STFTfreqdel) {

    int _i, _j, _c, _c0, _size0;

    emxArray_real_T* _matWindow, * _matOffset, * _matWW, * _matIdx,
                    * _matIdx1, * _leftDatas, * _rightDatas, * _matS, * _matSdel;
    emxArray_creal_T* _matSTFT, * _matSTFTdel, * _matSTFTfreqdel;

    // Calculate the window matrix
    emxInit_real_T(&_matWindow, 1);
    if (!calcMatWindow(setting.window, _matWindow)) {

        emxFree_real_T(&_matWindow);
        return false;
    }

    //displayMatrix(_matWindow, "window matrix", 10);

    // Calculate the offset matrix
    emxInit_real_T(&_matOffset, 1);
```

```
if (!calcMatOffset(setting.signalCount, setting.window, setting.ovelap, _matOffset)) {

    emxFree_real_T(&_matWindow);
    emxFree_real_T(&_matOffset);
    return false;
}

//displayMatrix(_matOffset, "offset matrix", 10);

// Calculate the WW matrix
emxInit_real_T(&_matWW, 2);
_size0 = _matWW->size[0] * _matWW->size[1];
_matWW->size[0] = _matWindow->size[0];
_matWW->size[1] = _matOffset->size[0];
emxEnsureCapacity_real_T(_matWW, _size0);
if (!matrixOnesMultipy(_matWindow->data, _matWindow->size[0], _matOffset->size[0], _matW

    emxFree_real_T(&_matWindow);
    emxFree_real_T(&_matOffset);
    emxFree_real_T(&_matWW);
    return false;
}

//displayMatrix(_matWW, "WW matrix", 10);

// Calculate the idx matrix
emxInit_real_T(&_leftDatas, 2);
_size0 = _leftDatas->size[0] * _leftDatas->size[1];
_leftDatas->size[0] = _matWindow->size[0];
_leftDatas->size[1] = _matOffset->size[0];
emxEnsureCapacity_real_T(_leftDatas, _size0);
for (_i = 0; _i < _leftDatas->size[0]; _i++) {

    _c0 = _i * _leftDatas->size[1];
    for (_j = 0; _j < _leftDatas->size[1]; _j++) {

        _c = _c0 + _j;

        _leftDatas->data[_c] = _i + 1;
    } // end for _j
} // end for _i

emxInit_real_T(&_rightDatas, 2);
```

```
_size0 = _rightDatas->size[0] * _rightDatas->size[1];
_rightDatas->size[0] = _matWindow->size[0];
_rightDatas->size[1] = _matOffset->size[0];
emxEnsureCapacity_real_T(_rightDatas, _size0);
matrixLeftOnesMultipy(_matOffset->data, _matOffset->size[0], _matWindow->size[0], _right

emxInit_real_T(&_matIdx, 2);
_size0 = _matIdx->size[0] * _matIdx->size[1];
_matIdx->size[0] = _matWindow->size[0];
_matIdx->size[1] = _matOffset->size[0];
emxEnsureCapacity_real_T(_matIdx, _size0);

matrixAddition(_leftDatas->data, _rightDatas->data, _leftDatas->size[0], _leftDatas->siz

//displayMatrix(_matIdx, "idx matrix", 10);

// Calculate the idx1 matrix
emxInit_real_T(&_matIdx1, 2);
_size0 = _matIdx1->size[0] * _matIdx1->size[1];
_matIdx1->size[0] = _matWindow->size[0];
_matIdx1->size[1] = _matOffset->size[0];
emxEnsureCapacity_real_T(_matIdx1, _size0);
matrixValueAddition(_matIdx->data, _matIdx->size[0], _matIdx->size[1], 1, _matIdx1->data

//displayMatrix(_matIdx1, "idx+1 matrix", true, 10);

// Calculate the S matrix
emxInit_real_T(&_matS, 2);
_size0 = _matS->size[0] * _matS->size[1];
_matS->size[0] = _matWindow->size[0];
_matS->size[1] = _matOffset->size[0];
emxEnsureCapacity_real_T(_matS, _size0);
if (!matrixGetAsIndex(signals, setting.signalCount, _matIdx1, _matS)) {

    emxFree_real_T(&_matS);
    emxFree_real_T(&_matIdx1);
    emxFree_real_T(&_leftDatas);
    emxFree_real_T(&_rightDatas);
    emxFree_real_T(&_matIdx);
    emxFree_real_T(&_matWW);
    emxFree_real_T(&_matOffset);
    emxFree_real_T(&_matWindow);
```

```
    return false;
}

matrixDotMultiply( _matS->data, _matWW->data, _matWW->size[0], _matWW->size[1], _matS->d
matrixTranslate(_matS->data, _matS->size[0], _matS->size[1], _matS->data);

// Calculate the Sdel matrix
emxInit_real_T(&_matSdel, 2);
_size0 = _matSdel->size[0] * _matSdel->size[1];
_matSdel->size[0] = _matWindow->size[0];
_matSdel->size[1] = _matOffset->size[0];
emxEnsureCapacity_real_T(_matSdel, _size0);
if (!matrixGetAsIndex(signals, setting.signalCount, _matIdx, _matSdel)) {

    emxFree_real_T(&_matSdel);
    emxFree_real_T(&_matS);
    emxFree_real_T(&_matIdx1);
    emxFree_real_T(&_leftDatas);
    emxFree_real_T(&_rightDatas);
    emxFree_real_T(&_matIdx);
    emxFree_real_T(&_matWW);
    emxFree_real_T(&_matOffset);
    emxFree_real_T(&_matWindow);

    return false;
}

matrixDotMultiply(_matSdel->data, _matWW->data, _matWW->size[0], _matWW->size[1], _matSd
matrixTranslate(_matSdel->data, _matSdel->size[0], _matSdel->size[1], _matSdel->data);

// Calculate the STFT matrix
emxInit_creal_T(&_matSTFT, 2);
fft_emx( _matS, setting.fftn, _matSTFT);
matrixComplexTranslateE(_matSTFT, _matSTFT);
//displayComplexMatrix(_matSTFT, "STFT matrix", 5);

// Calculate the STFTfreqdel matrix
emxInit_creal_T(&_matSTFTfreqdel, 2);
_size0 = _matSTFTfreqdel->size[0] * _matSTFTfreqdel->size[1];
_matSTFTfreqdel->size[0] = _matWindow->size[0];
_matSTFTfreqdel->size[1] = _matOffset->size[0];
emxEnsureCapacity_creal_T(_matSTFTfreqdel, _size0);
matrixComplexCopy(_matSTFTfreqdel, _matSTFT);
```

```
matrixComplexOneRowMove(_matSTFTfreqdel, _matSTFTfreqdel);
//displayComplexMatrix(_matSTFTfreqdel, "STFTfreqdel matrix", 5);

// Calculate the STFTdel matrix
emxInit_creal_T(&_matSTFTdel, 2);
fft_emx(_matSdel, setting.fftn, _matSTFTdel);
matrixComplexTranslateE(_matSTFTdel, _matSTFTdel);
//displayComplexMatrix(_matSTFTdel, "STFTdel matrix", 5);

// Copy the result to STFT
_size0 = STFT->size[0] * STFT->size[1];
STFT->size[0] = _matWindow->size[0];
STFT->size[1] = _matOffset->size[0];
emxEnsureCapacity_creal_T(STFT, _size0);
matrixComplexCopy(STFT, _matSTFT);

// Copy the result to STFTdel
_size0 = STFTdel->size[0] * STFTdel->size[1];
STFTdel->size[0] = _matWindow->size[0];
STFTdel->size[1] = _matOffset->size[0];
emxEnsureCapacity_creal_T(STFTdel, _size0);
matrixComplexCopy(STFTdel, _matSTFTdel);

// Copy the result to STFTfreqdel
_size0 = STFTfreqdel->size[0] * STFTfreqdel->size[1];
STFTfreqdel->size[0] = _matWindow->size[0];
STFTfreqdel->size[1] = _matOffset->size[0];
emxEnsureCapacity_creal_T(STFTfreqdel, _size0);
matrixComplexCopy(STFTfreqdel, _matSTFTfreqdel);

emxFree_creal_T(&_matSTFTfreqdel);
emxFree_creal_T(&_matSTFTdel);
emxFree_creal_T(&_matSTFT);
emxFree_real_T(&_matSdel);
emxFree_real_T(&_matS);
emxFree_real_T(&_matIdx1);
emxFree_real_T(&_leftDatas);
emxFree_real_T(&_rightDatas);
emxFree_real_T(&_matIdx);
emxFree_real_T(&_matWW);
emxFree_real_T(&_matOffset);
emxFree_real_T(&_matWindow);
```

```
    return true;
}

bool CalcNelsonspecSecond(NelsonSetting setting, emxArray_creal_T* STFT, emxArray_creal_T* S
                          emxArray_real_T* STFTmag, emxArray_real_T * STFTplot, emxArray_r

    int _i, _count, _pos, _c, _row, _col, _rows, _cols, _size0, _rows2,
        _fftn, _Fs, _n, _low, _high, _lowIndex, _highIndex, _iData;
    double _delay, _coef, _data, _data2, _x, _y, _mxData;

    emxArray_creal_T* _matSTFTpos, * _matSTFTdelpos, * _matSTFTfreqdelpos, * _matC, * _matL;
    emxArray_real_T* _matArgC, * _matArgL, * _matCIFpos, * _matLGDpos, * _matTreMap,
                    * _matT, * _matOffset, * _matSTFTmag, * _matPlotThese0, * _matPlotThese,
                    * _matSTFTplot, * _matCIFplot, * _matTreMapplot;

    _fftn = setting.fftn;
    _high = setting.high;
    _low = setting.low;
    _Fs = setting.Fs;
    _delay = 1.0;

    if (_fftn < 1 || _high < 1 || _low < 0)
        return false;

    if (STFT == NULL || STFTdel == NULL || STFTfreqdel == NULL)
        return false;

    if (STFT->numDimensions != 2 || STFTdel->numDimensions != 2 || STFTfreqdel->numDimension
        return false;

    _rows = STFT->size[0];
    _cols = STFT->size[1];

    if (_rows < 1 || _cols < 1)
        return false;

    _n = _fftn / 2;
    _iData = _Fs * (_n - 1) / _fftn;
    if (_high > _iData)
        _high = _iData;

    _lowIndex = (int)((double)_low * _fftn / _Fs + 0.5);
    if (_lowIndex == 0)
```

```
    _lowIndex = 1;
_highIndex = (int)((double)_high * _fftn / _Fs + 0.5);

_rows2 = _highIndex - _lowIndex + 1;
if (_rows2 < 1)
    return false;

// Initialize the STFTpos matrix
emxInit_creal_T(&_matSTFTpos, 2);
_size0 = _matSTFTpos->size[0] * _matSTFTpos->size[1];
_matSTFTpos->size[0] = _rows2;
_matSTFTpos->size[1] = _cols;
emxEnsureCapacity_creal_T(_matSTFTpos, _size0);

// Initialize the STFTdelpos matrix
emxInit_creal_T(&_matSTFTdelpos, 2);
_size0 = _matSTFTdelpos->size[0] * _matSTFTdelpos->size[1];
_matSTFTdelpos->size[0] = _rows2;
_matSTFTdelpos->size[1] = _cols;
emxEnsureCapacity_creal_T(_matSTFTdelpos, _size0);

// Initialize the STFTfreqdelpos matrix
emxInit_creal_T(&_matSTFTfreqdelpos, 2);
_size0 = _matSTFTfreqdelpos->size[0] * _matSTFTfreqdelpos->size[1];
_matSTFTfreqdelpos->size[0] = _rows2;
_matSTFTfreqdelpos->size[1] = _cols;
emxEnsureCapacity_creal_T(_matSTFTfreqdelpos, _size0);

// Get the STFTpos, STFTdelpos, STFTfreqdelpos Matrix

_lowIndex--;
_highIndex--;

_c = 0;
for (_row = _lowIndex; _row <= _highIndex; _row++) {

    _pos = _row * _cols;
    for (_col = 0; _col < _cols; _col++) {

        _matSTFTpos->data[_c].re = STFT->data[_pos].re;
        _matSTFTpos->data[_c].im = STFT->data[_pos].im;

        _matSTFTdelpos->data[_c].re = STFTdel->data[_pos].re;
```

```
            _matSTFTdelpos->data[_c].im = STFTdel->data[_pos].im;

            _matSTFTfreqdelpos->data[_c].re = STFTfreqdel->data[_pos].re;
            _matSTFTfreqdelpos->data[_c].im = STFTfreqdel->data[_pos].im;

            _pos++;
            _c++;
        } // end for _col
    } // end for _row

    //displayComplexMatrix(_matSTFTpos, "STFTpos MATRIX", 5);
    //displayComplexMatrix(_matSTFTdelpos, "STFTdelpos MATRIX", 5);

    // Initialize the C matrix
    emxInit_creal_T(&_matC, 2);
    _size0 = _matC->size[0] * _matC->size[1];
    _matC->size[0] = _rows2;
    _matC->size[1] = _cols;
    emxEnsureCapacity_creal_T(_matC, _size0);

    // Calculate the C matrix
    if (!matrixDotConjComplexMultiply(_matSTFTpos, _matSTFTdelpos, _matC)) {

        emxFree_creal_T(&_matC);
        emxFree_creal_T(&_matSTFTpos);
        emxFree_creal_T(&_matSTFTdelpos);
        emxFree_creal_T(&_matSTFTfreqdelpos);

        return false;
    }

    //displayComplexMatrix(_matC, "C Matrix", 5);

    // Initialize the L matrix
    emxInit_creal_T(&_matL, 2);
    _size0 = _matL->size[0] * _matL->size[1];
    _matL->size[0] = _rows2;
    _matL->size[1] = _cols;
    emxEnsureCapacity_creal_T(_matL, _size0);

    // Calculate the L matrix
    if (!matrixDotConjComplexMultiply(_matSTFTpos, _matSTFTfreqdelpos, _matL)) {
```

```
    emxFree_creal_T(&_matL);
    emxFree_creal_T(&_matC);
    emxFree_creal_T(&_matSTFTpos);
    emxFree_creal_T(&_matSTFTdelpos);
    emxFree_creal_T(&_matSTFTfreqdelpos);

    return false;
}

// Get the argC matrix
emxInit_real_T(&_matArgC, 2);
_size0 = _matArgC->size[0] * _matArgC->size[1];
_matArgC->size[0] = _rows2;
_matArgC->size[1] = _cols;
emxEnsureCapacity_real_T(_matArgC, _size0);
matrixDotGetAngleComplex(_matC, false, _matArgC);

//displayMatrix(_matArgC, "argC matrix", false, 10);

// Calculate the CIFpos matrix
emxInit_real_T(&_matCIFpos, 2);
_size0 = _matCIFpos->size[0] * _matCIFpos->size[1];
_matCIFpos->size[0] = _rows2;
_matCIFpos->size[1] = _cols;
emxEnsureCapacity_real_T(_matCIFpos, _size0);
_coef = (double)_Fs / _delay / 2.0 / M_PI;
matrixValueMultiply(_matArgC->data, _matArgC->size[0], _matArgC->size[1], _coef, _matCIF

//displayMatrix(_matCIFpos, "CIFpos matrix", false, 10);

// Get the argL matrix
emxInit_real_T(&_matArgL, 2);
_size0 = _matArgL->size[0] * _matArgL->size[1];
_matArgL->size[0] = _rows2;
_matArgL->size[1] = _cols;
emxEnsureCapacity_real_T(_matArgL, _size0);
matrixDotGetAngleComplex(_matL, true, _matArgL);

// Calculate the LGDpos matrix
emxInit_real_T(&_matLGDpos, 2);
_size0 = _matLGDpos->size[0] * _matLGDpos->size[1];
_matLGDpos->size[0] = _rows2;
_matLGDpos->size[1] = _cols;
```

```
emxEnsureCapacity_real_T(_matLGDpos, _size0);
_coef = -(double)_fftn / (double)_Fs / 2.0 / M_PI;
matrixValueMultiply(_matArgL->data, _matArgL->size[0], _matArgL->size[1], _coef, _matLGD
//displayMatrix(_matLGDpos, "LGDpos matrix", false, 10);

// Calculate the offset matrix
emxInit_real_T(&_matOffset, 1);
_size0 = _matOffset->size[0];
_matOffset->size[0] = _cols;
emxEnsureCapacity_real_T(_matOffset, _size0);
calcMatOffset(setting.signalCount, setting.window, setting.ovelap, _matOffset);

// Calculate the t matrix
emxInit_real_T(&_matT, 1);
_size0 = _matT->size[0];
_matT->size[0] = _cols;
emxEnsureCapacity_real_T(_matT, _size0);
for (_col = 0; _col < _cols; _col++)
    _matT->data[_col] = (_matOffset->data[_col] + (double)(setting.window) / 2.0) / _Fs;

// Calculate the tremap matrix
emxInit_real_T(&_matTreMap, 2);
_size0 = _matTreMap->size[0] * _matTreMap->size[1];
_matTreMap->size[0] = _rows2;
_matTreMap->size[1] = _cols;
emxEnsureCapacity_real_T(_matTreMap, _size0);
_c = 0;
for (_row = 0; _row < _rows2; _row++) {

    for (_col = 0; _col < _cols; _col++) {

        _data2 = _matT->data[_col] - ((double)(setting.window) / 2.0 - 1.0) / _Fs;
        _data = _matLGDpos->data[_c] + _data2;
        _matTreMap->data[_c] = _data;
        _c++;
    } // end for _col
} // end for _row

//displayMatrix(_matTreMap, "tremap matrix", false, 10);

// Calculate the STFTmag matrix
emxInit_real_T(&_matSTFTmag, 2);
_size0 = _matSTFTmag->size[0] * _matSTFTmag->size[1];
```

```
_matSTFTmag->size[0] = _rows2;
_matSTFTmag->size[1] = _cols;
emxEnsureCapacity_real_T(_matSTFTmag, _size0);

_count = _rows2 * _cols;
for (_i = 0; _i < _count; _i++) {

    _x = _matSTFTpos->data[_i].re;
    _y = _matSTFTpos->data[_i].im;
    _matSTFTmag->data[_i] = sqrt(_x * _x + _y * _y);
}

// Get the maximum data
_mxData = _matSTFTmag->data[0];
for (_i = 1; _i < _count; _i++) {

    _data = _matSTFTmag->data[_i];
    if (_data > _mxData)
        _mxData = _data;
} // end for _i

// Normalize and calculate
for (_i = 0; _i < _count; _i++) {

    _data = _matSTFTmag->data[_i] / _mxData;
    _data = 20.0 * log10(_data);
    _matSTFTmag->data[_i] = _data;
} // end for _i
//displayMatrix(_matSTFTmag, "STFTmag matrix", false, 10);

// Calculate the plot these
emxInit_real_T(&_matPlotThese0, 1);
_size0 = _matPlotThese0->size[0];
_matPlotThese0->size[0] = _rows2 * _cols;
emxEnsureCapacity_real_T(_matPlotThese0, _size0);
_count = _rows2 * _cols;
_c = 0;
for (_col = 0; _col < _cols; _col++) {

    for (_row = 0; _row < _rows2; _row++) {

        _pos = _row * _cols + _col;
        if (_matSTFTmag->data[_pos] >= setting.clip && _matCIFpos->data[_pos] >= setting
```

```
            && _matCIFpos->data[_pos] <= _high && _matTreMap->data[_pos] >= _matT->data[
            && _matTreMap->data[_pos] <= _matT->data[_matT->size[0] - 1]) {

            _matPlotThese0->data[_c] = _pos;
            _c++;
        } // end if ...
    } // end for _row
} // end for _col

if (_c < 1) {

    emxFree_real_T(&_matPlotThese0);
    emxFree_real_T(&_matSTFTmag);
    emxFree_real_T(&_matTreMap);
    emxFree_real_T(&_matT);
    emxFree_real_T(&_matLGDpos);
    emxFree_real_T(&_matCIFpos);
    emxFree_real_T(&_matArgL);
    emxFree_real_T(&_matArgC);
    emxFree_creal_T(&_matL);
    emxFree_creal_T(&_matC);
    emxFree_creal_T(&_matSTFTpos);
    emxFree_creal_T(&_matSTFTdelpos);
    emxFree_creal_T(&_matSTFTfreqdelpos);

    return false;
}

// Set the plot these matrix
emxInit_real_T(&_matPlotThese, 1);
_size0 = _matPlotThese->size[0];
_matPlotThese->size[0] = _c;
emxEnsureCapacity_real_T(_matPlotThese, _size0);
for (_i = 0; _i < _c; _i++)
    _matPlotThese->data[_i] = _matPlotThese0->data[_i];
emxFree_real_T(&_matPlotThese0);

//displayMatrix(_matPlotThese, "plot these matrix", true, 10);

// Calculate the STFT, CIF, tremap plot matrix
_count = _matPlotThese->size[0];
emxInit_real_T(&_matSTFTplot, 1);
_size0 = _matSTFTplot->size[0];
```

```
_matSTFTplot->size[0] = _matPlotThese->size[0];
emxEnsureCapacity_real_T(_matSTFTplot, _size0);

emxInit_real_T(&_matCIFplot, 1);
_size0 = _matCIFplot->size[0];
_matCIFplot->size[0] = _matPlotThese->size[0];
emxEnsureCapacity_real_T(_matCIFplot, _size0);

emxInit_real_T(&_matTreMapplot, 1);
_size0 = _matTreMapplot->size[0];
_matTreMapplot->size[0] = _matPlotThese->size[0];
emxEnsureCapacity_real_T(_matTreMapplot, _size0);

for (_i = 0; _i < _count; _i++){

    _pos = _matPlotThese->data[_i];
    _matSTFTplot->data[_i] = _matSTFTmag->data[_pos];
    _matCIFplot->data[_i] = _matCIFpos->data[_pos];
    _matTreMapplot->data[_i] = _matTreMap->data[_pos];
}

//displayMatrix(_matSTFTplot, "STFT plot matrix", false, 5);
//displayMatrix(_matCIFplot, "CIF plot matrix", false, 5);
//displayMatrix(_matTreMapplot, "tremap plot matrix", false, 5);

// Copy the results
_size0 = STFTplot->size[0];
STFTplot->size[0] = _count;
emxEnsureCapacity_real_T(STFTplot, _size0);

_size0 = CIFplot->size[0];
CIFplot->size[0] = _count;
emxEnsureCapacity_real_T(CIFplot, _size0);

_size0 = TreMapPlot->size[0];
TreMapPlot->size[0] = _count;
emxEnsureCapacity_real_T(TreMapPlot, _size0);
for (_i = 0; _i < _count; _i++) {

    STFTplot->data[_i] = _matSTFTplot->data[_i];
    CIFplot->data[_i] = _matCIFplot->data[_i];
    TreMapPlot->data[_i] = _matTreMapplot->data[_i];
} // end for _i
```

```
    // Calculate the STFTmag
    _size0 = STFTmag->size[0] * STFTmag->size[1];
    STFTmag->size[0] = _rows2;
    STFTmag->size[1] = _cols;
    emxEnsureCapacity_real_T(STFTmag, _size0);
    _c = 0;
    for (_col = 0; _col < _cols; _col++) {

        for (_row = 0; _row < _rows2; _row++) {

            STFTmag->data[_c] = _matSTFTmag->data[_c];
            _c++;
        } // end for _row
    } // end for _col

    emxFree_real_T(&_matTreMapplot);
    emxFree_real_T(&_matCIFplot);
    emxFree_real_T(&_matSTFTplot);
    emxFree_real_T(&_matPlotThese);
    emxFree_real_T(&_matSTFTmag);
    emxFree_real_T(&_matTreMap);
    emxFree_real_T(&_matT);
    emxFree_real_T(&_matLGDpos);
    emxFree_real_T(&_matCIFpos);
    emxFree_real_T(&_matArgL);
    emxFree_real_T(&_matArgC);
    emxFree_creal_T(&_matL);
    emxFree_creal_T(&_matC);
    emxFree_creal_T(&_matSTFTpos);
    emxFree_creal_T(&_matSTFTdelpos);
    emxFree_creal_T(&_matSTFTfreqdelpos);

    return true;
}

void Nelsonspec(const double b_signal[33707], double Fs, double win_size, double
    overlap, double fftn, double low, double high, double clip,
    emxArray_real_T* STFTmag, emxArray_real_T* CIFPos_mag, emxArray_real_T* time, emxArray_r
{
    emxArray_real_T* y;
    int i0;
    int loop_ub;
```

```
    emxArray_real_T* window;
    int nx;
    int k;
    double d;
    emxArray_real_T* offset;
    emxArray_real_T* time_offset;
    double ndbl;
    double cdiff;
    double apnd;
    int i1;
    emxArray_real_T* b_y;
    emxArray_real_T* r0;
#if 1
    emxArray_real_T* r0_cif;
    emxArray_real_T* r1_cif;
    emxArray_creal_T* STFTfreqdelpos;
    emxArray_creal_T* STFTdelpos;
    emxArray_creal_T* CIFpos;
    emxArray_creal_T* LGDpos;
    emxArray_creal_T* time_t;
    emxArray_creal_T* L;


#endif
    emxArray_real_T* r1;
    emxArray_creal_T* STFT;
    double d0;
    emxArray_creal_T* STFTpos;
    int m;
    unsigned int unnamed_idx_1;
    boolean_T exitg1;
    (void)clip;
    emxInit_real_T(&y, 2);
    if (rtIsNaN(win_size)) {
        i0 = y->size[0] * y->size[1];
        y->size[0] = 1;
        y->size[1] = 1;
        emxEnsureCapacity_real_T(y, i0);
        y->data[0] = rtNaN;
    }
    else if (win_size < 1.0) {
        y->size[0] = 1;
        y->size[1] = 0;
```

```
}
else if (rtIsInf(win_size) && (1.0 == win_size)) {
    i0 = y->size[0] * y->size[1];
    y->size[0] = 1;
    y->size[1] = 1;
    emxEnsureCapacity_real_T(y, i0);
    y->data[0] = rtNaN;
}
else {
    i0 = y->size[0] * y->size[1];
    y->size[0] = 1;
    loop_ub = (int)floor(win_size - 1.0);
    y->size[1] = loop_ub + 1;
    emxEnsureCapacity_real_T(y, i0);
    for (i0 = 0; i0 <= loop_ub; i0++) {
        y->data[i0] = 1.0 + (double)i0;
    }
}

emxInit_real_T(&window, 1);
i0 = window->size[0];
window->size[0] = y->size[1];
emxEnsureCapacity_real_T(window, i0);
loop_ub = y->size[1];
for (i0 = 0; i0 < loop_ub; i0++) {
    window->data[i0] = 6.2831853071795862 * y->data[i0] / (win_size + 1.0);
}

nx = window->size[0];
for (k = 0; k < nx; k++) {
    window->data[k] = cos(window->data[k]);
}

i0 = window->size[0];
emxEnsureCapacity_real_T(window, i0);
loop_ub = window->size[0];
for (i0 = 0; i0 < loop_ub; i0++) {
    window->data[i0] = 0.5 * (1.0 - window->data[i0]);
}

d = (double)window->size[0] - overlap;
emxInit_real_T(&offset, 2);
emxInit_real_T(&time_offset, 2);
```

```
if (rtIsNaN(d)) {
    i0 = offset->size[0] * offset->size[1];
    offset->size[0] = 1;
    offset->size[1] = 1;
    emxEnsureCapacity_real_T(offset, i0);
    offset->data[0] = rtNaN;
}
else if ((d == 0.0) || ((0 < 33706 - window->size[0]) && (d < 0.0)) ||
    ((33706 - window->size[0] < 0) && (d > 0.0))) {
    offset->size[0] = 1;
    offset->size[1] = 0;
}
else if (rtIsInf(d)) {
    i0 = offset->size[0] * offset->size[1];
    offset->size[0] = 1;
    offset->size[1] = 1;
    emxEnsureCapacity_real_T(offset, i0);
    offset->data[0] = 0.0;
}
else if (floor(d) == d) {
    cdiff = (33707.0 - (double)window->size[0]) - 1.0;
    i0 = offset->size[0] * offset->size[1];
    offset->size[0] = 1;
    loop_ub = (int)floor(cdiff / d);
    offset->size[1] = loop_ub + 1;
    emxEnsureCapacity_real_T(offset, i0);
    for (i0 = 0; i0 <= loop_ub; i0++) {
        offset->data[i0] = d * (double)i0;
    }
}
else {
    ndbl = floor(((33707.0 - (double)window->size[0]) - 1.0) / d + 0.5);
    apnd = ndbl * d;
    if (d > 0.0) {
        cdiff = apnd - ((33707.0 - (double)
            window->size[0]) - 1.0);
    }
    else {
        cdiff = ((33707.0 - (double)window->size[0]) - 1.0) - apnd;
    }

    if (fabs(cdiff) < 4.4408920985006262E-16 * fabs((33707.0 - (double)
        window->size[0]) - 1.0)) {
```

```
            ndbl++;
            apnd = (33707.0 - (double)window->size[0]) - 1.0;
        }
        else if (cdiff > 0.0) {
            apnd = (ndbl - 1.0) * d;
        }
        else {
            ndbl++;
        }

        if (ndbl >= 0.0) {
            loop_ub = (int)ndbl;
        }
        else {
            loop_ub = 0;
        }

        i0 = offset->size[0] * offset->size[1];
        offset->size[0] = 1;
        offset->size[1] = loop_ub;
        emxEnsureCapacity_real_T(offset, i0);
        if (loop_ub > 0) {
            offset->data[0] = 0.0;
            if (loop_ub > 1) {
                offset->data[loop_ub - 1] = apnd;
                nx = (loop_ub - 1) / 2;
                for (k = 0; k <= nx - 2; k++) {
                    cdiff = (1.0 + (double)k) * d;
                    offset->data[1 + k] = cdiff;
                    offset->data[(loop_ub - k) - 2] = apnd - cdiff;
                }

                if (nx << 1 == loop_ub - 1) {
                    offset->data[nx] = apnd / 2.0;
                }
                else {
                    cdiff = (double)nx * d;
                    offset->data[nx] = cdiff;
                    offset->data[nx + 1] = apnd - cdiff;
                }
            }
        }
    }
```

```
if (window->size[0] < 1) {
    y->size[0] = 1;
    y->size[1] = 0;
}
else {
    i0 = window->size[0];
    i1 = y->size[0] * y->size[1];
    y->size[0] = 1;
    loop_ub = (int)((double)i0 - 1.0);
    y->size[1] = loop_ub + 1;
    emxEnsureCapacity_real_T(y, i1);
    for (i0 = 0; i0 <= loop_ub; i0++) {
        y->data[i0] = 1.0 + (double)i0;
    }
}

emxInit_real_T(&b_y, 2);
k = offset->size[1];
nx = window->size[0];
i0 = b_y->size[0] * b_y->size[1];
b_y->size[0] = y->size[1];
b_y->size[1] = k;
emxEnsureCapacity_real_T(b_y, i0);
loop_ub = y->size[1];
for (i0 = 0; i0 < loop_ub; i0++) {
    for (i1 = 0; i1 < k; i1++) {
        b_y->data[i0 + b_y->size[0] * i1] = y->data[i0];
    }
}

emxInit_real_T(&r0, 2);
emxInit_real_T(&r0_cif, 2);
i0 = r0->size[0] * r0->size[1];
r0->size[0] = nx;
r0->size[1] = offset->size[1];
// do for r0_clif as well
r0_cif->size[0] = nx;
r0_cif->size[1] = offset->size[1];
emxEnsureCapacity_real_T(r0, i0);
emxEnsureCapacity_real_T(r0_cif, i0);
for (i0 = 0; i0 < nx; i0++) {
    loop_ub = offset->size[1];
```

```
        for (i1 = 0; i1 < loop_ub; i1++) {
            r0->data[i0 + r0->size[0] * i1] = offset->data[i1];
            r0_cif->data[i0 + r0_cif->size[0] * i1] = offset->data[i1];
        }
    }

    emxInit_real_T(&r1, 2);
    emxInit_real_T(&r1_cif, 2);
    i0 = r1->size[0] * r1->size[1];
    r1->size[0] = b_y->size[0];
    r1->size[1] = b_y->size[1];

    r1_cif->size[0] = b_y->size[0];
    r1_cif->size[1] = b_y->size[1];
    // do r1_cif as well
    emxEnsureCapacity_real_T(r1, i0);
    emxEnsureCapacity_real_T(r1_cif, i0);
    loop_ub = b_y->size[0] * b_y->size[1];
    for (i0 = 0; i0 < loop_ub; i0++) {
        r1->data[i0] = b_signal[(int)((b_y->data[i0] + r0->data[i0]) + 1.0) - 1];
        r1_cif->data[i0] = b_signal[(int)((b_y->data[i0] + r0_cif->data[i0])) - 1];
    }

    k = offset->size[1];
    i0 = b_y->size[0] * b_y->size[1];
    b_y->size[0] = window->size[0];
    b_y->size[1] = k;
    emxEnsureCapacity_real_T(b_y, i0);
    loop_ub = window->size[0];
#if 0
    //emxFree_real_T(&offset);
#else
    //emxFree_real_T(&offset);
#endif
    for (i0 = 0; i0 < loop_ub; i0++) {
        for (i1 = 0; i1 < k; i1++) {
            b_y->data[i0 + b_y->size[0] * i1] = window->data[i0];
        }
    }

    emxFree_real_T(&window);
    i0 = r0->size[0] * r0->size[1];
    r0->size[0] = r1->size[0];
```

```
    r0->size[1] = r1->size[1];
    emxEnsureCapacity_real_T(r0, i0);
    loop_ub = r1->size[0] * r1->size[1];
    for (i0 = 0; i0 < loop_ub; i0++) {
        r0->data[i0] = r1->data[i0] * b_y->data[i0];
        r0_cif->data[i0] = r1_cif->data[i0] * b_y->data[i0];
    }
#if 0
    //emxFree_real_T(&b_y);
#else
    emxFree_real_T(&b_y);
#endif
    emxFree_real_T(&r1);
    emxInit_creal_T(&STFT, 2);
    emxInit_creal_T(&STFTfreqdelpos, 2);
    emxInit_creal_T(&STFTdelpos, 2);
    emxInit_creal_T(&CIFpos, 2);
    emxInit_creal_T(&LGDpos, 2);
    emxInit_creal_T(&time_t, 2);

    //matrixTranslate(r0->data, r0->size[0], r0->size[1], r0->data);
    displayMatrix(r0, "r0 MATRIX", false, 10);
    fft_emx(r0, fftn, STFT);
    //displayMatrix(STFT, "STFT MATRIX", false, 10);
    fft_emx(r0_cif, fftn, STFTfreqdelpos);
    fft_emx(r0_cif, fftn, STFTdelpos);
    // dummy calculation, so that proper memory is allocated for CIF pos
    fft_emx(r0, fftn, CIFpos);
    fft_emx(r0, fftn, LGDpos);
    fft_emx(r0, fftn, time_t);
    emxFree_real_T(&r0);
    //validate
    int s1 = STFT->size[0];
    int s2 = STFT->size[1];
    printf("DEBUG\n");
    int t = 0;
    for (int j = 0; j < s2; j++)
    {
        if (j == 0)
            t = s1 - 1;
        else
            t = ((s1 - 1) * (j + 1)) + j;
```

```
        STFTfreqdelpos->data[j].re = STFT->data[t].re;
        STFTfreqdelpos->data[j].im = STFT->data[t].im;
        //  printf("%d\t %lf\t %lf\n", j, STFTfreqdelpos->data[j].re, STFTfreqdelpos->data[j
    }

    //printf("next line\n");
    int cnt = s2;
    for (int j = 0; j < s1 - 1; j++)
    {
        for (int i = 0; i < s2; i++)
        {
            t = s1 * i + j;
            STFTfreqdelpos->data[cnt].re = STFT->data[t].re;
            STFTfreqdelpos->data[cnt].im = STFT->data[t].im;
            //printf("%d\t %lf\t %lf\n", i, STFTfreqdelpos->data[cnt].re, STFTfreqdelpos->da
            cnt++;
        }
    }

    if (rt_remd_snf(fftn, 2.0) == 1.0) {
        d0 = (fftn - 1.0) / 2.0;
    }
    else {
        d0 = fftn / 2.0;
    }

    cdiff = Fs * (d0 - 1.0) / fftn;
    if (high > cdiff) {
        high = cdiff;
    }

    cdiff = rt_roundd_snf(low / Fs * fftn);
    if (cdiff == 0.0) {
        cdiff = 1.0;
    }

    ndbl = rt_roundd_snf(high / Fs * fftn);
    if (cdiff > ndbl) {
        i0 = -1;
        i1 = -1;
    }
    else {
        i0 = (int)cdiff - 2;
```

```
        i1 = (int)ndbl - 1;
    }

    emxInit_creal_T(&STFTpos, 2);
    loop_ub = STFT->size[1];
    nx = STFTpos->size[0] * STFTpos->size[1];
    m = i1 - i0;
    STFTpos->size[0] = m;
    STFTpos->size[1] = loop_ub;
    emxEnsureCapacity_creal_T(STFTpos, nx);
    for (i1 = 0; i1 < loop_ub; i1++) {
        for (nx = 0; nx < m; nx++) {
            STFTpos->data[nx + STFTpos->size[0] * i1] = STFT->data[((i0 + nx) +
                STFT->size[0] * i1) + 1];
        }
    }
    /*
    here STFTPos is available, so start computing CIFPos as well
    below is the matlab code
    C = STFTpos .* conj(STFTdelpos);

    L = STFTpos .* conj(STFTfreqdelpos);
    argC = mod(angle(C),2*pi);
    CIFpos = ((Fs/delay).* argC)./(2.*pi);
    */
    /*
   Find C
    1. compute STFTdelpos
    2. compute conj of STFTdelpos
    3. multiply STFTpos with step2 ---> C computed
    */
    // compute STFTdel


    /*
    Compute C:
          1. first conj(STFTdelpos)
          2. STFTpos . * conj(STFTdelpos)
     */
     // computing conj(STFTdelpos)
    int size_0 = STFTdelpos->size[0];
    int size_1 = STFTdelpos->size[1];
    for (int i = 0; i < size_0 * size_1; i++)
```

```
    {
        //conj(STFTdelpos)
        double im1 = -STFTdelpos->data[i].im;
        STFTdelpos->data[i].im = im1;
        // conj(STFTfreqdelpos);
        double im2 = -STFTfreqdelpos->data[i].im;
        STFTfreqdelpos->data[i].im = im2;
    }
#if 0
    int cnt1 = 0;
    for (int i = 0; i < size_0; i++)
    {

        for (int j = 0; j < size_1; j++)
        {

            printf("re = %lf\t", STFTfreqdelpos->data[cnt1].re);
            printf("im = %lf\t", STFTfreqdelpos->data[cnt1].im);
            cnt1++;
        }
        printf("\n");
    }
#endif
    //compute STFTpos . * conj(STFTdelpos)
    printf(" %lf\t %lf\n", STFTpos->data[0].re, STFTpos->data[0].im);
    printf(" %lf\t %lf\n", STFTpos->data[1].re, STFTpos->data[1].im);
    printf(" %lf\t %lf\n", STFTpos->data[2].re, STFTpos->data[2].im);
    printf(" %lf\t %lf\n", STFTpos->data[3].re, STFTpos->data[3].im);



    printf(" %lf\t %lf\n", STFTdelpos->data[0].re, STFTdelpos->data[0].im);
    printf(" %lf\t %lf\n", STFTdelpos->data[1].re, STFTdelpos->data[1].im);
    printf(" %lf\t %lf\n", STFTdelpos->data[2].re, STFTdelpos->data[2].im);
    printf(" %lf\t %lf\n", STFTdelpos->data[3].re, STFTdelpos->data[3].im);
    size_0 = STFTpos->size[0];
    size_1 = STFTpos->size[1];
    t = 0;
#if 0
    for (int j = 0; j < s2; j++)
    {
        for (int i = 0; i < s1; i++)
        {
```

```
            t = s1 * i + j;
            double im_delpos_im = STFTdelpos->data[t].im;
            double im_pos_im = STFTpos->data[t].im;

            double im = im_delpos_im * im_pos_im;
            printf("%lf\n", im);
            double im_delpos_re = STFTdelpos->data[t].re;
            double im_pos_re = STFTpos->data[t].re;

            double re = im_delpos_re * im_pos_re;
            printf("%lf\n", re);
            STFTdelpos->data[t].im = im;
            STFTdelpos->data[t].re = re;
            // printf("%d\t %lf\t %lf\n", i, STFTfreqdelpos->data[cnt].re, STFTfreqdelpos->d

        }
    }
#endif
#if 1
    /*
    * (a+bi) .* (c + di)
    * = (a*c) + (a * di) + (bi * c) - (b*d)
    */
    for (int i = 0; i < size_0 * size_1; i++)
    {
        double im_delpos_im = STFTdelpos->data[i].im;
        double im_pos_im = STFTpos->data[i].im;

        double im_delpos_re = STFTdelpos->data[i].re;
        double im_pos_re = STFTpos->data[i].re;


        // b*d
        double re_2 = (im_delpos_im * im_pos_im);
        //a*c
        double re_1 = im_delpos_re * im_pos_re;
        //a*di
        double im_1 = im_delpos_re * im_pos_im;
        double im_2 = im_delpos_im * im_pos_re;
        //  printf("%lf\n", re_1 - re_2);
         // printf("%lf\n", im_1 + im_2);
        STFTdelpos->data[i].im = im_1 + im_2;
        STFTdelpos->data[i].re = re_1 - re_2;
```

```
    }
#endif
    //compute STFTpos . * conj(STFTfreqdelpos)
    /*compute L*/

    int cnt2 = 0;
    for (int i = 0; i < size_0; i++)
    {
        for (int j = 0; j < size_1; j++)
        {

            double im_delpos_im = STFTfreqdelpos->data[cnt2].im;
            double im_pos_im = STFTpos->data[j * size_0 + i].im;

            double im_delpos_re = STFTfreqdelpos->data[cnt2].re;
            double im_pos_re = STFTpos->data[j * size_0 + i].re;

            // printf("STFTpos->data[i].re = %lf\t", STFTpos->data[i].re);
             //printf("STFTpos->data[i].im = %lf\n", STFTpos->data[i].im);
             //printf("STFTfreqdelpos->data[i].re = %lf\t", STFTfreqdelpos->data[i].re);
             //printf("STFTfreqdelpos->data[i].im = %lf\n", STFTfreqdelpos->data[i].im);
             // b*d
            double re_2 = (im_delpos_im * im_pos_im);
            //a*c
            double re_1 = im_delpos_re * im_pos_re;
            //a*di
            double im_1 = im_delpos_re * im_pos_im;
            double im_2 = im_delpos_im * im_pos_re;
            //  printf("%lf\n", re_1 - re_2);
             // printf("%lf\n", im_1 + im_2);
            STFTfreqdelpos->data[cnt2].im = im_1 + im_2;
            STFTfreqdelpos->data[cnt2].re = re_1 - re_2;
            //printf("STFTfreqdelpos->data[i].re = %lf\t", STFTfreqdelpos->data[cnt].re);
            //printf("STFTfreqdelpos->data[i].im = %lf\n", STFTfreqdelpos->data[cnt].im);
            cnt2++;
        }
    }
    int delay = 1;

    for (int i = 0; i < size_0 * size_1; i++)
    {
        CIFpos->data[i].re = 0.0;
        CIFpos->data[i].im = 0.0;
```

```
}

for (int i = 0; i < size_0 * size_1; i++)
{
    LGDpos->data[i].re = 0.0;
    LGDpos->data[i].im = 0.0;
}

for (int i = 0; i < size_0 * size_1; i++)
{
    time_t->data[i].re = 0.0;
    time_t->data[i].im = 0.0;
}
for (int i = 0; i < size_0 * size_1; i++)
{
    double x = STFTdelpos->data[i].re;
    double y = STFTdelpos->data[i].im;
    double angle = (double)atan2(y, x);
    //if(i >= 3365)
   // printf("angle = %lf\n", angle);
    double mod_v = fmod(angle, (2 * M_PI));
    double temp = (mod_v * (Fs / delay));
    temp = temp / (2 * M_PI);
    CIFpos->data[i].re = temp;
    CIFpos->data[i].im = 0.0;
}

for (int i = 0; i < size_0 * size_1; i++)
{
    double x = STFTfreqdelpos->data[i].re;
    double y = STFTfreqdelpos->data[i].im;
    double angle = (double)atan2(y, x);
    double mod_v = fmod(angle, (-2 * M_PI));
    double temp = -(mod_v * (fftn / Fs));
    temp = temp / (2 * M_PI);
    LGDpos->data[i].re = temp;
    LGDpos->data[i].im = 0.0;

    // printf("angle = %lf\n", LGDpos->data[i].re);
}

// compute t
double step = win_size - overlap;
```

```
// t = (offset + win_size / 2) . / Fs;
int s0_offset = offset->size[0];
int s1_offset = offset->size[1];
time_offset->size[0] = s0_offset;
time_offset->size[1] = s1_offset;
emxEnsureCapacity_real_T(time_offset, s0_offset * s1_offset);
for (int i = 0; i < s0_offset * s1_offset; i++)
{
    printf("%lf\n", offset->data[i]);
    time_offset->data[i] = ((offset->data[i] + (win_size / 2))) / Fs;
    printf("%lf\n", time_offset->data[i]);
}
int tmp_cnt = 0;
double y_t = 0.0031;// ((win_size / 2) - 1) / Fs;
for (int i = 0; i < size_0 * size_1; i++)
{

    // printf("LGDpos->data[i].re = %lf\n", LGDpos->data[i].re);
    time_t->data[i].re = LGDpos->data[i].re + time_offset->data[tmp_cnt++] - y_t;
    printf("time_t->data[i].re = %lf\n", time_t->data[i].re);
    if (tmp_cnt == size_1)
        tmp_cnt = 0;
}
i0 = STFT->size[1];
nx = m * i0;
i0 = STFT->size[1];
i1 = STFTmag->size[0] * STFTmag->size[1];
STFTmag->size[0] = m;
STFTmag->size[1] = i0;
CIFPos_mag->size[0] = m;
CIFPos_mag->size[1] = i0;
time->size[0] = m;
time->size[1] = i0;
emxEnsureCapacity_real_T(STFTmag, i1);
for (k = 0; k < nx; k++) {
    STFTmag->data[k] = rt_hypotd_snf(STFTpos->data[k].re, STFTpos->data[k].im);
}

emxFree_creal_T(&STFTpos);
emxFree_creal_T(&STFT);
m = STFTmag->size[0];
loop_ub = STFTmag->size[1];
unnamed_idx_1 = (unsigned int)STFTmag->size[1];
```

```
i0 = y->size[0] * y->size[1];
y->size[0] = 1;
y->size[1] = (int)unnamed_idx_1;
emxEnsureCapacity_real_T(y, i0);
if (STFTmag->size[1] >= 1) {
    for (nx = 0; nx < loop_ub; nx++) {
        y->data[nx] = STFTmag->data[STFTmag->size[0] * nx];
        for (k = 2; k <= m; k++) {
            cdiff = y->data[nx];
            ndbl = STFTmag->data[(k + STFTmag->size[0] * nx) - 1];
            if ((!rtIsNaN(ndbl)) && (rtIsNaN(cdiff) || (cdiff < ndbl))) {
                y->data[nx] = STFTmag->data[(k + STFTmag->size[0] * nx) - 1];
            }
        }
    }
}

loop_ub = y->size[1];
if (y->size[1] <= 2) {
    if (y->size[1] == 1) {
        cdiff = y->data[0];
    }
    else if ((y->data[0] < y->data[1]) || (rtIsNaN(y->data[0]) && (!rtIsNaN
    (y->data[1])))) {
        cdiff = y->data[1];
    }
    else {
        cdiff = y->data[0];
    }
}
else {
    if (!rtIsNaN(y->data[0])) {
        nx = 1;
    }
    else {
        nx = 0;
        k = 2;
        exitg1 = false;
        while ((!exitg1) && (k <= y->size[1])) {
            if (!rtIsNaN(y->data[k - 1])) {
                nx = k;
                exitg1 = true;
            }
```

```
            else {
                k++;
            }
        }
    }

    if (nx == 0) {
        cdiff = y->data[0];
    }
    else {
        cdiff = y->data[nx - 1];
        i0 = nx + 1;
        for (k = i0; k <= loop_ub; k++) {
            if (cdiff < y->data[k - 1]) {
                cdiff = y->data[k - 1];
            }
        }
    }
}

emxFree_real_T(&y);
i0 = STFTmag->size[0] * STFTmag->size[1];
i1 = STFTmag->size[0] * STFTmag->size[1];
emxEnsureCapacity_real_T(STFTmag, i1);
loop_ub = i0 - 1;
for (i0 = 0; i0 <= loop_ub; i0++) {
    STFTmag->data[i0] /= cdiff;
}

nx = STFTmag->size[0] * STFTmag->size[1];
for (k = 0; k < nx; k++) {
    STFTmag->data[k] = log10(STFTmag->data[k]);
}

i0 = STFTmag->size[0] * STFTmag->size[1];
i1 = STFTmag->size[0] * STFTmag->size[1];
emxEnsureCapacity_real_T(STFTmag, i1);
emxEnsureCapacity_real_T(CIFPos_mag, i1);
emxEnsureCapacity_real_T(time, i1);
loop_ub = i0 - 1;
for (i0 = 0; i0 <= loop_ub; i0++) {
    STFTmag->data[i0] *= 20.0;
    CIFPos_mag->data[i0] = CIFpos->data[i0].re;
```

```
        time->data[i0] = time_t->data[i0].re;
    }
}
```

Appendix B

Appendix: xkl formant saving Module

```
#include "xspec_util.h"
#include "xinfo.h"
#include "xklspec.h"
#include "spectrum.h"
#include "textgrid.h"
typedef struct _FILES {
    int* indexCmd;
    int  icount;
    char markString[100]
} FILES, * LPFILES;
typedef struct _TIMES {
    double* times;
    int     icount;
} TIMES, * LPTIMES;
//print test info
void printSpectro(XSPECTRO* spectro, const char* sourceStr)
{
    int n3, n;
    int tmpfreq = 0;
    float winms = (float)spectro->sizwin / spectro->spers * 1000.0;
    printf("\n\nspectro->type_spec:%d win:%.1fms %s\n", spectro->type_spec, winms, sourceStr
    printf("\n\n\t\t\t print spectro->fltr:\n\n\tn\tdB\tn\tdB\tn\tdB\n");
    n3 = (spectro->lenfltr + 2) / 3;
    for (n = 0; n < n3; n++) {
        printf("\t%3d\t%4.1f  ", n + 1, (float)spectro->fltr[n] / 10.0);
        printf("\t%3d\t%4.1f  ", n + n3 + 1, (float)spectro->fltr[n + n3] / 10.0);
        if ((n + n3 + n3) < spectro->lenfltr) {
            printf("\t%3d\t%4.1f\n", n + n3 + n3 + 1,
                (float)spectro->fltr[n + n3 + n3] / 10.0);

        }
```

```
        else printf("\n");
    }
    printf("\n\n\t\t\t print spectro->dftmag:\n\n\tn\tdB\tn\tdB\tn\tdB\n");
    n3 = (spectro->sizfft + 2) / 3;
    for (n = 0; n < n3; n++) {
        printf("\t%3d\t%4.6f  ", n + 1, spectro->dftmag[n]);
        printf("\t%3d\t%4.6f  ", n + n3 + 1, spectro->dftmag[n + n3]);
        if ((n + n3 + n3) < spectro->dftmag) {
            printf("\t%3d\t%4.6f\n", n + n3 + n3 + 1,
                (float)spectro->dftmag[n + n3 + n3] / 10.0);

        }
        else printf("\n");
    }
    if (!spectro->type_spec)return;
    printf("\n\n\t\t\t print FREQ AMP\n\n\tFREQ\tAMP\n");
    for (n = 0; n < spectro->nforfreq; n++)
    {
        tmpfreq = spectro->forfreq[n];
        if (tmpfreq < 0)
        {
            tmpfreq = -tmpfreq;
        }
        printf("\t%5d\t%d\n", tmpfreq, (int)(spectro->foramp[n] / 10));
    }
}
void writeFreqAmp(XSPECTRO* spectro, FILE* file)
{
    int n;
    int tmpfreq = 0;

    fprintf(file,"\n\nFREQ AMP %0.1fms\n\n\tFREQ\tAMP\n", spectro->savetime);

    for (n = 0; n < spectro->nforfreq; n++)
    {
        tmpfreq = spectro->forfreq[n];
        if (tmpfreq < 0)
        {
            tmpfreq = -tmpfreq;
        }
        fprintf(file,"\t%5d\t%d\n", tmpfreq, (int)(spectro->foramp[n] / 10));
    }
}
```

```
void getDevSize(int* w, int* h, int* fw, int* fh)
{
    Display* display;
    char* display_name = NULL;
    int screen_num;
    XFontStruct* font;
    if ((display = XOpenDisplay(display_name)) == (Display*)NULL) {

        fprintf(stderr, "ERROR trying to open DISPLAY \n");
        exit(0);
    }
    screen_num = DefaultScreen(display);
    *w = DisplayWidth(display, screen_num);
    *h = DisplayHeight(display, screen_num);

    font = XQueryFont(display,
        XGContextFromGC(DefaultGC(display, screen_num)));

    *fw = font->max_bounds.rbearing - font->min_bounds.lbearing;
    *fh = font->max_bounds.ascent + font->max_bounds.descent;
    XCloseDisplay(display);
}
int getCmdTimeParam(char** times, int timeNum, float* outTime)
{
    for (int i = 0; i < timeNum; i++)
    {
        if (strspn(times[i], "0123456789.") == strlen(times[i]))
        {
            outTime[i] = atof(times[i]);
        }
        else
        {
            printf("%s not number!\n", times[i]);
            return 1;
        }
    }
    return 0;
}
int checkTimeParam(XSPECTRO* spectro, double* times, int timeCout)
{
    double maxTime = (double)spectro->totsamp / spectro->spers * 1000;
    for (int i = 0; i < timeCout; i++)
    {
```

```
        if (times[i]<1 || times[i]>maxTime)
        {
            printf("intput time %0.1f invalid.\n%s maxtime:%0.1f total number of \
                samples:%d \n", times[i], spectro->wavename, maxTime, spectro->totsamp);
            return -1;
        }
    }
    return 0;
}
int IsValidFile(char* strFile, const char* extName)
{
    int iflen = strlen(strFile);
    int iextlen = strlen(extName);
    char* ptmpPtr = strFile;
    if (iflen < iextlen)
        return 1;
    ptmpPtr += iflen - iextlen;
    if (strcasecmp(ptmpPtr, extName) == 0)
    {
        FILE* f = fopen(strFile, "r");
        if (!f)
        {
            printf("file:%s not exists!\n", strFile);
            return -1;
        }
        fclose(f);
        return 0;
    }
    return 1;
}
int ExtractParamFromCmdLine(int argc, char** argv,
    LPFILES wavFiles,
    LPFILES textGridFiles,
    LPTIMES cmdTimes)
{
    wavFiles->indexCmd = (int*)malloc(sizeof(int) * (argc));
    textGridFiles->indexCmd = (int*)malloc(sizeof(int) * (argc));
    cmdTimes->times = (double*)malloc(sizeof(double) * (argc));
    int iwavIndex = 0;
    int iTextGridIndex = 0;
    int itimesIndex = 0;
    for (int i = 1; i < argc; i++)
    {
```

```
        if (IsValidFile(argv[i], ".wav") == 0)
        {
            wavFiles->indexCmd[iwavIndex++] = i;
            wavFiles->icount++;
        }
        else if (IsValidFile(argv[i], ".TextGrid") == 0)
        {
            textGridFiles->indexCmd[iTextGridIndex++] = i;
            if (i + 1 < argc)
            {
                strcpy(textGridFiles->markString, argv[++i]);
            }
            else
            {
                strcpy(textGridFiles->markString,"V");
            }
            textGridFiles->icount++;
        }
        else
        {
            if (strspn(argv[i], "0123456789.") == strlen(argv[i]))
            {
                //number
                cmdTimes->times[itimesIndex++] = atof(argv[i]);
                cmdTimes->icount++;
            }
        }
    }
    if (iwavIndex && (iTextGridIndex || itimesIndex))
    {
        return 0;
    }
    return -1;
}
int ProcessOneWavFile(XSPECTRO* spectro,char* wavefile, double* times, int imaxTimeCout,FILE
{
    add_spectro(spectro, "xkl_defs.dat", wavefile);
    if (checkTimeParam(spectro, times, imaxTimeCout))
    {
        return 1;
    }
    fprintf(fp, "\n%s\n", wavefile);
    for (int i = 0; i < imaxTimeCout; i++)
```

```
    {
        spectro->saveindex = (times[i] * spectro->spers / 1000.0) + .5;
        spectro->savetime = (float)spectro->saveindex / spectro->spers * 1000;
        new_spectrum(spectro);
        getform(spectro);
        writeFreqAmp(spectro, fp);
    }
    return 0;
}
int enterConsoleApp(int argc, char** argv)
{

    getcwd(curdir, DIR_LENGTH);
    printf("enterConsoleApp %s\n", curdir);
    int imaxTimeCout=0;
    double* times = NULL;
    int devwidth, devheight;
    int wchar, hchar;
    XSPECTRO* spectro;
    FILE* fp=NULL;
    FILES fwavs;
    FILES ftextGrid;
    TIMES ttimes;
    memset(&fwavs, 0x0, sizeof(FILES));
    memset(&ftextGrid, 0x0, sizeof(FILES));
    memset(&ttimes, 0x0, sizeof(TIMES));
    if (argc < 3)
    {
        printf("usage: ./xkl-3.2 filename time1...timeN\n");
        return 0;
    }
    if (ExtractParamFromCmdLine(argc, argv, &fwavs, &ftextGrid, &ttimes))
    {
        printf("param error!\n");
        printf("usage: ./xkl-3.2 name.wav time1...timeN\n");
        printf("usage: ./xkl-3.2 name.wav name.textGrid\n");
        return 0;
    }
    if (ftextGrid.icount)
    {
        //A textGrid file is supported
        times = ParserTimesFromFile(argv[ftextGrid.indexCmd[0]], &imaxTimeCout, ftextGrid.ma
        for (int i = 0; i < imaxTimeCout; i++)
```

```
        {
            //mec to ms
            times[i] *= 1000;
            printf("%0.1f\n", times[i]);
        }
        free(ftextGrid.indexCmd);
    }
    else
    {
        //cmdline time param!
        times = ttimes.times;
        imaxTimeCout = ttimes.icount;
    }
    for (int s = 100; s < SIZCBSKIRT; s++)
        cbskrt[s] = 0.975 * cbskrt[s - 1];
    getDevSize(&devwidth, &devheight, &wchar, &hchar);
    fp = fopen("outfreqamp.txt", "w");
    if (!fp)
    {
        printf("outfreqamp.txt error!\n");
        free(fwavs.indexCmd);
        free(times);
        return 0;
    }
    for (int iwavs = 0; iwavs < fwavs.icount; iwavs++)
    {
        spectro = (XSPECTRO*)malloc(sizeof(XSPECTRO));
        memset(spectro, 0x0, sizeof(XSPECTRO));
        strcpy(spectro->synDefPath, argv[0]);
        spectro->swap = 0;
        spectro->spectrogram = 0;
        spectro->devwidth = devwidth;
        spectro->devheight = devheight;
        spectro->wchar = wchar;
        spectro->hchar = hchar;
        ProcessOneWavFile(spectro, argv[fwavs.indexCmd[iwavs]], times, imaxTimeCout,fp);
        free(spectro);
    }
    if (fp)
    {
        fclose(fp);
    }
    free(fwavs.indexCmd);
```

```
    free(times);
    return 0;
}
```

Appendix C

Appendix: Automatic Vowel Landmarks Detection

```
import librosa
import numpy as np
import os
import pandas as pd
import tgt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

#audio_folder =

audio_folder = "C:path"
textgrid_folder = "C:path"

frame_length = 10  # in ms
final_data = []

for filename in os.listdir(audio_folder):
    if filename.endswith(".wav"):
        y, sr = librosa.load(os.path.join(audio_folder, filename), sr=10000)
        frame_length_samples = int(sr * frame_length / 1000)

        total_frames = int(len(y) / frame_length_samples)
        energy = np.zeros(total_frames)
```

```
mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=128,
                                                 hop_length=frame_length_samples, po
mel_spectrogram_db = librosa.power_to_db(mel_spectrogram, ref=np.max)

for i in range(total_frames):
    frame = y[i * frame_length_samples:(i + 1) * frame_length_samples]
    frame_energy = np.sum(np.square(frame))
    energy[i] = frame_energy

time_interval = np.array([i * frame_length / 1000 for i in range(total_frames)])

# Read LM tier from textgrid file
textgrid_filename = filename.replace('.wav', '.TextGrid')
textgrid_file = os.path.join(textgrid_folder, textgrid_filename)
tg = tgt.io.read_textgrid(textgrid_file)
lm_tier = tg.get_tier_by_name("LM")

# Create a list to store LM tier information for each frame
lm_info = []
for i in range(total_frames):
    frame_start_time = time_interval[i]
    frame_end_time = time_interval[i] + frame_length / 1000
    found_v = False
    for annotation in lm_tier.annotations:
        if annotation.time >= frame_start_time and annotation.time <= frame_end_time
            if annotation.text == "V":
                lm_info.append("V")
                found_v = True
                break
    if not found_v:
        lm_info.append("No_Landmark")

n = len(lm_info)
for i in range(n):
    if lm_info[i] == "V":
        if i-1 >= 0:
            lm_info[i-1] = "V"
        if i-2 >= 0:
            lm_info[i-2] = "V"
        if i+1 < n:
            lm_info[i+1] = "V"
        if i+2 < n:
```

```
                    lm_info[i+2] = "V"

        for i in range(total_frames):
            final_data.append([energy[i], mel_spectrogram_db[:, i], lm_info[i]])

# Convert final_data to DataFrame for further processing
df = pd.DataFrame(final_data, columns=["Energy", "MelSpectrogram", "Label"])

# Prepare data for LSTM
X_energy = np.array(df["Energy"].tolist()).reshape(-1, 1)
X_mel = np.array(df["MelSpectrogram"].tolist())
X = np.hstack((X_energy, X_mel))

# Standardize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Encode labels
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df["Label"])

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Reshape data for LSTM (samples, timesteps, features)
timesteps = 1
X_train = X_train.reshape((X_train.shape[0], timesteps, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], timesteps, X_test.shape[1]))

# Build LSTM model
model = Sequential()
model.add(LSTM(64, input_shape=(timesteps, X_train.shape[2])))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy:.4f}')
```

```
# Generate predictions
y_pred = (model.predict(X_test) > 0.5).astype("int32")

# Classification report
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot accuracy and loss over epochs
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Model Loss')

plt.show()

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classe
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
```

```
print(f'Test Accuracy: {accuracy:.4f}')

# Generate predictions
y_pred = (model.predict(X_test) > 0.5).astype("int32")

# Classification report
report = classification_report(y_test, y_pred, target_names=label_encoder.classes_)
print(report)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot accuracy and loss over epochs
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Model Loss')

plt.show()

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classe
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```