HELLENIC MEDITERRANEAN UNIVERSITY

SCHOOL OF MUSIC AND OPTOACOUSTIC TECHNOLOGIES

Department of Music Technologies and Acoustics



Master Thesis

Web-Based Automatic Analysis, Visualization and Curation of Musical Content

Neoklis Mimidis

Supervisor: ***Chrisoula Alexandraki***

Assistant Professor

Rethymno, February 2024

Copyright

Copyright © Neoklis Mimidis, 2024

Πνευματικά Δικαιώματα© Νεοκλής Μιμίδης, 2024

All rights reserved.

Με επιφύλαξη παντός δικαιώματος.

The approval of the thesis by the Department of Music Technology & Acoustics of the Hellenic Mediterranean University does not necessarily imply acceptance of the author's views on behalf of the Department.

Η έγκριση της πτυχιακής εργασίας από το Τμήμα Μουσικής Τεχνολογίας & Ακουστικής του Ελληνικού Μεσογειακού Πανεπιστημίου δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα εκ μέρους του Τμήματος.

Abstract

The present thesis focuses on the development of an innovative web application aiming at enhancing collaboration and user engagement in music education through automatic analysis, visualization, and curation of musical content. By integrating state-of-the-art achievements in Music Information Retrieval (MIR), it allows users to automatically retrieve and subsequently process the rhythmic and harmonic content of sound recordings, which in the target application are sought as musical accompaniment to the teacher's and/or student's performance. The application can be used both for asynchronous educational tasks, e.g., preparation of educational materials by the teacher, student self-practice, etc., as well as for synchronous collaborative online music lessons.

Keywords: online music education, play-along music practice, computer supported cooperative work, intelligent music signal analysis, audio beat tracking, audio chord estimation

Περίληψη

Η παρούσα διπλωματική εργασία, επικεντρώνεται στην ανάπτυξη μιας καινοτόμου διαδικτυακής εφαρμογής η οποία στοχεύει στην ενίσχυση της συνεργασίας κατά τη μουσική εκπαίδευση μέσω της αυτόματης ανάλυσης, απεικόνισης και διαχείρισης μουσικού περιεχομένου. Μέσω της ενσωμάτωσης σύγχρονων αλγορίθμων Ανάκτησης Μουσικής Πληροφορίας (ΑΜΠ), η εφαρμογή διευκολύνει την διαδραστική εκμάθηση και διδασκαλία της μουσικής, επιτρέποντας στους χρήστες να αναλύουν και να επεξεργάζονται το ρυθμικό και αρμονικό περιεχόμενο ηχογραφήσεων, οι οποίες στην εφαρμογή στόχου χρησιμοποιούνται ως συνοδεία της μουσικής ερμηνείας του δασκάλου ή/και του μαθητή. Η εφαρμογή δύναται να χρησιμοποιηθεί τόσο σε πλαίσιο ασύγχρονης εκπαίδευσης, π.χ. προετοιμασία εκπαιδευτικού υλικού από το δάσκαλο, εξάσκηση μαθητή κ.α. όσο και σε σύγχρονες συνεδρίες διαδικτυακών μαθημάτων.

Λέξεις-κλειδιά: διαδικτυακή μουσική εκπαίδευση, πρακτική μουσικής εκτέλεσης με συνοδεία, συνεργατική εργασία υποστηριζόμενη από υπολογιστή, ευφυής ανάλυση μουσικού σήματος, παρακολούθηση ηχητικού ρυθμού, εκτίμηση συγχορδιών

Acknowledgements

This thesis is a result not only of the main author's efforts but also of a group of people who contributed to its production, and I would like to thank them.

Firstly, I want to thank my supervisor Assistant Professor Chrisoula Alexandraki for her valuable guidance, continuous help, encouragement, inspiration and understanding during all phases of my master thesis. Additionally, I am deeply thankful to her for providing me with the opportunity to participate in the MusiCoLab research program, where she served as the principal investigator. This experience was pivotal, enabling me to improve my research skills, collaborate and exchange ideas on various project components, and contribute to the creation of the final intended research platform. Notably, my master's thesis was an integral part of this platform, offering me the unique chance to evaluate the developed application in real-world scenarios.

During my participation in the MusiCoLab research program, I had the opportunity to meet and work with its members, whose support and guidance I deeply appreciate. More importantly, I would like to thank Giannis Viglis, Alexandros Nousias, Dimitris Milios, Konstantinos Tsioutas for their co-development efforts on the Play-Along-Together (PAT) application, which exceeded the functionalities presented in this thesis. I would also like to thank, Michalis Kalochristianakis for his guidance on server-side asynchronous communication of the application and for providing me with valuable programming insights. Additionally, I am grateful to all the participants of the PAT application experiments, whose feedback was essential for identifying improvements and future directions.

Finally, I can't express enough gratitude for the support of my parents Maria and Konstantinos, as well as my brothers Panagiotis and Manolis, who patiently and unwaveringly offered the necessary moral support for the completion of my thesis.

Table of Contents

Abstract ..... **iii**

Acknowledgements ..... **iv**

Table of Contents ..... **v**

List of Figures ..... **vii**

List of Tables ..... **xi**

1 Introduction **1**

1.1 Objective and Focus 1

1.2 Thesis structure 2

2 Related Work **3**

2.1 Music Information Retrieval 3

2.2 MIR Programming Libraries 5

2.3 Audio Beat Tracking 7

2.3.1 Task Description *9*

2.3.2 Early Computational Methodologies *10*

2.3.3 State of the art approaches *12*

2.3.4 Overview of Beat Tracking Applications *14*

2.4 Audio Chord Estimation 15

2.4.1 Task Description *17*

2.4.2 Early Computational Methodologies *18*

2.4.3 State of the art approaches *35*

2.4.4 Overview of Chord Analysis Applications *39*

2.5 Annotation Formats 41

2.6 Discussion 45

3 Implementation **49**

3.1 General Architecture 49

3.2 Content Analysis 50

3.2.1 Command-line Configuration *51*

3.2.2 Script Workflow *52*

3.2.3 Chord Analysis ... *53*
3.2.4 Beat Analysis ... *53*
3.2.5 Heuristically Driven Chord-Beat Alignment ... *54*
3.3 User Interaction and Graphical User Interface ... 55
3.3.1 Audio Player ... *56*
3.3.2 Context-aware Toolbar ... *61*
3.3.3 Annotation Editing ... *64*
3.3.4 Additional Features and System Integration ... *68*
3.4 Using the PAT Application in Music Education ... 69
3.4.1 Asynchronous Student Practice ... *69*
3.4.2 Asynchronous Assignments combined with Synchronous Lessons ... *69*
3.4.3 Synchronous Group Improvisation Session on a Jazz Piece ... *70*
4 Evaluation ... 71
4.1 Evaluation of algorithmic output ... 71
4.1.1 Beats ... *72*
4.1.2 Chords ... *76*
4.1.3 Chords at Beats ... *81*
4.2 Evaluation of usability ... 83
4.3 Consolidation ... 86
5 Conclusions ... 87
5.1 Summary ... 87
5.2 Contributions and Outcomes ... 88
5.3 Future Perspectives ... 88
References ... 90
Appendix A ... 95
Availability and software dependencies ... 95
Availability ... *95*
Software dependencies ... *95*
Appendix B ... 97
Chord symbols and encodings ... 97

List of Figures

Fig. 2.1 Music Information Retrieval across different domains 3
Fig. 2.2 ADSR (Attack Decay Sustain Release) envelope with onset and transient 7
Fig. 2.3 Correlation of beat terminology with staff notation and waveform representation 8
Fig. 2.4 The process of audio onset detection 10
Fig. 2.5 Tempogram of click track with tempo increase from 100 to 140 BPM 12
Fig. 2.6 Basic triads chords in the key of C major 15
Fig. 2.7 C major chord presented in both closed position and arpeggiated 16
Fig. 2.8 Early computational methodologies in ACE 18
Fig. 2.9 Waveform, spectrum, and spectrogram visualizations of an audio file 20
Fig. 2.10 Spectrogram matrix: a tabular representation of frequency and time 22
Fig. 2.11 Magnitude spectrograms of the chromatic scale on piano: a) linear frequency scale; b) logarithmic mel scale; c) pitch-based scale; d) Constant-Q transform scale; 23
Fig. 2.12 Semitone spectrogram of the chromatic scale using IIR filters by calculating for each filter band the STMSP 24
Fig. 2.13 Harmonic-Percussive Source Separation (HPSS) as a noise removal process in ACE 25
Fig. 2.14 Range specific chromagrams (top to bottom: tremble chroma and bass chroma) ... 27
Fig. 2.15 STFT and CQT chromagrams of the chromatic scale on piano, as seen in Fig. 2.11 a) All the notes from the chromatic scale [A0 to C8], where the Red rectangle is the note C3; b) Zoomed-in chromagrams, where the Blue rectangle is 1 frame from the CQT chromagram of the note C3; The Blue rectangle frame: c) in PCP data structure vector array mathematical representation, d) graphically represented with a bar chart 28
Fig. 2.16 Chromagrams filtered by various pre-filtering techniques of the same audio file: (a) Original chromagram; Smoothed chromagram using: (b) beat-synchronous chromagram (c) moving average filter with L = 21; (d) median filter with L = 21; (created based on Cho 2014 Figure 5, PhD thesis) 30
Fig. 2.17 Chroma templates for Major and Minor chords with their musical staff notation. Red cells indicate notes in a chord with value of 1, otherwise blank cells indicate their absence with value of 0; The root notes in each chord/template are highlighted with bold border cells (created based on Müller 2015 Fig 5.6. and Fig. 5.14, Fundamentals of Music Processing - Springer) 31

Fig. 2.18 Example of a 3-state HMM in the key of D Major, decoding a sequence of 6 observations, where a1=Dmajor, a2=Gmajor, a3=Amajor, so our final chord recognition sequence is D G G G A A. **Black** arrows indicate state transitions (A), **red** arrows denote emissions (B). Initial state probabilities (C) are implicit in the model initialization and not depicted here (created based on Müller 2015 Fig. 5.28, Fundamentals of Music Processing – Springer) 34
Fig. 2.20 On the left, the architectural diagram of the CREMA model is presented. On the right, tables delineate both the model's parameter configurations and its training settings 37
Fig. 2.21 Diagram illustrating the Decibel's framework. 'M' denotes the matching between various representations of the same track. Data formats are indicated by the rectangular shapes, while processes are displayed as rounded rectangles. The components shaded in grey highlight how decibel extends existing ACE methods. (created based on Odekerken et al 2020 Figure 3.1, Technical Report) 39
Fig. 2.22 Model for chord representation by its three main components: the root pitchname, the list of the component intervals and the bass note interval with a chord example (Note: The 5th is not omitted in the Musical Staff Notation) 41
Fig. 2.23 Four possible combinations of chord syntax definition 41
Fig. 2.24 ACE Transcription file formats: lab format (left), and JAMS format (right) 43
Fig. 3.1 Application architecture diagram presenting client-server communication for audio content analysis and visualization 50
Fig. 3.2 The three main parts of PAT. Brown: 'PAT Controls'; Blue: 'Backing Track Audio Player'; Red: 'Multitrack Recordings' 55
Fig. 3.3 The three parts of the 'Audio Player' 56
Fig. 3.4 'Audio I/O Controls' sidebar in: a) closed state b) opened state c) description tooltip 56
Fig. 3.5 Audio I/O Controls capabilities: a) 'Import from Disk' via file Select, b) 'Import from Disk' via drag and drop, c) 'Import from Repository', and d) 'Export to Disk or Repository' 57
Fig. 3.6 Loading progress bar in the waveform container 57
Fig. 3.7 Audio Player stages: a) Initial, b) Audio Loaded c) Analysis Loaded 58
Fig. 3.8 Representation of the 23 chord regions comprised from the total amount of root notes plus special symbols (14 colors are unique due to enharmonic notes are sharing color) 59

Fig. 3.9 a) One object in the 67-Item variations array; b) Blue: Tooltip according to description field; Red: Repeated chords; Brown: 'No chord' case 60
Fig. 3.10 Context-aware toolbar across different application stages: a) Initial; b) Audio Loaded; c) Analysis Loaded 61
Fig. 3.11 Analysis confirmation; b) Analysis progress bar; c) Local storage analysis load ... 62
Fig. 3.12 a) 'Snap (beats)' activated; b) 'Click Track' activated; c) Respective tooltip 62
Fig. 3.13 Toolbar center featuring the Annotation list, Delete button, and Edit toggle 63
Fig. 3.14 a) The question icon tooltip explains how the beats and chords are visualized in the waveform. b) The available options of the Player Settings menu 63
Fig. 3.15 Comparing Backing Track Player States - a) Edit Mode and Editing Status both off (before any modification); b) Edit Mode on, Editing Status off; c) Edit Mode and Editing Status both on; d) Edit Mode off, Editing Status on; e) Edit Mode and Editing Status both off, (after successful modification). 64
Fig. 3.16 Popup menu showcasing 'Delete Beat' action 65
Fig. 3.17 Chord editor (a) without chord categories (b) with chord categories. The chord editor, accessible during chord editing, can display chord categories based on user selection in settings menu. Additionally, it offers tooltip descriptions for hovered symbols. The fonts for chord types, roots, and accidentals align with those used in marker labels..... 66
Fig. 3.18 Tooltips for Edit Mode: (a) editing tools and (b) Info icon 66
Fig. 3.19 Popup menus showcasing (a) 'Save' and (b) 'Cancel' actions 67
Fig. 4.1 Waveform with ground truth (dotted lines) and predicted (solid lines) beat annotations..... 72
Fig. 4.2 WCSR between MADMOM and CREMA chord recognition models 80
Fig. 4.3 Left: Histograms showing the distribution of WCSR scores for the best-performing alignment (CREMA 0.41) across various evaluation metrics for a dataset of 415 audio files. Right: Corresponding boxplots illustrating the median, interquartile range, and outliers, offering a statistical summary for each metric 81
Fig. 4.4 Bar chart comparing the Weighted Chord Symbol Recall (WCSR) between the CREMA model and the top three beat alignment derivatives (CREMA 0.41||0.43||0.42) 82
Fig. 4.5 Histogram of differences between the CREMA model and the top alignment..... 82
Fig. 4.6 Screenshot from the screen recording of experiment 1 83

Fig. 4.7 PAT application effectiveness and importance of content analysis and curation 84
Fig. 4.8 Screenshot from the screen recording of the experiment 2 84
Fig. 4.9 Screenshot from the screen recording of the experiment 3 85
Fig. 4.10 PAT application features assessment and future enhancements 85

List of Tables

TABLE 2.1 Indicative Music Information Retrieval Tasks Evaluated in the MIREX Contests 4
TABLE 2.2 Definition of shorthand notations for common chords, with their interval lists .. 42
TABLE 3.1 Breakdown of command-line arguments for chords and beats analysis script 51
TABLE 3.2 Summarizing key functions, methods, and outputs of the audio analysis script . 52
TABLE 4.1 Descriptive statistics for performance metrics in the quantitative assessment of beat tracking 73
TABLE 4.2 Chord qualities and their evaluation in chord vocabularies 77
TABLE 4.3 Segmentation value equals to the lowest between under- and over-segmentation 79
TABLE 5.1 Library dependencies for the scripts of music content analysis and evaluation .. 95
TABLE 5.2 Packages dependencies for the web application development and interface 96
TABLE 5.3 Chord types mapping used between the shorthands in JAMS format to: a) GJT OTF ‘Encoded’ format; or b) ‘Description’ format, which provides the full name of the chord and serves as an extended tooltip; or c) ‘Simplified’ format, serving as tooltip in Edit mode. 97

1 Introduction

1.1 Objective and Focus

The objective of this thesis is to develop a web application that visually depicts the rhythmic and harmonic content of a music piece and explore its effectiveness in enhancing user engagement in the context of collaborative, computer-assisted music lessons. The rhythmic and harmonic content of music can be automatically derived using intelligent and state of the art programming toolkits used in Music Information Retrieval (MIR) research. As the target application is aimed at music education, it additionally provides functionalities for curating the automatically retrieved content, with the purposes of increasing the accuracy of automatic content retrieval as well as for providing the means for designing and carrying out collaborative student assignments related to rhythm and harmony courses.

The motivation for developing this application originates from the research endeavors of the MusiCoLab project[1], which aims to provide an efficient platform for music learning and teaching. The web application developed in the context of this thesis represents one of the several applications of the MusiCoLab platform. The name of this application is Play-Along-Together (PAT). PAT draws inspiration from play-along, which refers to practicing music performance while listening to an accompaniment, commonly a pre-recorded performance referred to as backing track. Moreover, PAT incorporates real-time audio and video conferencing via Jitsi-Meet and is seamlessly integrated in the MusiCoLab platform. Additional components of the platform also used by PAT are an Online Data Repository and a Moodle-based Learning Management System (Alexandraki et al., 2023).

Building upon a set of versatile capabilities, the Play-Along-Together (PAT) application incorporates affordances of automatic content analysis of backing tracks, playing along with these tracks, overdubbing teacher/student performances, producing mix-downs, and generating new backing tracks from recorded performances. This thesis focuses on the automatic analysis of rhythmic and harmonic content in backing tracks as well on the implementation of further interactive utilities that enable collaborative, real-time editing and annotation to allow curating the automatically derived content. These additional features aim at enhancing the process of

[1] https://musicolab.hmu.gr/musicolab/project_en.php

music learning and teaching in an attempt to deliver more efficient, intuitive and engaging experiences. Details on the implementation of these functionalities are presented in chapter 3.

1.2 Thesis structure

The remaining of this thesis is structured as follows:

Chapter 2 provides an overview of Music Information Retrieval (MIR) research and introduces relevant programming libraries in this field. It explains the algorithmic tasks of Audio Beat Tracking and Audio Chord Estimation, including their methodologies, state of the art implementations, and annotation formats commonly used for dataset assemblage. The chapter concludes by summarizing the main challenges and limitations of these methodologies, as identified in previous research endeavors and discusses perspectives for future work.

Chapter 3 presents the architecture of the developed application and provides details on the development of the client and server-side components. It explains how these components interact and the purpose they serve in the application. The chapter concludes by consolidating the findings of this evaluation with respect to its capability to support the requirements of music teaching and learning.

Chapter 4 presents a computational evaluation of the chord and beat estimation algorithms as well as an assessment of the experience of users of the PAT application. The computational evaluation uses standard performance metrics of MIR benchmarks. The assessment of user experience is based on three pilot experiments, which were carried out as part of the research activities of the MusiCoLab platform. The chapter concludes by discussing the potential benefits of encompassing innovative developments in artificial intelligence in computer-assisted music education.

Finally, chapter 5 summarizes the achievements of this work, outlines its contributions, and discusses further improvements of the PAT application as well as suggested perspectives for future research and development at the intersection of intelligent music analysis and computer-assisted music education.

2 Related Work

2.1 Music Information Retrieval

In recent years, the increasing availability of digital music content has revealed the necessity for mechanisms that provide easier access to music tracks and the information describing their content. Furthermore, it appears that music metadata (e.g., title, artist, album, genre etc.) are insufficient for effectively accessing music content, because they are often unreliable and not detailed enough to the requirements of the users. To address this problem, computational algorithms designed to automatically retrieve music information from audio signals are being increasingly developed. Such algorithms focus on extracting *objective* characteristics (e.g., artist, album, length), *subjective* characteristics (e.g., musical genre/style, similarity to another piece), and *musicological* characteristics (e.g., key, rhythm, musical instruments). These algorithms permit the development of an abundance of innovative applications in the music industry, ranging from services offered to music consumers, such as music recommendation (e.g., Spotify) and song identification services (e.g., Shazam) to functionalities for musicians such as audio transcription (e.g., Melody Scanner) and chord recognition systems (e.g., Chordify).

The scientific field dealing with the automatic retrieval of information from music is known as Music Information Retrieval (MIR) and it is highly interdisciplinary. It leverages knowledge from domains including but not limited to acoustics, signal processing, machine learning, mathematics, musicology and music theory itself (see Fig. 2.1). The objective of MIR research is the development of algorithms and methodologies that enable a computer to 'listen' to music and perceive its content in a way which is comparable to an experienced musician.

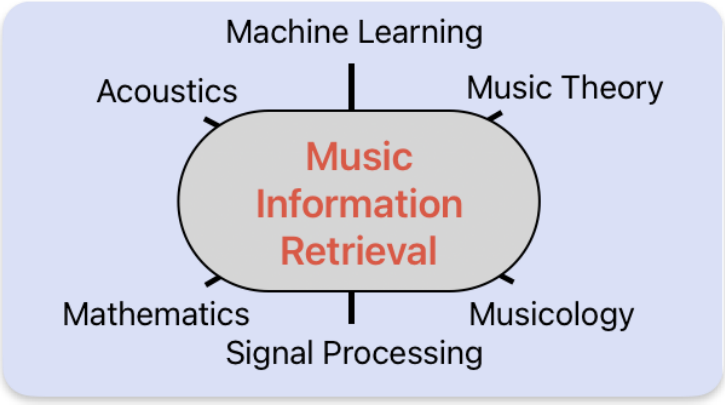


Fig. 2.1 Music Information Retrieval across different domains

To advance research in the field of MIR, the *International Society for Music Information Retrieval*[1] (ISMIR) conference has been established and is being held since 2000[2]. Additionally, a key initiative of ISMIR that provides a standardized framework for evaluating MIR algorithms is the Music Information Retrieval Evaluation eXchange[3] (MIREX) yearly contests. Those contests allow researchers to evaluate and compare their algorithms through a range of algorithmic problems - MIR tasks (see TABLE 2.1), defined by the ISMIR community, and make their work publishable at the annual ISMIR conference (Downie, 2008; Κούτρας et al., 2023).

TABLE 2.1 Indicative Music Information Retrieval Tasks Evaluated in the MIREX Contests

	Music Information Retrieval Task
AA	Audio Artist Identification
AB	Audio Beat Tracking
ADE	Audio Downbeat Estimation
ACE	Audio Chord Estimation
ACS	Audio Cover Song Identification
AFP	Audio Fingerprinting
AG	Audio Genre Classification
AK	Audio Key Detection
AME	Audio Melody Extraction
AMC	Audio Mood Classification
AMS	Audio Music Similarity and Retrieval
AO	Audio Onset Detection
AT	Audio Tempo Extraction
MFT	Multiple Fundamental Frequency Estimation & Tracking
ATC	Audio Tag Classification
ALS	Automatic Lyrics Transcription
SF	Score Following (a.k.a. Audio to Score Alignment)
DT	Drum Transcription
MD	Music Detection
QBSH	Query by Singing and Humming
QBT	Query by Tapping
SS	Structural Segmentation
SG	Symbolic Genre Classification
SMS	Symbolic Melodic Similarity

[1] https://ismir.net/

[2] Initially known as International Symposium for Music Information Retrieval, prior to 2002

[3] https://www.music-ir.org/mirex/wiki/MIREX_HOME

2.2 MIR Programming Libraries

In MIR, musical content information is retrieved by analyzing either audio (e.g., MP3s, WAV, etc.) or symbolic files (e.g., MIDI, score, etc.), or both. For most tasks a combination of Digital Signal Processing and Machine Learning approaches is used to provide an algorithmic solution. Commonly, the methodology for deriving a solution consists of the following steps:

1) Collect annotated files, for initial experimentation and optimization of the algorithm.
2) Extract audio features from the music signal (in cases of audio signals)
3) Implement, test, and optimize the algorithm in a programming language.
4) Search for datasets to be used for evaluation and/or training of ML models (if such models are facilitated)
5) Evaluate the performance of the algorithm using the dataset of step 4

Building upon the outlined methodology, the role of specialized programming libraries like *librosa*, *essentia*, *madmom*, and *essentia.js* becomes versatile. These libraries offer a variety of tools, ranging from extracting low level audio features to high-level ones (e.g., spectral centroid, spectral flux, melodic contour), retrieving mid- and high-level MIR task descriptors (e.g., beat tracking, chord estimation, music genre classification) implemented with or without statistical models (e.g., Hidden Markov Models, Deep Learning architectures), evaluation routines, basic audio input/output utilities and audio related visualization tools. Their comprehensive functionalities not only streamline the complex process of MIR research but also facilitate rapid experimentation and development, reducing the technical overhead for researchers, allowing them to concentrate on innovative aspects of MIR tasks (Κούτρας et al., 2023). The next few paragraphs provide a brief description of some of these libraries.

Librosa[1] is a Python library designed for audio and music signal processing, focusing on core MIR techniques such as feature extraction, including spectral, rhythm and structural analysis. It offers extensive documentation and tutorials, and the reader can quickly jump into the implemented source code of the desired functionality (McFee et al., 2015).

Madmom[2] is a newer Python library that additionally emphasizes the integration of machine learning in MIR tasks, providing algorithms for onset detection, beat, tempo, and downbeat

[1] https://librosa.org/doc/latest/index.html

[2] https://madmom.readthedocs.io/en/v0.16.1/

estimation, alongside machine learning techniques such as neural networks and Hidden Markov Models. Its goal is to incorporate the full processing chain within a single software framework so that standalone programs can be built without any dependency on other machine learning frameworks. A unique aspect of the library is the inclusion of high-performance pre-trained models for the tasks of chord recognition and downbeat estimation (Böck et al., 2016).

Essentia[1] is a C++ library that provides an extensive suite of over 260 algorithms for audio and music analysis and synthesis, with the addition of machine-learning-based descriptors like mood and genre classification. It is modular, optimized for high performance, and a large part of the provided algorithms are well-suited for real-time applications (Bogdanov et al., 2013). In addition to the C++ version, there are also versions of *essentia* (with bindings) for JavaScript and Python.

The JavaScript version of the library, called *essentia.js*[2], offers audio analysis capabilities to web browsers, by incorporating the latest advancements of the Web Audio API. *Essentia.js* is powered by the Web Assembly (WASM) backend and has been cross-compiled from the *essentia* C++ library, providing an easy-to-use JavaScript interface to a wide range of low-level and high-level audio features, including signal processing MIR algorithms and pre-trained TensorFlow.js machine learning models (Correya et al., 2021).

Collectively, these libraries provide excellent tools for advancing MIR research, but it is essential for researchers to possess both the knowledge and the programming skills necessary for advancing new developments. Such skills can be attained through experimentation with available libraries and comprehensive exploration of the underlying theories and practical applications of music information retrieval tasks. The tasks of Audio Onset Detection (AO), Audio Beat Tracking (AB), and Audio Chord Estimation (ACE) represent just a fraction of the diverse range of challenges in MIR and will be analyzed further in the following sections.

[1] https://essentia.upf.edu/index.html

[2] https://mtg.github.io/essentia.js/

2.3 Audio Beat Tracking

In music, the beat is the basic unit of time, which contributes to the understanding of rhythm and timing of a musical piece. It is often perceived as the pulse (regularly repeating event), to which a listener naturally taps along, and is a concept that is both intuitive and complex, leading to varying perceptions among listeners. Objectively, it is defined as a sequence of perceived pulses, usually equally spaced in time and is characterized by two key parameters: *phase* and *period*, where the beat positions are often aligned with the note **onsets** (beginning of a sound).

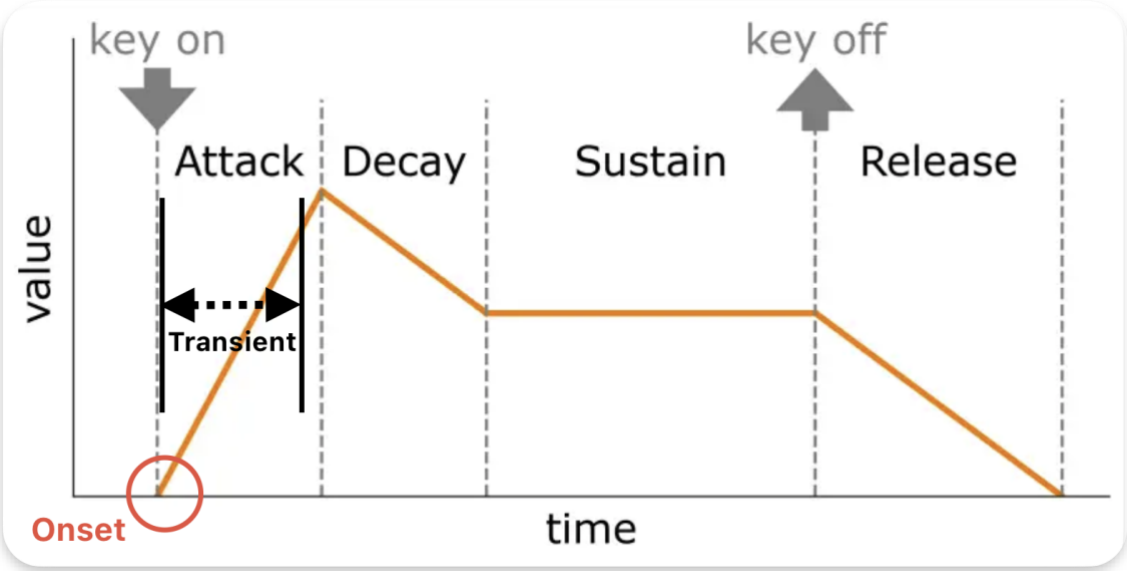


Fig. 2.2 ADSR (Attack Decay Sustain Release) envelope with onset and transient

In practice, the notion of onsets is related with other concepts such as the ADSR envelope of the sound, and the transient (see Fig. 2.2). The Attack, Decay, Sustain, and Release (ADSR) describe the dynamic progression of a musical sound, otherwise known as the amplitude envelope. However, not every musical sound features all four stages of this envelope. The Attack phase is where sound intensity grows to its peak. Then in the Decay phase, the sound's level decreases to what is known as the Sustain level, which persists until the Release, where the sound gradually ceases. Finally, the **transient**, characterized by a brief, high-energy burst, marks the onset of a sound, and contains important cues about the nature of the sound.

To describe the pulses within a musical structure, terms such as Tatum, Tactus/Beat, and Measure/Downbeat are pivotal (see Fig. 2.3). The Tatum, which corresponds to the sixteenth notes in the given example, represents the smallest rhythmic subdivision, and captures the fastest rhythmic event that occurs. The Tactus, refers always to the beat, or pulse that structures

the rhythm. In the example it aligns with the prominent waveform peaks and matches the quarter notes in the musical staff representation. Finally, the Measure, outlined between the vertical lines on the staff, describes a sequence of beats, with the Downbeat marking the first and typically the strongest beat, that signals the start of a new measure (Müller, 2015).

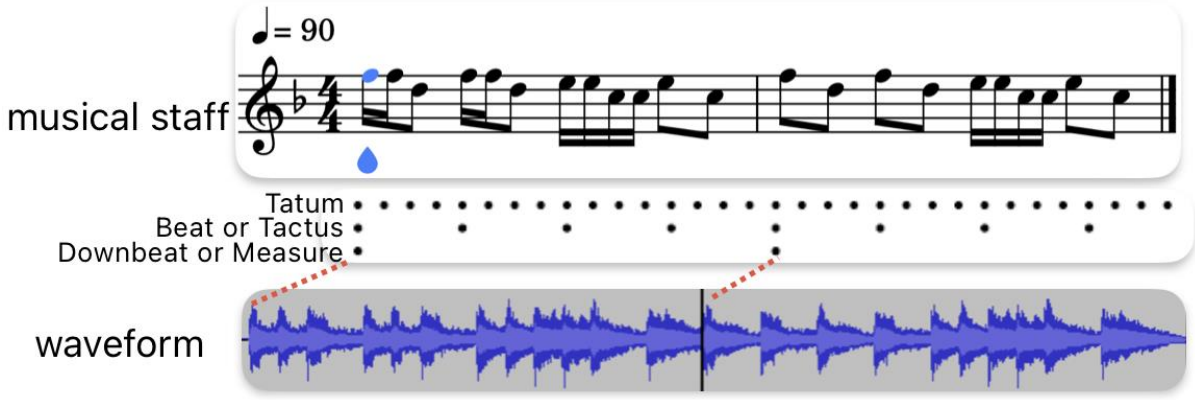


Fig. 2.3 Correlation of beat terminology with staff notation and waveform representation

Leveraging the above beat concepts, computational beat tracking approaches aim to automate the detection of beat locations, by integrating these characteristics. By capturing the rhythmic pulse and downbeat positions, tasks like cover song identification, audio structure analysis, instrumentation analysis, performance analysis (e.g., understanding of expressive performance attributes such as microtiming) and chord recognition become more accurate, and interactive music applications such as DJ mixing are created (Böck, Krebs, et al., 2016; Müller, 2015).

In order to achieve effective beat tracking systems, proper evaluation methods are required to estimate their accuracy. One such approach involves the subjective evaluation by human listeners, which has been proven robust even for untrained individuals, as they can differentiate between correct and incorrect beat estimations (e.g., beats tapped at an incorrect tempo). However, the inherent variability in human beat perception leads to inconsistencies as significant time and cost is required to conduct such evaluations across large datasets with multiple listeners. On the other hand, objective evaluations complicate the evaluation scheme by including instances where beats are evaluated at double or half the intended tempo, or on the off-beat (i.e., out-of-phase) by a small constant margin, leading to very low accuracy scores, even at 0%, which punish unfairly the system. Finally, legal constraints concerning the use of copyright protected recordings in combination with the effort required for generating accurate ground truth data, further complicate the evaluation process of beat tracking algorithms (Pinto et al., 2021).

2.3.1 Task Description

Computational rhythm analysis within the Music Information Retrieval (MIR) domain, addresses various aspects of musical rhythm, including onset detection, tempo estimation, downbeat estimation and beat tracking. Initially, tasks within Music Information Retrieval (MIR) relied on signal processing techniques and stochastic modeling. However, the advent of deep learning technologies has significantly transformed many MIR tasks. Specifically, in the case of beat tracking, deep learning techniques, involving recurrent and convolutional architectures, are now commonly employed.

Early computational methodologies relied on a sequence of processing steps beginning with the generation of a time–frequency representation, such as the short-time Fourier transform (STFT), from the audio signal. This was followed by the extraction of mid-level representations, which included onset detection functions to pinpoint note onsets, and the use of techniques like autocorrelation and dynamic programming to estimate the periodicity and phase of the given beats. These methods were then evaluated on small and often private datasets to assess the accuracy of the given approach. In contrast, modern deep learning approaches rely on large annotated datasets to train neural networks, where they learn beat activation functions that closely match target impulse trains. This is followed by inferring the final beat positions, with thresholding and/or peak-picking strategies or more advanced strategies like a dynamic Bayesian network approximated by a Hidden Markov Model (HMM), which takes into account uncertainties in beat prediction (e.g., spurious peaks or the absence of reliable information).

These types of methods achieve accuracy scores of the order of 90% on diverse annotated datasets. However, it is known that such methods are highly data-sensitive, meaning that these systems struggle with musical material that deviates from the training dataset, which generally yield much lower performance. A further concern relates to the fact that most annotated datasets focus on musical excerpts of up to one minute in duration, which raises questions about effectiveness of these systems in tracking beats across longer musical pieces.

To provide further insight to beat tracking, the following subsections (2.3.2 and 2.3.3) describe the methodologies and architectures of early computational approaches, as well as a state-of-the-art beat tracking system (Pinto et al., 2021).

2.3.2 Early Computational Methodologies

Typical approaches in beat tracking, start with the computational task of Audio Onset (or novelty) detection (AO), which aims to locate note onsets in the audio signal.

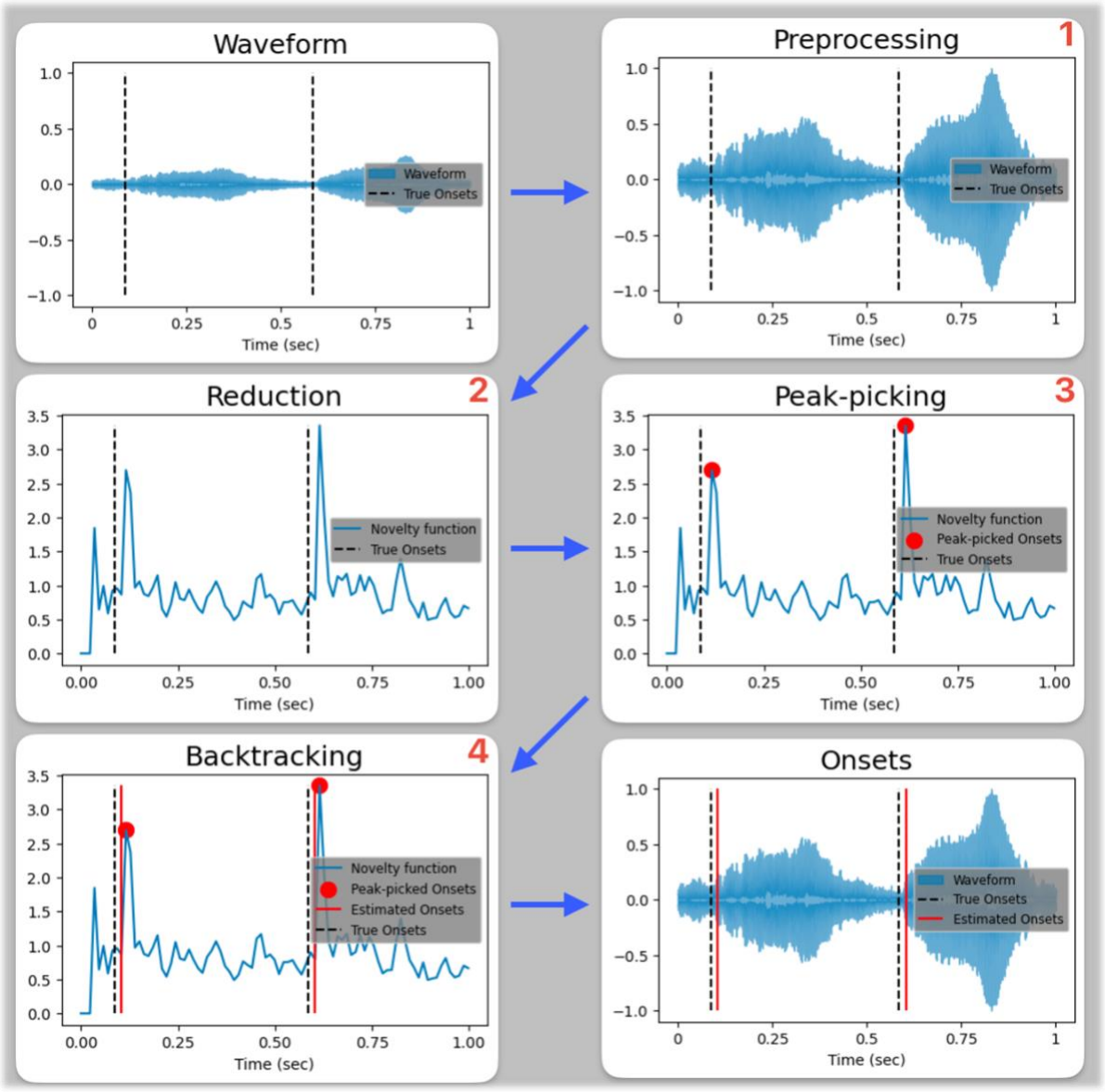


Fig. 2.4 The process of audio onset detection

The process involves converting the signal into a feature representation that highlights the desired properties, followed by applying a derivative operator[1] to this feature sequence to

[1]Four primary approaches exist for computing novelty functions: *energy-based, spectral-based, phase-based,* and *complex-domain* novelty. The reader can delve more with the practical examples available in https://www.audiolabs-erlangen.de/resources/MIR/FMP/C6/C6S1_OnsetDetection.html

derive a novelty function. Subsequently, a peak-picking algorithm is employed to identify onset candidates, which are then backtracked to the nearest preceding local minimum of the energy function used. This general procedure is akin to novelty detection used in music structure analysis, although the features and temporal levels relevant for structure analysis differ significantly from those in onset detection. In structure analysis, a tolerance window may range from 500 ms to a few seconds, whereas onset detection requires much higher accuracy, often far below 100 ms sometimes even on the order of 10 ms.

Most automated beat tracking approaches begin with the onset detection of an audio signal to identify rhythmic patterns, and are largely based on two assumptions: that beat positions coincide with note onsets, and that these beats are evenly spaced for a certain duration (tempo is roughly constant). While these assumptions hold true for a wide range of music, including most rock and pop songs, they may not apply to all genres, such as certain interpretations of romantic piano music. In the following paragraphs some of those approaches are going to be briefly analyzed, based on how they use the onset detection results. A common difference between those approaches is that some of them focus on the explicit extraction of note onset times, while the others utilize the novelty function directly to discern relative patterns within the music.

Approaches that rely on the explicit extraction of note onset times from the novelty function, typically employ peak-picking strategies based on adaptive thresholding. These methods then analyze the structure of these onsets to infer rhythmic elements. One such method calculates Inter Onset Intervals (IOIs) - the time gaps between successive note onsets, and analyzes their structure using histograms or probability distributions to hypothesize the beat period and tempo. Another method in this category is based on the computation of the autocorrelation function of the extracted onset times. By computing autocorrelation, this approach identifies repeating patterns in the timing of note onsets, focusing on periodicity to derive the beat period.

On the other hand, methods that avoid explicit extraction of note onsets, analyze directly the novelty function (see Fig. 2.4 Reduction plot). One such method applies autocorrelation to the novelty function, identifying periodic self-similarities and local periodicities within the music's structure. Another method compares the novelty function with a bank of comb filter resonators, where each resonator template represents a specific tempo, thereby detecting tempo and rhythm through resonance patterns. An alternative method utilizes a Short-Time Fourier Transform or a nonstationary Gabor transform to derive a frequency representation of the novelty function.

In this case, the novelty function is compared with sinusoidal templates, each corresponding to a specific frequency of the underlying pulse. All these methods estimate effectively local periodicities occurring in the underlying novelty function, and can be visualized by means of spectrogram-like representations referred to as **tempograms** (see Fig. 2.5**)**, **rhythmograms**, or **beat spectrograms** which also capture periodicities that change over time.

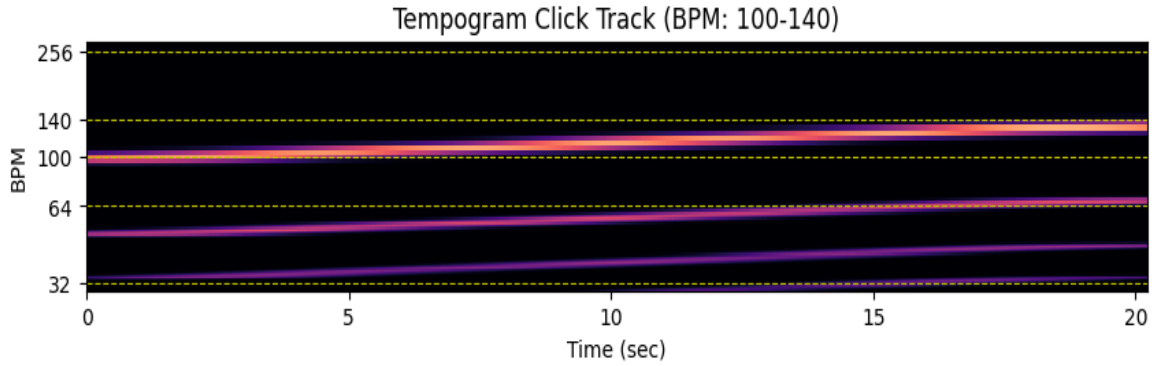


Fig. 2.5 Tempogram of click track with tempo increase from 100 to 140 BPM

Extending these methodologies, advanced techniques like the Predominant Local Pulse (PLP) approach can further account for tempo variations. The PLP approach offers a robust method for extracting local pulse information in complex music by enhancing the periodic nature of a novelty function. This process begins by determining a windowed sinusoid for each time position that best captures the local peak structure of the novelty function Then an overlap-add technique is employed to accumulate all sinusoids over time, creating as a result the PLP function, which effectively reveals the predominant local pulse, taking into account continuous tempo variations and local changes in pulse level. This method essentially focuses on finding the main beats that a human would typically tap his foot to and can adapt to various pulse levels (measure/downbeats, tactus/beat, tatum) by incorporating prior knowledge of the expected tempo range. Finally, while most research in music analysis has concentrated on identifying musical pulses at the tactus or foot-tapping level, a few approaches aim to simultaneously analyze the given piece at different pulse levels, offering a comprehensive understanding of the underlying rhythmic structures – like the previous discussed PLP approach (Müller, 2015).

2.3.3 State of the art approaches

This subsection presents a state-of-the-art beat recognition system that leverages data-driven deep learning techniques.

MADMOM BEAT RECOGNITION

The web application developed for the purposes of the present thesis uses the algorithm described in Böck et al. (2016), which has been implemented and integrated in the *madmom* Python library. The algorithm is data-driven and learns the relevant audio features directly from the spectrogram, thereby alleviating the cultural bias introduced due to hand-crafted audio features. The suggested model, comprises a Recurrent Neural Network (RNN), able to identify and distinguish between beats and downbeats.

A. Preprocessing

The audio signal is segmented into overlapping frames, weighted using a Hanning window before the transformation to a time-frequency representation via the Short-Time Fourier Transform (STFT). The system employs three different STFT lengths of 1024, 2048 and 4096 samples, limited to the frequency range of [30-17000] Hz, and then processes the spectrograms with logarithmically spaced filters. Finally, by adding the first order differences of the spectrograms to the input features the algorithm creates a neural network with an input dimension of 314.

B. Network Architecture and Training

The network architecture consists of three fully connected bidirectional recurrent layers with 25 LSTM (Long Short-term memory) units each, followed by a softmax classification layer to distinguish between beat, downbeat, and non-beat classes. The network is trained on various datasets with 8-fold cross-validation, using stochastic gradient descent and minimizing cross-entropy error. Initial weights are randomized with range [-0.1, 0.1], and training continues until no improvement is observed on the validation set.

C. Post Processing

The outputs of the neural network serve as observations for a Dynamic Bayesian Network (DBN), which jointly infers the meter, tempo, and phase of the final downbeat sequence. This DBN is parameterized to enhance downbeat tracking accuracy, using the validation set as a benchmark for performance and finds the globally best state sequence of the underlying beat positions (Böck, Krebs, et al., 2016).

2.3.4 Overview of Beat Tracking Applications

The development of beat tracking algorithms has led to the creation of applications that aid the work of musicians and DJs. For instance, beatmatching, which is an essential DJ technique for synchronizing two songs in terms of tempo (BPM) and beat alignment, has been made more accessible through software like DJ.Studio[1]. Similarly, other popular DJ software, such as SeratoDJ[2] and Traktor Pro[3], which aim at live mixing, employ real-time beat tracking algorithms to enhance performance and usability.

A different kind of application that leverages beat tracking algorithms is beaTunes[4], an advanced music tool for Windows and macOS designed for song analysis, playback, and the creation of engaging playlists. Beyond mere playback, beaTunes utilizes complex algorithms to analyze your music collection for metadata such as tempo (BPM), key, color, segments, similarities, loudness, and acoustical fingerprints.

For musicians, particularly drummers, tools like liveBPM[5] offer significant benefits. This particular beat detector uses the microphone to identify music and beats, and displays the tempo in real-time, which is invaluable during live performances and practice sessions, where a consistent tempo is required.

Furthermore, beat tracking serves as a fundamental step for other Music Information Retrieval (MIR) tasks, such as chord recognition. Applications designed for these tasks often align chords with beat positions, indicating the use of beat tracking algorithms. The next section will provide an overview of applications that demonstrate this integration.

[1] https://dj.studio/

[2] https://serato.com/dj

[3] https://www.native-instruments.com/en/products/traktor/dj-software/traktor-pro-3/

[4] https://www.beatunes.com/en/

[5] https://play.google.com/store/apps/details?id=com.DanielBach.liveBPM&hl=en&gl=US

2.4 Audio Chord Estimation

Harmony can be defined as the act of combining individual sounds to form cohesive units or compositions. In the Western music tradition, the fundamental elements of harmony are chords, which are typically composed of three or more notes that sound simultaneously (see Fig. 2.6). Based on its perception, this simultaneous combination of notes produces either consonance (a pleasant sound to the ear when played simultaneously) or dissonance (a harsh or unpleasant sound). The perception's intensity - of consonance or dissonance, is inherently linked with the related harmonic context of the musical piece which closely relates to the progression of chords over time. For that reason, the study of harmony delves deeply into chords, their creation, and the connecting concepts that direct their progressions. This understanding is crucial given that chord progressions play a pivotal role in the composition, description, and comprehension of genres like Western tonal music, jazz, and popular music. Consequently, features that capture this harmonic content are widely applied in music processing tasks such as music structure analysis and music information retrieval (Müller, 2015). Finally, chords describe the harmonic content of a piece at a mid-level, which lead the research community to use chord sequences in a variety of high-level tasks, including cover song identification, key detection, genre classification, lyric interpretation, and audio-to-lyrics alignment. These tasks involve identifying different versions of the same song, determining the musical key of a piece, tagging its musical style, interpreting its lyrics, and aligning the audio recording with its lyrics (McVicar et al., 2014).

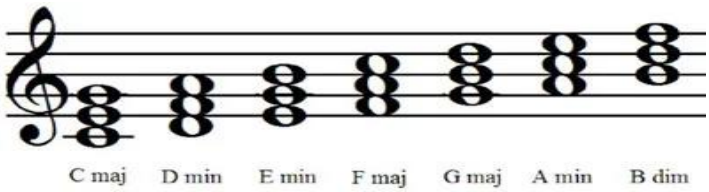


Fig. 2.6 Basic triads chords in the key of C major

Additionally, it is important to highlight that it is possible to perceive a chord through a melody or chord arpeggiation (see Fig. 2.7), even if non-chord notes are also present. Those non-chord notes are essential for the creation of a smooth melody line and the easier transition to another chord, however they also present a difficult challenge for automated systems that attempt to differentiate between harmonic and non-harmonic notes. Another difficulty arises from the definition of the term chord (a group of simultaneous sounding tones). While most researchers agree that a chord should contain at least three notes, some regard a combination of two notes

also as a chord. When considering computational approaches, a chord should contain notes from at least three different pitch classes (distinct pitch classes instead of distinct notes), considering that notes that are one or several octaves apart are often considered to belong to the same ‘sound quality’ (Korzeniowski, 2018; Müller, 2015; Odekerken et al., 2020).

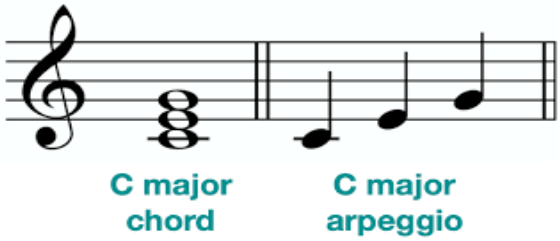


Fig. 2.7 C major chord presented in both closed position and arpeggiated

To evaluate the chord recognition task, human annotations are necessary, which are highly labor intensive to acquire. According to McVicar et al., (2014) an average duration of eight to eighteen minutes per song annotation is required. This is a highly expensive process that must be carried out by individuals with adequate musical knowledge and/or practice. In such cases, the identification of a chord can be subjective, and experts may disagree on its interpretation. This is evident by the existence of different ground-truth annotations for the same songs and ongoing discussions about appropriate evaluation metrics for chord identification (Korzeniowski, 2018).

For the task of recognizing chords in audio signals, different names have been occasionally used in the literature. The terms Audio/Automatic Chord Recognition (ACR), Audio/Automatic Chord Estimation (ACE), Audio/Automatic Chord Detection (ACD), and Audio/Automatic Chord Transcription (ACT) refer to the task of audio analysis, unless otherwise specified. The multiplicity of terms can create confusion for both researchers and practitioners, and it is important to differentiate this task from the closely related but fundamentally different task of Symbolic Chord Detection (SCD). It is important to note that the term ‘Symbolic Chord Detection’ (SCD) is often used interchangeably with ‘Automatic Chord Recognition from symbolic data’ or similar variations. Finally, one more observation is that the majority of research initiatives in this field concern chord identification of audio signals, rather than symbolic representations. This bias may be due to the fact that many researchers in this field have skills in signal processing, as observed by Downie (2008). In this thesis, the term Audio Chord Estimation (ACE) will be used to refer to the chord recognition task, which is also adopted by the MIREX community nowadays.

2.4.1 Task Description

Automatic Chord Recognition (ACR) as a research topic was initiated by Fujishima (1999). Prior to that, in 1985, Automatic Chord Estimation (ACE) methods were referred to as polyphonic note transcription (McVicar et al., 2014). While considerable progress has been made in the past two decades, the task is so complex that several challenges are still to be addressed. Nevertheless, ACE has evolved from the early knowledge-based systems to data-driven solutions, as the current progress in deep neural networks has remarkably improved the performance of the ACE task (Pauwels et al., 2019).

Fujishima introduced the use of chroma features, in which the frequency spectrum of a signal is mapped to the semitones of the chromatic scale via the 12 pitch classes. Although his system was designed for real-time applications and exhibited satisfactory performance in single-instrument sounds, it was not effective for poly-timbral signals representing the simultaneous performance of music instruments. To overcome this issue as well as the sequential nature of music, Sheh and Ellis introduced the use of Hidden Markov Models (HMM), a statistical model that is considered one of the most successful in Automatic Speech Recognition (ASR). These two works continue to be the foundational approaches upon which subsequent systems were developed, stemming from the early years of research on the task (Cho, 2014; Sheh & Ellis, 2003).

In general, they comprise two main steps: a *feature extraction* process and a *pattern matching* or *modeling* step (Cho, 2014; McVicar et al., 2014; Müller, 2015). First, in the feature extraction step, the audio signal is segmented into frames, and each frame is transformed into a suitable feature vector. Usually chroma-based audio features are commonly used as they are correlated with the underlying tonal information of the audio signal. Next, pattern matching techniques are applied to match each feature vector to a set of predefined chord models, with the best fit determining the chord label assigned to each frame. Additional techniques may be applied to improve the chord recognition results, such as *pre-filtering* before the pattern matching or *post-filtering* within or after the pattern matching step (Müller, 2015). In the case of post-filtering, it is often described in the literature as a separate main step called *chord sequence decoding* that incorporates the temporal aspect of music.

Initially, the techniques used in each stage were manually designed by an expert and used chroma extraction methods targeting specific nuisance factors, but they were unable to address all of them. However, in recent years, a shift has occurred with the emergence of deep learning

methodologies, following the publication of *'Rethinking automatic chord recognition with convolutional neural networks'* of Humphrey and Bello in 2012 (Korzeniowski & Widmer, 2016b). As a result, these steps have been modified to be compatible with deep learning approaches following naming convections of the architecture used in speech recognition systems. In the context of chord recognition, this involves *acoustic modeling*, which aims to predict a chord label for each frame of the audio, and *temporal* or *language modeling*, which aims to organize this information into meaningful segments of chords. A notable characteristic of these systems is that as they become more advanced, the relationship between tasks and elements becomes blurred, and the systems become more integrated (Pauwels et al., 2019; Korzeniowski, 2018).

In the following subsections, the main steps of the early chord recognition systems and two state-of-the-art approaches are analyzed in order to provide a more comprehensive understanding of the subject matter to the reader. For readers interested in an in-depth understanding, an excellent array of materials includes the works of Cho (2014), Mc Vicar et al. (2014), Müller (2015), and Korzeniowski (2018).

2.4.2 Early Computational Methodologies

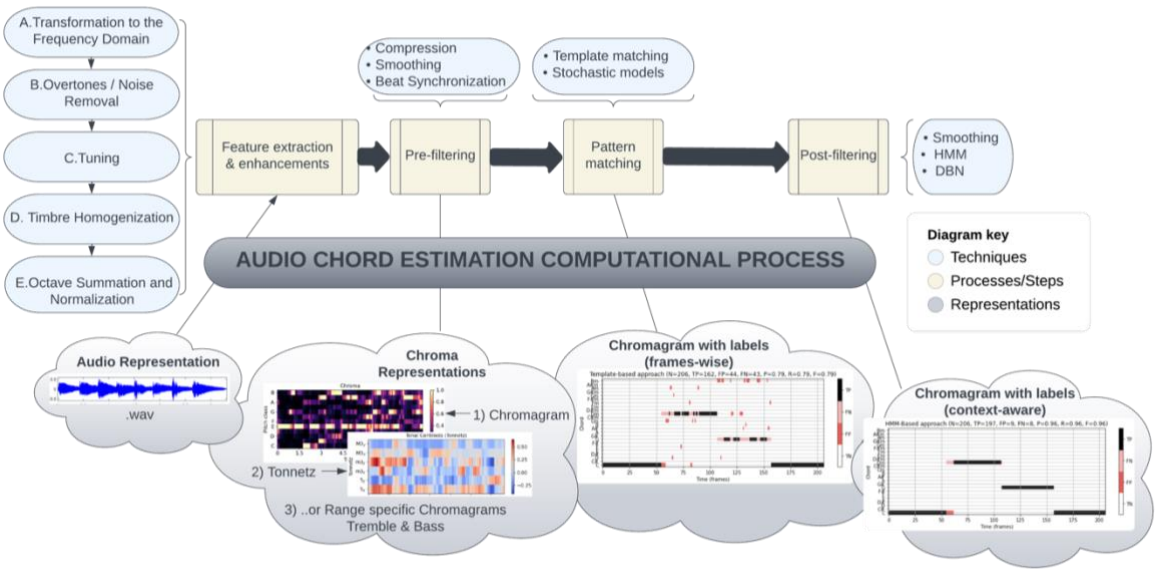


Fig. 2.8 Early computational methodologies in ACE

FEATURE EXTRACTION

The initial phase in various musical processing tasks is the process of feature extraction. This step aims to create appropriate features for the specific task being addressed. When dealing with Automatic Chord Estimation (ACE), this usually involves the creation of a chromagram (see Fig. 2.8). A chromagram is essentially a time-series of Pitch Class Profiles (PCPs), where each PCP includes the intensities of the twelve semitone pitch classes - C, C#, D, D#, E, E#, F, G, G#, A, A#, B – of the chromatic scale in Western music. These intensities indicate the strength of the presence of each note in the audio, providing a detailed view of the harmonic content at specific time intervals, which is useful for analyzing chords, melodies, and keys.

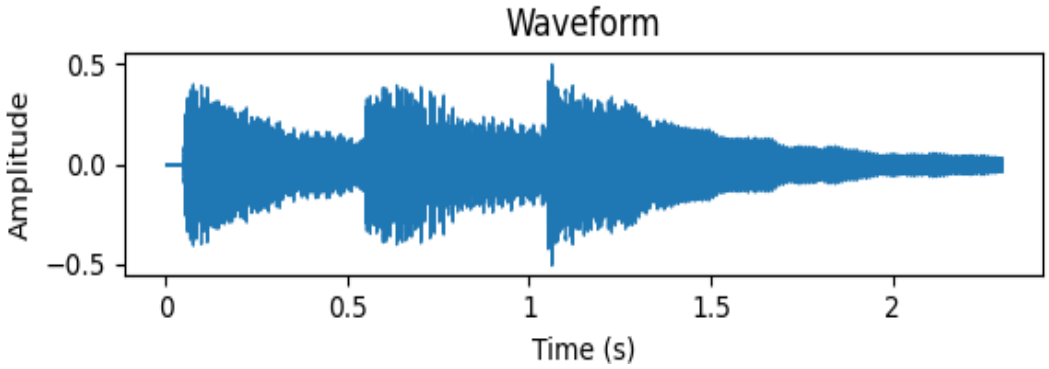
Prior to the advent of deep neural networks, the construction of a chromagram was carried out using processes incorporating a series of techniques such as: Transformation into the frequency domain, Overtones or Noise Removal, Tuning, Timbre Homogenization, and Octave Summation and Normalization. Those processes will be briefly analyzed in the subsequent subsections.

An alternative technique to achieve the same objective as the chromagram involves employing Tonal Centroid Vectors, that result in the Tonnetz representation (see Fig. 2.8). This transformation is founded on the principle that close harmonic relationships such as perfect fifths and major/minor thirds have large Euclidean distances in a chromagram, and a closer position could lead to increased performance (Mc Vicar et al., 2014).

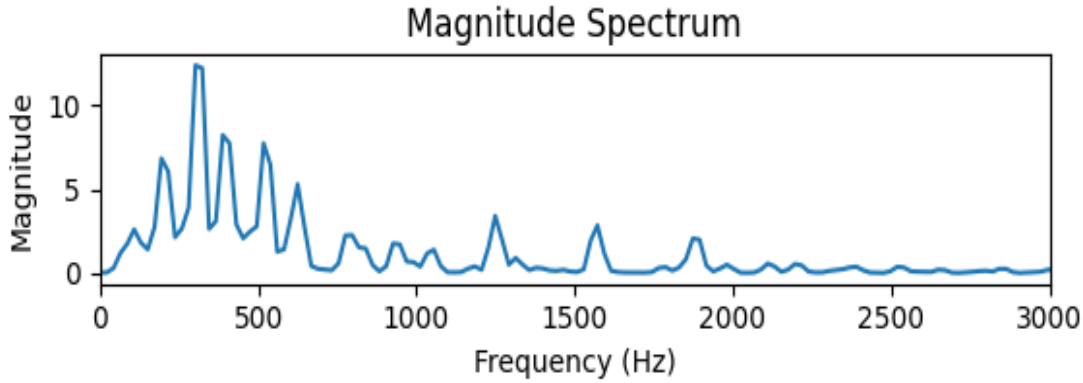
A. Transformation to the frequency domain

The extraction of chroma features in audio processing tasks, involves the conversion of the raw audio signal into a format that effectively represents its harmonic content. In general, digital music is usually sampled at up to 44100 samples per second, resulting in an extremely high-dimensional vector for each audio channel. However, this information does not provide a clear understanding of the audio's harmonic content, thus necessitating a transformation from the time to frequency domain, akin to the process of the human auditory system. In most cases, this transformation is calculated from the signal's magnitude spectrum, obtained through frequency analysis techniques like the Discrete Fourier Transform (DFT) or the Constant-Q Transform (CQT) (Cho, 2014). However, a different approach lies with the use of band-pass filters applied directly to a time-domain signal as proposed by Müller (2007, as cited in Cho, 2014; McVicar et al., 2014).

a)



b)



c)

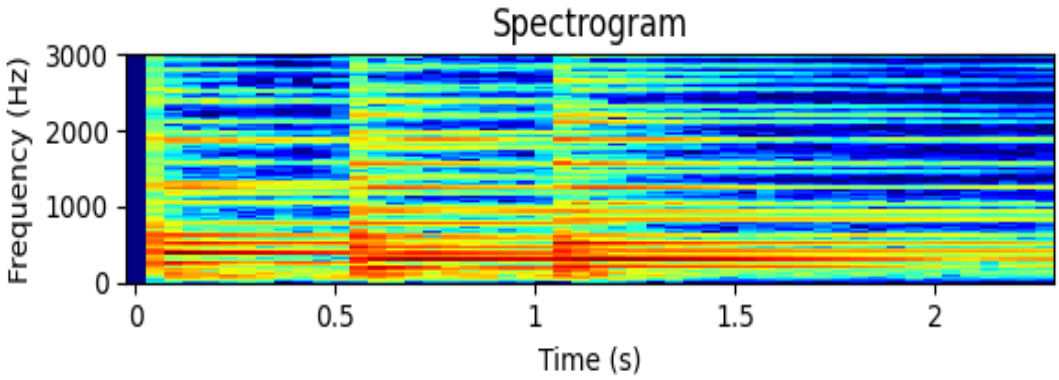


Fig. 2.9 Waveform, spectrum, and spectrogram visualizations of an audio file

The Discrete Fourier Transform (DFT) provides a global description of the frequencies present in the target audio, without any timing information (check the equation in (2.1)). However, in music processing tasks, capturing the local harmonic variations, such as chord progressions, is crucial. Therefore, it is necessary to use techniques that can preserve the timing information of the signal while still extracting frequency information. To overcome this limitation, the Short Time Fourier Transform (STFT) is commonly used in ACE (check the equation in (2.2)). To illustrate the differences between the previously mentioned techniques, consider Fig. 2.9, which shows three different plots: a *waveform*, a *spectrum*, and a *spectrogram*. The waveform

allows us to view the chronological events without revealing their content. The spectrum, which is the result of DFT shows us the spectral content but does not specify which events correspond to it. Finally, the spectrogram represents both the events and their spectral content.

- Discrete Fourier Transform (DFT) equation

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{j2pkn}{N}} \quad (2.1)$$

- Short Time Fourier Transform (STFT) equation

$$X(m,k) = \sum_{n=0}^{N-1} x(n+mH) w(n) e^{-\frac{j2pkn}{N}} \quad (2.2)$$

Where:

X(k) = The complex-valued result of the DFT for the *k*th frequency bin, containing both magnitude and phase information for the corresponding frequency

N = the length of the DFT window (in samples), $N \in \mathrm{N}$

n = DFT sample corresponding to a specific point in time, $n \in [0{:}N-1]$

x = real-valued discrete signal obtained by equidistant sampling with respect to a fixed sampling rate Fs given in Hertz

k = frequency index or frequency bin, $k \in [0{:}\frac{N}{2}]$

---

m = frame index of sliding window, $m \in Z$

H = hop size parameter (number of samples between adjacent DFT windows, $H \in N$]

w = real-valued discrete time window function (e.g., Hanning, Hamming, Blackman etc.)

The STFT computes the frequency magnitudes in a sliding window across the signal, generating a matrix (see Fig. 2.10), known as the spectrogram, where each column corresponds to the *squared magnitude* or *magnitude spectrum* of a particular time window (McVicar et al., 2014). This matrix leads to the commonly used visual representation (with the same name) spectrogram which in most cases illustrates the time on the horizontal axis and frequency on the vertical axis. Additionally, in the third dimension the magnitude of each frequency component is represented by color shades and is expressed in decibels (dB) due to the large dynamic range of values. The conversion to the decibel scale uses logarithmic functions and is

also called logarithmic compression. For a similar reason, the frequency axis is also often represented logarithmically (e.g., Mel spectrogram – Fig. 2.11b, Pitch based log spectrogram - Fig. 2.11c), which conform with the observation that the human perception of pitch is naturally logarithmic (Müller, 2015).

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ \vdots \\ X(N-2) \\ X(N-1) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & e^{-j\frac{2\pi}{N}} & e^{-j\frac{4\pi}{N}} & \cdots & e^{-j\frac{2(N-2)\pi}{N}} & e^{-j\frac{2(N-1)\pi}{N}} \\ 1 & e^{-j\frac{4\pi}{N}} & e^{-j\frac{8\pi}{N}} & \cdots & e^{-j\frac{4(N-2)\pi}{N}} & e^{-j\frac{4(N-2)\pi}{N}} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & e^{-j\frac{2(N-2)\pi}{N}} & e^{-j\frac{4(N-2)\pi}{N}} & \cdots & e^{-j\frac{2(N-2)^2\pi}{N}} & e^{-j\frac{2(N-2)(N-1)\pi}{N}} \\ 1 & e^{-j\frac{2(N-1)\pi}{N}} & e^{-j\frac{4(N-1)\pi}{N}} & \cdots & e^{-j\frac{2(N-1)(N-2)\pi}{N}} & e^{-j\frac{(N-1)^2\pi}{N}} \end{bmatrix}}_{\text{Fourier Transform Matrix}} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(N-2) \\ x(N-1) \end{bmatrix}$$

Fig. 2.10 Spectrogram matrix: a tabular representation of frequency and time

One of the limitations of the Short Time Fourier Transform (STFT) is that it uses a fixed-length window, which poses a trade-off between time and frequency resolution. Short windows result in poor frequency resolution, while long windows result in poor time resolution. For ACE applications, it is crucial to capture local harmonic variations with a semitone frequency resolution of at least half a semitone. In addition, the choice of window length and frequency range can significantly affect the time and frequency resolution, especially for low frequencies (McVicar et al., 2014). One possible solution to this problem is to use a frequency-dependent window length, which is implemented in the CQT technique. This technique was first introduced by Brown (1991) and has been applied to ACE by Nawab, Ayyash & Wotiz (2001) (McVicar et al., 2014). The frequency bins in the CQT are logarithmically spaced and have equal Q-factors (ratios of center frequencies to bandwidths). As a result, the frequency resolution is better for low frequencies and the time resolution is better for high frequencies. This technique is distinct from the case where a simple spectrogram frequency axis is converted to a logarithmic after the Fourier transformation (for a visual representation see Fig. 2.11d). In addition, CQT can be thought of as a wavelet transform but the term CQT is preferred because it emphasizes the transform's high Q-factors, which are equivalent to 12-96 bins per octave (Schörkhuber & Klapuri, 2010). Due to the nature of its design, the Constant-Q Transform exhibits varying time resolution across different frequency bins, leading to an unsynchronized sampling of these bins. This issue, along with other computational challenges, hindered the initial adoption of CQT. However, with the proposals made by Schörkhuber & Klapuri in 2010,

CQT has gained significant popularity and is currently considered one of the most widely used frequency representations for musical data. For further mathematical background in the CQT, the reader can consult the last-mentioned source.

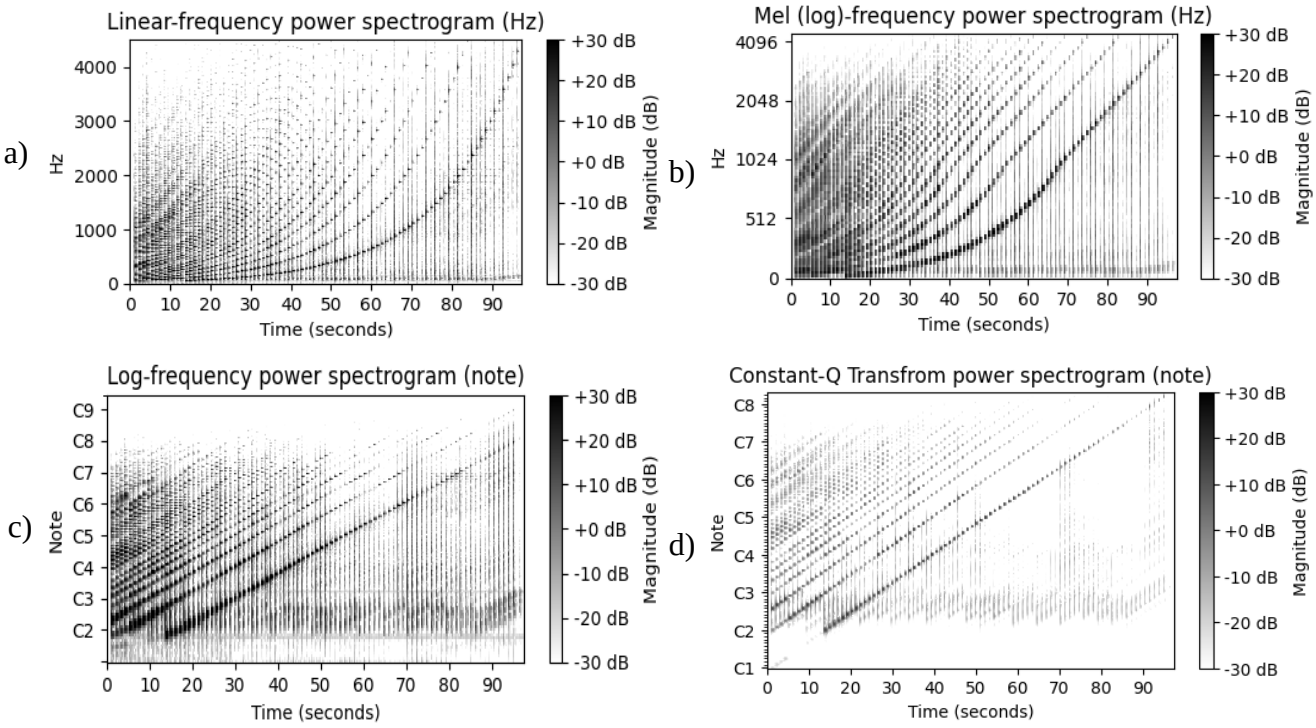


Fig. 2.11 Magnitude spectrograms of the chromatic scale on piano: a) linear frequency scale; b) logarithmic mel scale; c) pitch-based scale; d) Constant-Q transform scale;

Finally, the proposed method for utilizing band-pass filters, namely the Short-Time Mean-Square Power (STMSP) by Müller et al. (2004), employs a multi-rate filter bank. This approach differs from those of DFT and CQT, because a collection of band-pass filters is applied to a time-domain signal and the early segmentation is not required. Each individual band-pass filter allows frequencies within a semitone range to pass through, while attenuating frequencies outside that range and then segmenting it for further processing. An implementation is publicly available in the Chroma Toolbox[1] (Müller & Ewert, 2011), and in the *librosa* Python library.

[1] MATLAB implementations for extracting various types of pitch-based and chroma-based audio features are available in: https://www.audiolabs-erlangen.de/resources/MIR/chromatoolbox/

(The chromatic scale of the piano, as seen in Fig. 2.11, is also utilized to visualize the STMSP approach in Fig. 2.12)

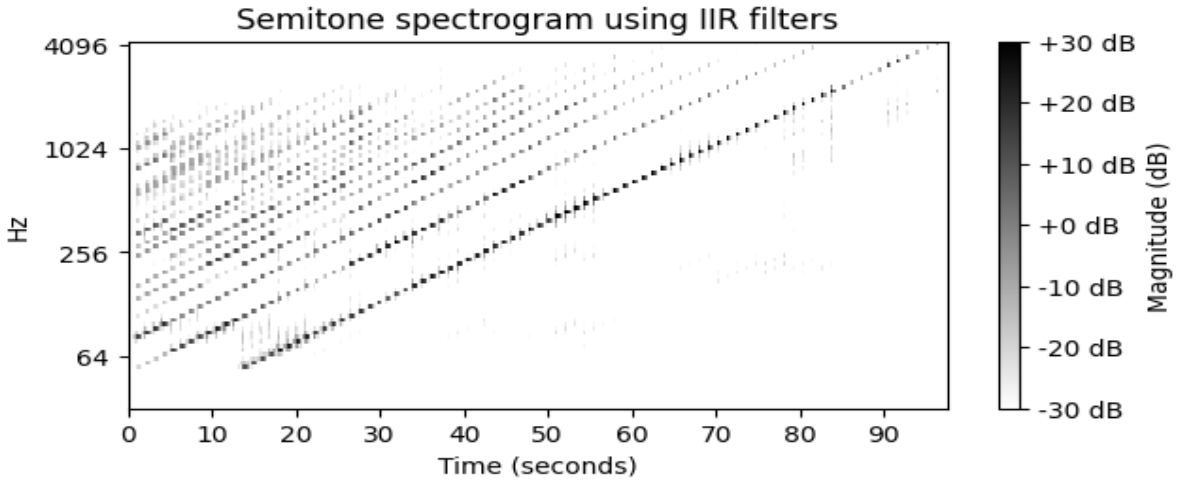


Fig. 2.12 Semitone spectrogram of the chromatic scale using IIR filters by calculating for each filter band the STMSP

B. Overtones / Noise Removal

In an ideal situation, a chroma vector should only contain the fundamental frequencies of the played notes in an audio signal. However, real-world music contains overtones produced by musical instruments, which inevitably affects the extracted chroma features. For example, a C major triad, which is made up of three distinct notes (C, E, and G) will produce energy at B due to the 2nd overtone of E, causing confusion with an E minor triad. Furthermore, percussive instruments like bass drums, due to their simple harmonic content and long duration, can blur the details in a chromagram, making harder the distinction of harmonies on instruments that have pitch (Cho, 2014). Therefore, it is advantageous to remove the signal that is unrelated to the harmonic content, which is the unhelpful part of the spectrum - also referred as background spectrum, and enhance as a result the clarity of the important features. However, removing the background spectrum may risk removing useful information for ACE, so it is necessary to ensure the content removed is not relevant to the task at hand (McVicar et al., 2014).

A conventional and simple technique to mitigate the issue is to eliminate or reduce high and low frequencies. This decreases the influence of overtones and unwanted noise, on the derived chroma features, because high-frequency bands primarily consist of overtones, while low frequencies may contain problematic bass drum sounds. The filtering process is often carried out by either limiting the range of the analyzed frequencies or applying a window function, such as a Gaussian window, to the pitch spectrum prior to the chroma folding process. For

example, J. Bello and Pickens (2005) computed chroma features by limiting the frequency range of 98 Hz to 5250 Hz, while Mauch and Dixon (2010) applied a Gaussian window centered at C4 (midi note = 60) (Cho, 2014).

When attempting to remove noise from percussive elements in music, a technique known as Harmonic Percussive Source Separation (HPSS) is commonly used, which was introduced by Ono et al. (2008) (McVicar et al., 2014) and has been shown to increase the accuracy of audio chord estimation (ACE) by Reed et al. (2009). This approach is based on the observation that percussive elements such as drums occupy a wide frequency range but are narrow in the time domain, while harmony elements such as melody, chords, and bassline behave in a reverse manner (McVicar et al., 2014). In essence, the spectrum is assumed to be a sum of percussive and harmonic components (see Fig. 2.13), which can be separated into two constituent spectra using median filtering across frequency bins (horizontal, or harmonic) and across time bins (vertical, or percussive). The resulting harmonic content can then be used for chordal analysis.

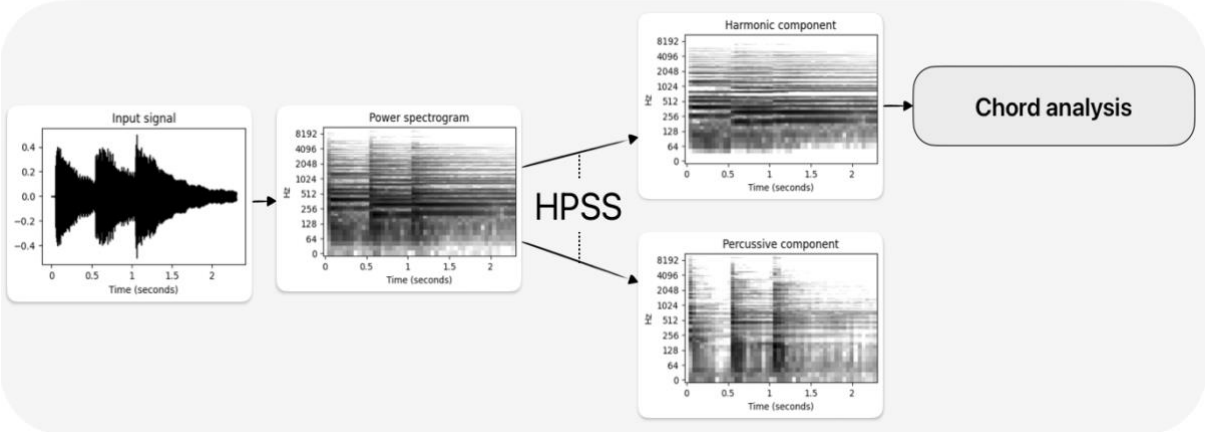


Fig. 2.13 Harmonic-Percussive Source Separation (HPSS) as a noise removal process in ACE

Other techniques within the same domain include the methodologies proposed by Gomez (2006), Varewyck et al. (2008), and Mauch and Dixon (2010). Gomez's approach places emphasis on the fundamental frequencies in a signal by incorporating the harmonic components of each frequency. Consequently, a Harmonic Pitch Class Profile (HPCP) is generated, instead of a Pitch Class Profile (PCP). Furthermore, Gomez's implementation also involved the restriction of the frequency analysis to the range between 100Hz and 5000Hz, for reasons as outlined before. Varewyck et al. (2008) proposed a method that uses multiple pitch tracking techniques to simultaneously remove background spectra and harmonics. Their resulting features matched chord profiles and outperformed unprocessed chromagrams. Finally, Mauch and Dixon proposed in 2010 an approach called the NNLS chroma which is

based on the concept of overtone removal using the Non-Negative Least Squares method. In their approach, the fundamental pitches in a signal are estimated from the log-frequency spectrum, which is similar to the CQT.

C. Tuning

In the work of Sheh and Ellis in 2003, it was determined that not all music tracks conform to the standard pitch A4 = 440 Hz, as standardized by the International Organization. To address this, they enhanced their chord recognition approach by doubling the frequency resolution in their spectrogram. Building upon this foundational work, Harte and Sandler (2005) ventured further by introducing a 36-bin chromagram. In their approach, each semitone was divided into three bins, making it possible to identify and adjust the tuning bias of each note by evaluating the energy peak across the note bins. By employing quadratic interpolation, they accurately inferred the peak positions, and using this information the global tuning bias was computed. The 36-bins were then circularly shifted to correct any bias and projected into a standard 12-bin chroma, ensuring a precise match with the intended pitch.

Gomez (2006) took a distinct approach. Instead of post-processing adjustments like Harte and Sandler, she preemptively estimated the reference frequency from the spectrogram. By recognizing the prominent peaks in the spectrogram as harmonics of notes with a shared detuning factor, she was able to measure the deviation from the standard pitch (A4 = 440 Hz). Finally, to ensure that the tuning was accurate and aligned with the given piece, she made adjustments based on amplitude-weighted histograms. This method was later expanded by Khadkevich and Omologo by integrating the phase information of the spectrogram to achieve a more accurate peak detection (Cho, 2014).

D. Timbre homogenization

Most real-world music is characterized by intricate combinations of sounds, stemming from various instruments that operate at different pitches and rhythms. The distinct qualities of the pitches that are produced from every instrument, can be furthermore described by the timbre of sound. Given this complexity, a basic method for managing the feature variance is the logarithmic compression of the dynamic range of the given features (McVicar et al., 2014).

Building on this concept, another technique shown to enhance ACE is adjusting pitch loudness to align with human perception. An example of this is the calculation of the loudness of the

spectrum using A-weighting, leading to loudness-based chromagrams, as proposed by Ni et al., (2012, as cited in McVicar et al., 2014).

Finally, a more direct approach that minimizes the timbre's influence is the method of Chroma Discrete Cosine Transform (DCT) Reduced log Pitch, that was developed by Müller and Ewert (2010) and is based on the understanding that a significant part of timbre is described by its spectral envelope (Cho, 2014).

E. Octave Summation and Normalization

The concluding step in chromagram construction involves aggregating pitch saliences within identical pitch classes, and then normalizing them (L^1 or L^2, or L^∞ norms) to ensure volume-independence. Applying this methodology across all frames produces a series of Pitch Class Profiles (PCP), that forms the final twelve-dimensional chromagram. Although this technique omits octave-specific data, which is often deemed irrelevant in ACE, it also limits the ability to differentiate between various inversions of the same chord.

To remedy this, some ACE systems employ two chromagrams: one for the treble range and another for the bass range (see Fig. 2.14) making it possible to identify chord inversions.

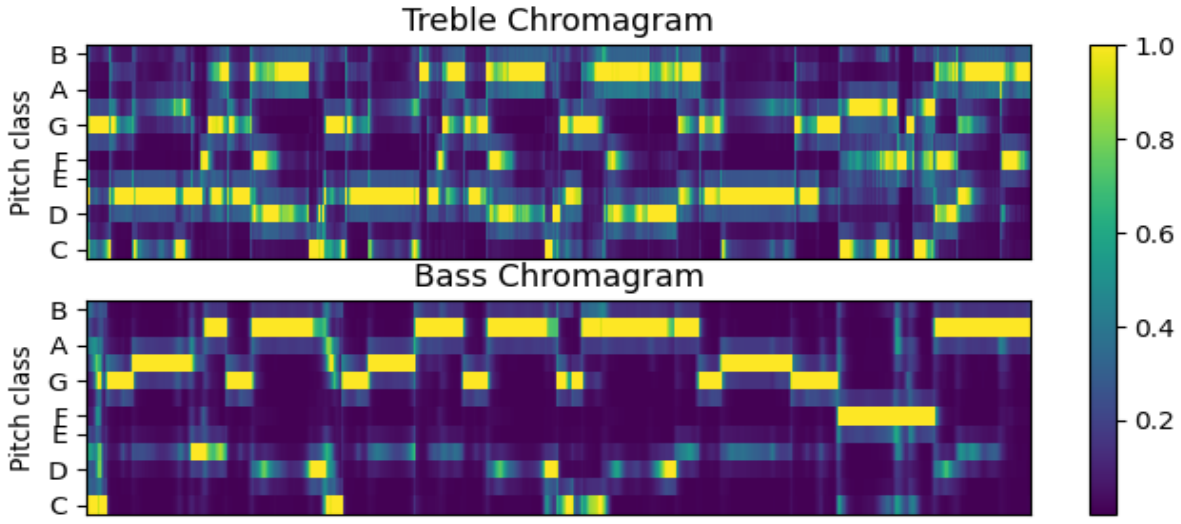
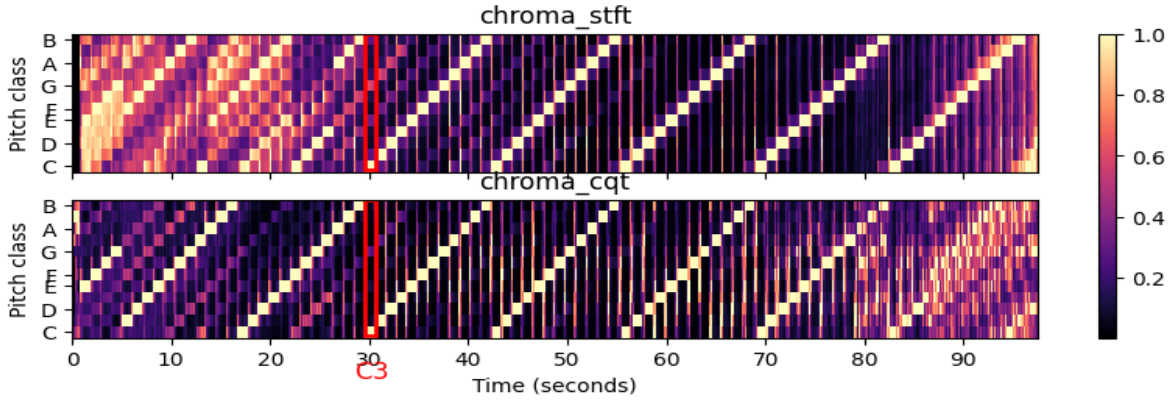


Fig. 2.14 Range specific chromagrams (top to bottom: tremble chroma and bass chroma)

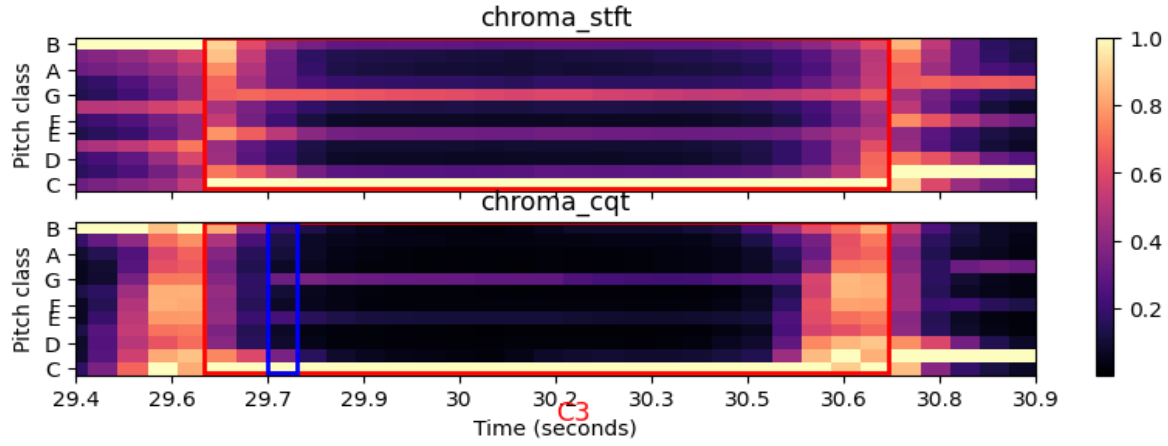
Moreover, it has been proven that estimating bass pitches from audio and incorporating bass probability into a hypothesis-search-based method enhances the accuracy of ACE models, which further validate the usage of such graphs/methods (Cho, 2014; McVicar et al., 2014).

Finally, a comparison between two chromagrams created using the STFT and CQT methods, can be seen in Fig. 2.15, which also illustrates the note C3 from the chromatic scale (red rectangle) within these chromagrams and one PCP vector of that given note.

a)



b)



c)

```
[1.         0.35300744 0.08753607 0.0771516  0.22667912 0.07672741
 0.101751   0.31977215 0.07768235 0.11242934 0.13201325 0.19689022]
```

d)

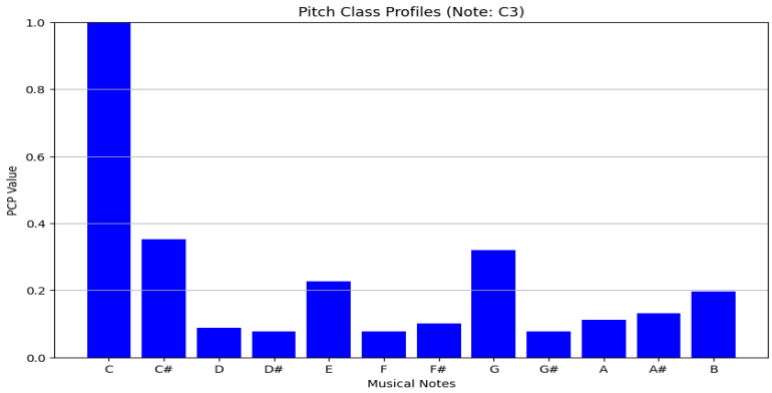


Fig. 2.15 STFT and CQT chromagrams of the chromatic scale on piano, as seen in Fig. 2.11 a) All the notes from the chromatic scale [A0 to C8], where the Red rectangle is the note C3; b) Zoomed-in chromagrams, where the Blue rectangle is 1 frame from the CQT chromagram of the note C3; The Blue rectangle frame: c) in PCP data structure vector array mathematical representation, d) graphically represented with a bar chart

PRE-FILTERING

The optimal step of pre-filtering (a.k.a. temporal smoothing) is applied before the pattern matching process to the chromagram and it involves smoothing of the features across adjacent frames. The smoothing is achieved using either a moving average filter (check the equation in (2.3)) or a moving median filter (Cho, 2014; Müller, 2015).

- Moving average filter

$$x_n^L(i) = \frac{1}{L}\sum_{l=0}^{L-1} x_{n+l-[\frac{L-1}{2}]}(i) \qquad (2.3)$$

Where:

$x_n^L(i)$ = output of the filter, which is a smoothed version of the original sequence X

L = the length of the moving averaging filter, $L \in N$ (the case of L = 1 yields the original sequence X)

i = chroma features component, $i \in [0:11]$

n = frame index containing physical timing position, $n \in Z$

$x_(n + l - [(L-1)/2])\ (i)$ = input frames within the window, centered around n

Illustrations of chromagrams filtered with or without pre-filtering techniques can be seen in the following Figure. In Fig. 2.16a a noisy chromagram with hidden harmonic signals can be seen. Fig. 2.16c and Fig. 2.16d display the chromagrams after smoothing, using moving average or median filters respectively. Alternatively, some systems employ pre-filtering through beat-synchronization, a method that averages feature frames between detected beat positions in a frame-based pitchgram, which can be seen in Fig. 2.16b. This approach assumes uniform harmonic content between consecutive beats, chords changing on beat positions, and a robust beat detection approach. By summarizing frames between beats, chroma features can be more resilient to noise, reflecting the long-term harmonic trends in the music signal. It is also important to emphasize that chord changes in music frequently align with beat positions, making the beat-synchronization method an optimal choice. However, as seen in section 2.3, finding beat positions through automated methods remains a complex task. When similar methods, such as smoothing, are used after or during the pattern matching step, they are typically categorized in the literature as post-filtering techniques. In the subsequent section, the case of the HMM-based recognizer will be discussed which integrates context-aware

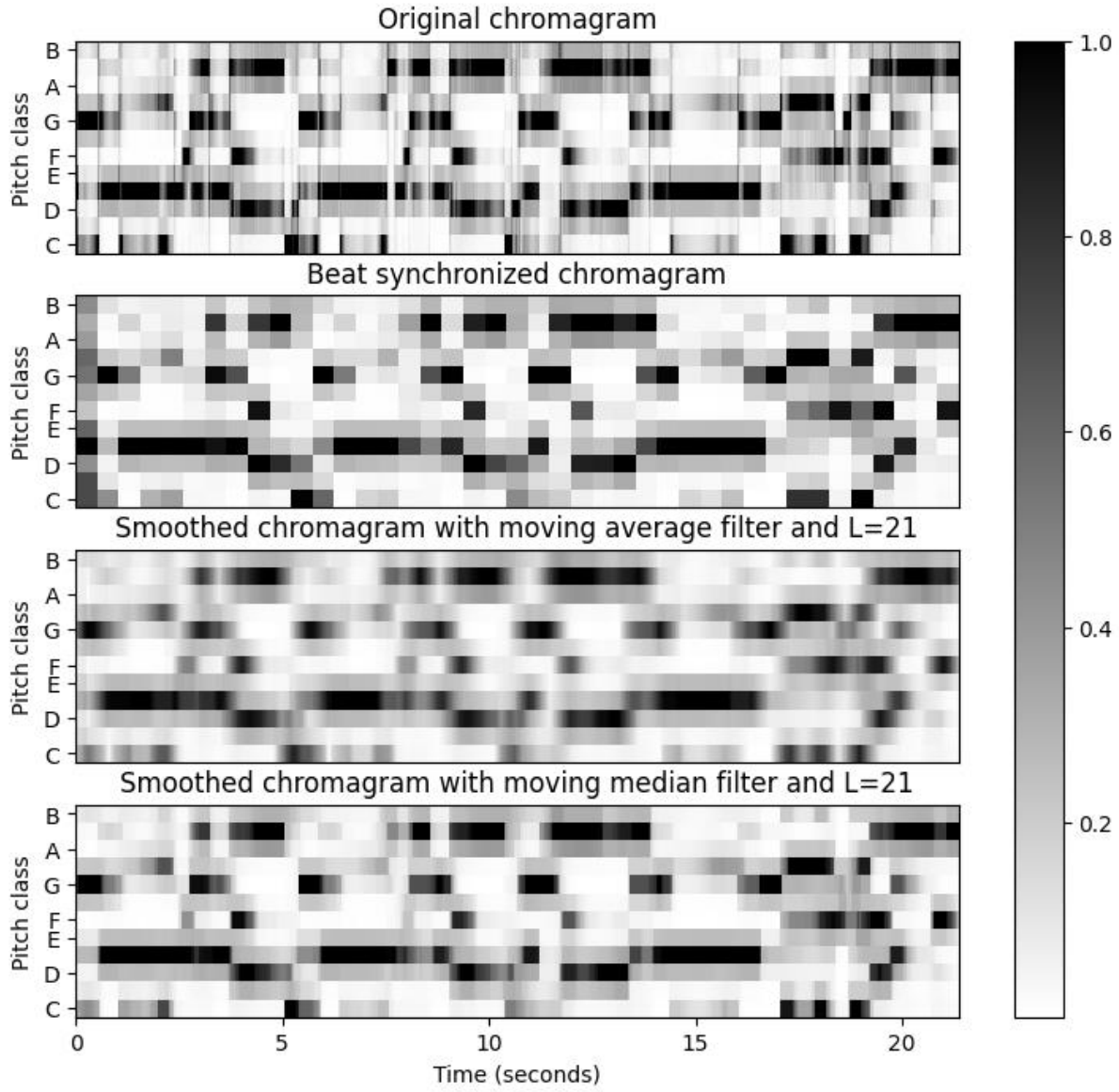


Fig. 2.16 Chromagrams filtered by various pre-filtering techniques of the same audio file: (a) Original chromagram; Smoothed chromagram using: (b) beat-synchronous chromagram (c) moving average filter with L = 21; (d) median filter with L = 21; (created based on Cho 2014 Figure 5, PhD thesis)

smoothing within the classification stage. In cases like these, it has been observed that pre-filtering smoothing techniques do not enhance the overall chord recognition result, and may even worsen it (Cho, 2014; Müller, 2015).

PATTERN MATCHING

Upon successfully extracting the necessary features for the ACE task, the next crucial step involves recognizing the chords corresponding to each frame. This is carried out during the

pattern matching stage, where each frame of the input chromagram is matched with a chord from a set of predefined chord templates. The two primary methods rely either on a deterministic chord template, originating from Fujishima's research, or the probabilistic chord model initially introduced by Sheh and Ellis.

A. Template-based matching

Template matching is a method used in chord recognition, where the feature vectors - usually 12-dimensional chroma vectors - extracted from the feature extraction step, are compared with deterministic chord templates. These templates contain binary information where 1 represents a note present in the chord and 0 the absence (Cho, 2014).

- C Major chord template

$$[1\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0] \qquad \text{(Do, Mi, Sol)} \qquad (2.4)$$

To describe a piece of music by the simplest form of chords - major and minor, a set consisting of the twelve major and minor triads is necessary. This results in:

- Set of minimum number of Major and Minor chord labels

$$\mathcal{A} = \{\text{C, C\#, D, D\#, E, F, F\#, G, G\#, A, A\#, B, Cm, C\#m, Dm, D\#m, Em, Fm, F\#m, Gm, G\#m, Am, A\#m, Bm}\} \qquad (2.5)$$

Given that set, the 24 distinct chord templates can be created (see Fig. 2.17, upper part):

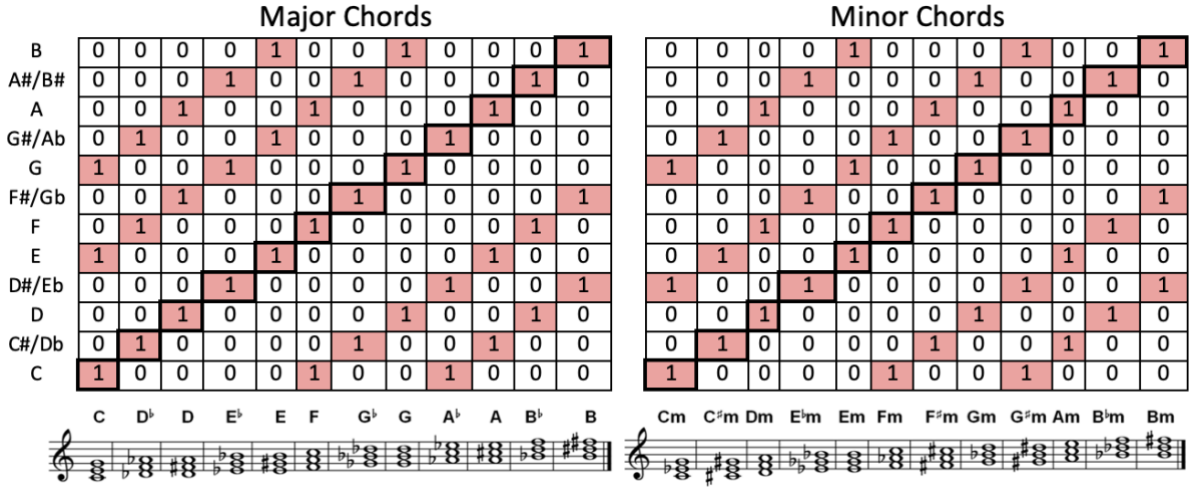


Fig. 2.17 Chroma templates for Major and Minor chords with their musical staff notation. Red cells indicate notes in a chord with value of 1, otherwise blank cells indicate their absence with value of 0; The root notes in each chord/template are highlighted with bold border cells (created based on Müller 2015 Fig 5.6. and Fig. 5.14, Fundamentals of Music Processing - Springer)

Subsequently, each frame of the chromagram is compared one by one to the 24 chord templates (T). This comparison is usually done with the Cosine Similarity metric (equation 2.6), although alternatives exist, such as Euclidean Distance and Correlation Coefficient. Then, the template-based procedure consists in assigning each frame (n) the chord label (λ_n), that maximizes (equation 2.7) the similarity between the corresponding template (t_λ) and the given feature vector (x_n). To do so it utilizes:

- Cosine similarity

$$s(x, y) = \frac{\langle x|y \rangle}{\|x\| \cdot \|y\|} \tag{2.6}$$

- and a Maximizing function

$$\lambda_n := \underset{\lambda \in \Lambda}{\operatorname{argmax}}\ s(t_\lambda, x_n) \tag{2.7}$$

This method is simple, convenient, and instructive, making it widely used in the early years of the chord recognition literature. Generally, it serves as the final step in many ACE systems, however, there are exceptions where additional post-filtering techniques are employed to manage temporal irregularities, akin to the pre-filtering phase (Müller, 2015).

For those interested in exploring further with hands on experience, an excellent array of resources are the Python Notebooks for Fundamentals of Music Processing (FMP)[1].

B. Stochastic models

While template matching provides a straightforward methodology for chord recognition, it is not able to capture intricate contextual information which is inherent in musical compositions. To address this limitation, stochastic modeling offers an approach that accounts for various aspects of musical context, by defining probability distributions for each chord class. In his 2014 thesis, Cho categorized various models into two main types: generative and discriminative. This classification scheme and additional hybrid models noted in other systematic reviews on the topic, are discussed in the following paragraphs.

On the generative side, three primary approaches are prominent for chord recognition: the Multivariate Gaussian Distribution (GM), the Gaussian Mixture Model (GMM), and the combination of a Gaussian Mixture Model with Hidden Markov Models (GMM+HMM). Each

[1] Python Notebooks for Fundamentals of Music Processing C5S2_ChordRec_Templates (audiolabs-erlangen.de)

offers unique advantages and complexities in terms of chord classification. The Multivariate Gaussian Distribution (GM) acts as a foundational model, using mean vectors and covariance matrices to compute the likelihood of a given chroma frame fitting into a specific chord category. Advancing from GM, the GMM introduces an additional layer of sophistication by incorporating multiple weighted Gaussian components, thereby capturing more complex feature spaces. However, this comes at the expense of increased computational demands. To account for the temporal sequence of chords, the GMM is frequently coupled with Hidden Markov Models, forming the GMM+HMM approach. This variant introduces temporal continuity constraints, providing a better understanding of chord progressions.

In the discriminative category, Conditional Random Fields (CRF) and Support Vector Machines (SVM) are some of the common choices where the focus of such models is on determining the classification boundaries between chords. The primary difference between these two models lies in their parameter tuning, where the CRF leverages the maximum conditional likelihood while SVM adopts a maximum margin criterion.

In addition, it is essential to underscore the prominence of Hidden Markov Models (HMMs) in the field of Automatic Chord Estimation (ACE). Their widespread usage likely stems from their successful applications in Audio Speech Recognition (ASR) and their ability to effectively model time-varying spectral vector sequences.

HMMs are characterized by three critical parameters: the initial state distribution (C), the state transition probabilities (A), and the state-to-observation probability distributions (B).

- The **initial state distribution** (C) lays the probabilistic groundwork for initiating the chord sequence. This can be informed by prior empirical data or domain-specific knowledge regarding chord distributions within a particular musical genre.
- The **state transition probabilities** (A) are guided by basic musical theory, specifying the likelihood of transitioning from one chord to another. This often reflects common chord progressions within the genre of music being analyzed (Dominant → Tonic).
- The **state-to-observation probability distributions** or **state-dependent emission probabilities** (B) serve as the link between the extracted musical features and potential underlying chords. These distributions essentially transform the observed musical data (which is a sequence of chroma vectors) into probabilistic chord predictions.

For the decoding step in Hidden Markov Models (HMMs), the Viterbi algorithm is often used. This algorithm applies dynamic programming to calculate the most probable sequence of states. It does this by breaking the entire sequence of observations into smaller, more manageable parts and finding the best solution for each part. As a result, the Viterbi algorithm effectively predicts the most likely sequence of chords. Additionally, it integrates by design a Post-Filtering step into the Pattern Matching stage, which helps to smooth out the analysis over each time frame (Cho, 2014; Müller, 2015).

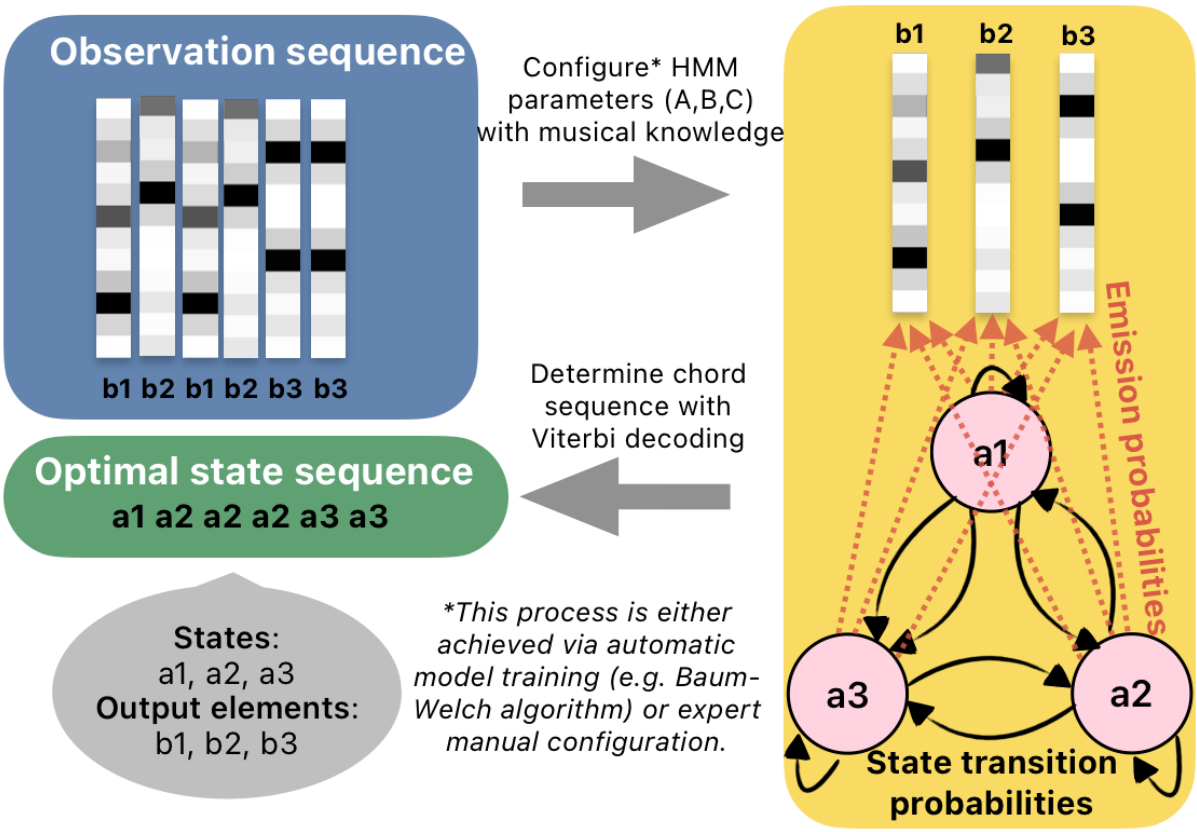


Fig. 2.18 Example of a 3-state HMM in the key of D Major, decoding a sequence of 6 observations, where a1=Dmajor, a2=Gmajor, a3=Amajor, so our final chord recognition sequence is D G G G A A. **Black** arrows indicate state transitions (A), **red** arrows denote emissions (B). Initial state probabilities (C) are implicit in the model initialization and not depicted here (created based on Müller 2015 Fig. 5.28, Fundamentals of Music Processing – Springer)

Further research in stochastic models has demonstrated the limitations of utilizing a single model across multiple musical genres. In contrast, it has been confirmed that incorporating genre-specific information enhances model performance by nearly 10% and improves probabilistic genre identification, a method which could be easily integrated into newer chord recognition systems (McVicar et al., 2014).

2.4.3 State of the art approaches

In recent years, chord recognitions systems utilize data-driven approaches by exploring new deep learning techniques, extending the stochastic models mentioned in the previous section. These models improve the accuracy of chord detection by accounting for complex inherent musical relationships, where it is very difficult to hand craft them in conventional computation approaches. Nonetheless, the discussion of early computational approaches remains relevant because a more detailed examination reveals that the internal mechanisms of data-driven approaches exhibit similarities in their functionality.

According to the Automated Chord Estimation (ACE) results published on the MIREX website[1], one of the top performing models for simple chord vocabularies (root, majmin) across various datasets was the one introduced by Filip Korzeniowski, Sebastian Böck, and Florian Krebs[2]. This model employs a Fully Convolutional Deep Auditory Model for predicting chord features and uses a Conditional Random Field model for the decoding phase[3].

In more complex vocabularies, the models developed in 2018 by Junyan Jiang, Ke Chen, Wei Li, and Guangyu Xia can be considered state of the art [4]. These models are based on a Recurrent Convolutional Neural Network (RCNN) and incorporate data augmentation strategies to address the rarity of certain chords in large vocabularies and to predict chord inversions. Although no open-source implementation of these models is available, one model with a similar architecture that also targets large vocabularies and employs comparable techniques is from McFee & Bello in 2017[5], which however has not been evaluated in the MIREX.

In the following subsections, two state-of-the-art approaches will be presented in more depth: one is the chord recognition algorithm used by this thesis, CREMA (McFee & Bello, 2017), and the other is DECIBEL, which improves ACE accuracy by making use of symbolic representations (Odekerken et al., 2020).

---

[1] Refer to www.music-ir.org/mirex for the latest MIREX challenge results (ACE task was last updated in 2020)

[2] Model first submitted in 2017 (KBK2) and resubmitted in 2018 (FK2)

[3] These models are publicly available through the MADMOM library; used in this thesis's server-side script

[4] Between the two submissions the latter model synchronizes chords with beats using the DBNDownBeatTracker processor from the *madmom* library, a methodology akin to the one implemented in this thesis

[5] Available through CREMA; primary model in this thesis's server-side script

CREMA AUDIO CHORD ESTIMATION

The chord recognition model integrated into the web application is provided from the open-source Python *crema* (Convolutional and Recurrent Estimators for Music Analysis) package and it is based[1]on the research conducted by McFee & Bello (2017). This implementation is one of the several attempts in the literature that utilizes a comprehensive vocabulary of chords and it capitalizes on inherent structural relationships between different chord classes.

A. Vocabulary of Chords

The model recognizes a rich vocabulary of 14 chord templates: min, maj, dim, aug, min6, maj6, min7, minmaj7, maj7, 7, dim7, hdim7, sus2, and sus4. This stands in contrast to models that merely distinguish between major and minor chords. Additionally, when a pitch class set does not align with these predefined templates, it is classified under special symbols: 'X' for an unknown chord, and 'N' for a no-chord scenario[2].

B. Architecture Overview

The model is designed with an encoder-decoder architecture, although not in the traditional mirrored sense (decoder mirrors encoder) commonly found in canonical examples. For a detailed breakdown of the model layers, please refer to Fig. 2.19.

❖ Encoder

The input audio is preprocessed into a harmonic constant-Q spectrogram (HCQT) with a hop size of 4096 at a sampling rate of 44,100 Hz which leads to a time resolution of 92.9ms between frames. For frequency resolution, the HCQT spans over 6 octaves starting at C1, comprising 36 bins per octave that captures the signal's fundamental tone along with its first harmonic.

The architecture comprises two Conv2D (two-dimensional convolutional) layers, the first of which amplifies harmonic saliency while the second behaves like a Conv1D layer and functions as a local feature extractor. Subsequently, a squeeze layer (squeezing out the

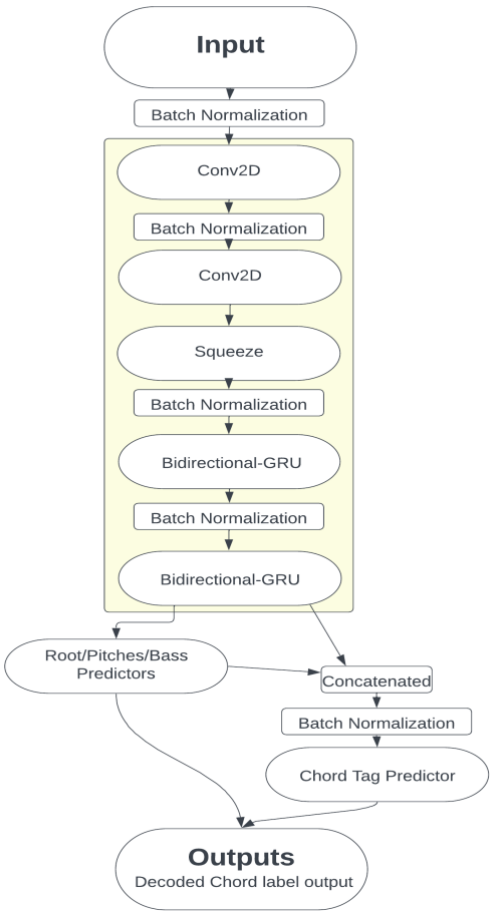
[1] Refer to https://github.com/bmcfee/crema/tree/main/training/chords for the implementation

[2] As a result, the vocabulary contains 170 classes; 12pitch_classes * 14chord _templates + 2special_symbols

frequency dimension) and two bi-directional gated recurrent units (Bi-GRUs) are employed to encode temporal features.

❖ Decoder

The decoder part of the model, features binary encoding predictors for the root, pitch classes, and bass note, in addition to a standard chord tag predictor derived from the concatenation of these layers and the hidden state from the last Bi-GRU.



Model parameters	
HCQT	Sample Rate: 44100, Hop Length: 4096, No. of Octaves: 6, Harmonics: [1, 2]
Conv2D (Layer 1)	Filters: 1, Kernel Size: (5, 5), Activation: ReLU, Padding: Same
Conv2D (Layer 2)	Filters: 72, Kernel Size: (1, 216), Activation: ReLU, Padding: Valid
Bi-GRU (Layer 1)	Base Layer: GRU, Units: 128, Return Sequences: True
Bi-GRU (Layer 2)	Base Layer: GRU, Units: 128, Return Sequences: True

Training parameters	
Data Augmentation	+- 6 semitones for each training track
Patch Duration	8-second patches
Learning Rate Reduction	At epoch 10
Dataset	1217 Tracks from Isophonics, Billboard, RWC Pop, and MARL Collections
Early Stopping	At epoch 100

Fig. 2.19 On the left, the architectural diagram of the CREMA model is presented. On the right, tables delineate both the model's parameter configurations and its training settings

C. Training and Post-Processing

During training, the model minimizes the sum of the losses across all outputs - four in total, utilizing softmax activation for the root, bass, and concatenated chord tag (merged layer), while employing multi-label logistic (sigmoid) activation for the pitch classes. Batch normalization layers are included to accelerate and stabilize the training process.

The model leverages structural similarities between chord classes to generate the time-varying sequence of chord labels - output of ACE (e.g., C:maj etc.) from the binary encodings of the chord roots, basses, and pitch classes. This output undergoes post-processing with Viterbi decoding, while chord inversion tracking is achieved through the chord/root output of the structured model.

DECIBEL AUDIO CHORD ESTIMATION

DECIBEL (DEtection of Chords Improved By Exploiting Linking symbolic formats) is a novel system[1], that exploits symbolic representations into the ACE task. It aligns midis and tabs to their respective audio files and extracts chord sequences from each representation, and then fuses the resultant chord sequences to increase accuracy[2].

A. Vocabulary

In terms of its vocabulary, DECIBEL recognizes 24 chords, consisting of all 12 major chords and all 12 minor chords, and the no-chord. The developers of this framework plan to investigate the estimation of more complex chords (e.g., sevenths) through symbolic representations in the future.

B. Framework

The DECIBEL framework is presented in the diagram of Fig. 2.20. It uses audio, MIDI, and tablature representations into a unified chord sequence. DECIBEL does not provide an ACE algorithm itself. Instead, it allows using existing algorithms within the framework and combine them with symbolic representations to achieve improved estimations of chord sequences. A Dynamic Time Warping method is used to align MIDI files with corresponding audio recordings. Subsequently, chord estimation is performed via a template matching algorithm. Tabs are parsed to produce untimed chord sequences, and then aligned with audio using a

[1] Refer to https://github.com/DaphneO/DECIBEL for the Python implementation

[2] Data fusion improved 2017/2018 MIREX ACE entries and the commercial application Chordify by 3.05% in the Weighted Chord Symbol Recall evaluation metric

Hidden Markov Model-based algorithm. Finally, DECIBEL employs a data fusion method to combine all estimated chord sequences into the final output sequence.

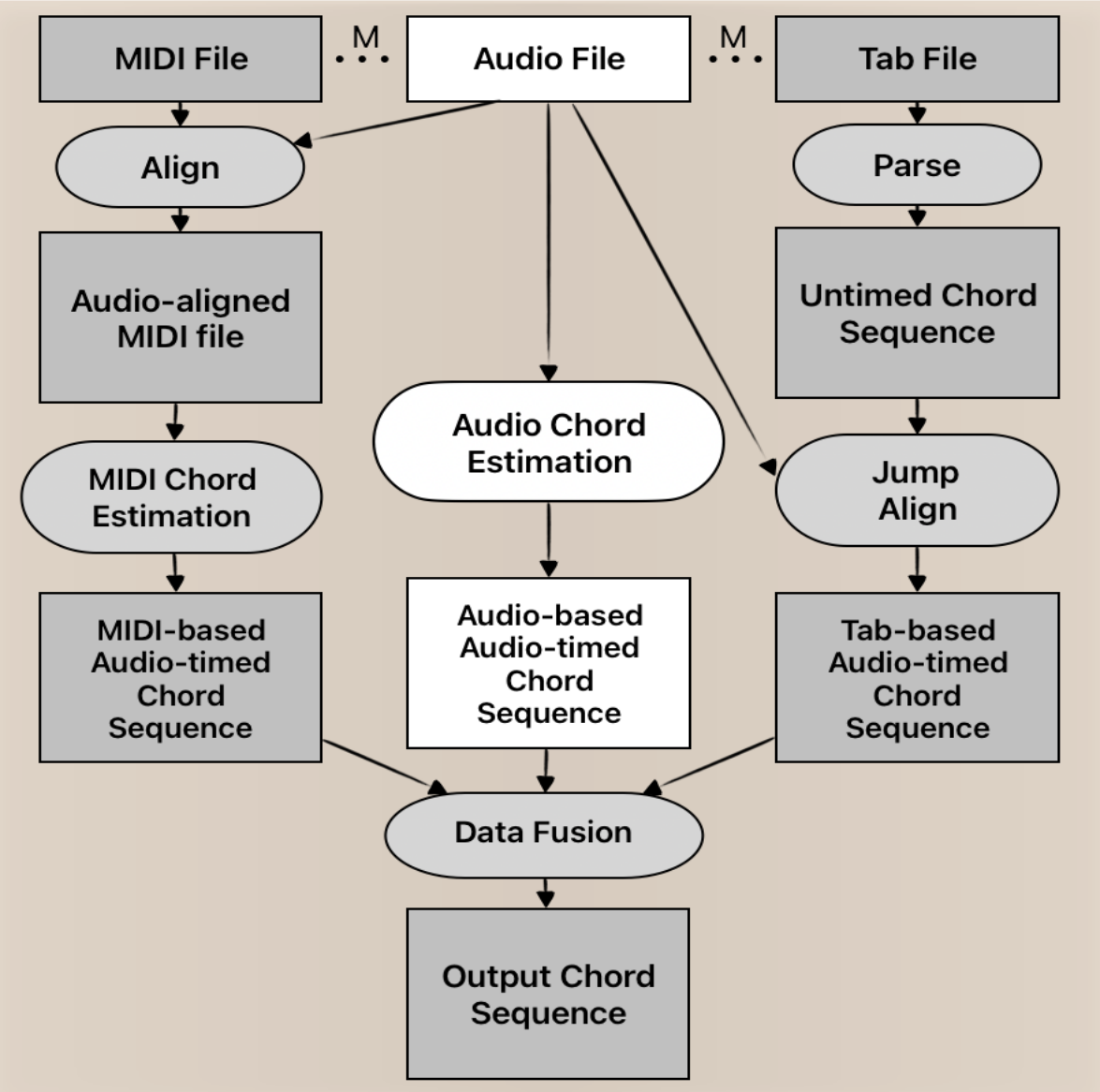


Fig. 2.20 Diagram illustrating the Decibel's framework. 'M' denotes the matching between various representations of the same track. Data formats are indicated by the rectangular shapes, while processes are displayed as rounded rectangles. The components shaded in grey highlight how decibel extends existing ACE methods. (created based on Odekerken et al 2020 Figure 3.1, Technical Report)

2.4.4 Overview of Chord Analysis Applications

At present, a number of standalone applications for chord analysis have been developed, including applications for mobile platforms, most of which use pre-trained models and AI algorithms to infer chord sequences. These include:

- **Chord AI**[1]: Operating on iOS and Android. This app offers a freemium (limited free plan with premium paid features) model along with a one-off purchase for full feature access
- **Chordify**[2]**:** a popular AI-driven chord detection application accessible via iOS, Android, and web, adopting a freemium model
- **ChordU**[3]: Similar to Chordify in both function and platform/services availability, which possible incorporates an AI driven approach, though details are not disclosed.
- **Moises AI**[4]**:** A freemium app accessible across iOS, Android, desktop, and web platforms.
- **Yalp**[5]**:** an exclusively web-based freemium application
- Yamaha **Chord Tracker**[6]**:** A cost-free application for iOS and Android, presumably incorporating AI technologies.

These applications typically present chords over waveforms or through diagrams for instruments like the guitar, ukulele, piano, and mandolin. Furthermore, they offer editing capabilities to refine analysis and save the result in a server-side database for quicker retrieval. Some of them even employ additional MIR methodologies such as beat tracking and instrument separation.

Other noteworthy applications include the **Melody Scanner**[7]**,** which is an automated music transcription tool that also displays chords on the musical staff and is available on Android and Web platforms, and the website of **ScalesChords**[8] that delivers a suite of musical tools, including a live chord finder that applies a Digital Signal Processing approach, similar to the ones discussed in subsection 2.4.2.

[1] https://chordai.net/

[2] https://chordify.net/

[3] https://chordu.com/

[4] https://moises.ai/products/moises-web-app/

[5] https://www.yalp.io/

[6] https://usa.yamaha.com/products/musical_instruments/pianos/apps/chord_tracker/features.html#product-tabs

[7] https://melodyscanner.com/

[8] https://www.scales-chords.com/live-chord-sound-finder/

2.5 Annotation Formats

The following section outlines the annotation formats employed in chord and beat recognition systems. Additionally, it details the chord label syntax that is necessary for the task of chord recognition among researchers.

CHORD SYNTAX

According to Harte (2010), a chord can be represented correctly using three components (see Fig. 2.21). Additionally, he proposed a chord root dependent syntax that influenced the MIREX ACE community as it can represent any chord of Western tonal harmony.

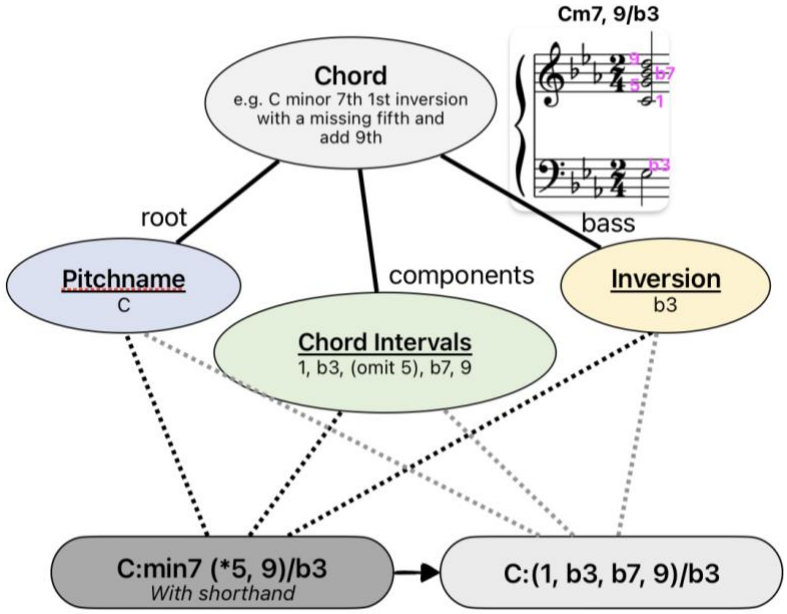


Fig. 2.21 Model for chord representation by its three main components: the root pitchname, the list of the component intervals and the bass note interval with a chord example (Note: The 5th is not omitted in the Musical Staff Notation)

From this model, the syntax can take be defined in four ways:

	Harte's Chord Syntax Alternatives Definitions					
	Root		Chord Intervals		Inversion	Description
1	root	:	shorthand (extra-intevals)	/	bass interval	Chord with shorthand mnemonic
2	root	:	(extra-intevals)	/	bass interval	Chord depicted from intervals
3	root			/	bass interval	Major chord implicit
4	N					No chord

Fig. 2.22 Four possible combinations of chord syntax definition

The **root** note is selected from the seven 'natural' musical notes {'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G'} with an accidental {sharp: # | flat: b | none}, yielding 21 unique root notes or 12,

considering enharmonic equivalents (for instance, D# is the enharmonic equal of Eb). In the special case that no chord is present, the 'N' symbol can be used, negating the need for additional syntax.

The syntax begins with the root pitch name, succeeded by a colon ':', which precedes the chord's interval components. These components can be represented by a combination of **shorthand** mnemonics consisting some of the most common chord types - similar to those in Jazz, and an **extra-intervals** list. The **shorthands** proposed by Harte, with their corresponding intervals are presented in TABLE 2.2 (additional shorthands can be added for other common chords if needed). Additionally, like in most hand written chord notation systems, a root pitchname written on its own is assumed to be a major chord, so the 'maj' shorthand can be omitted. The **extra-intervals,** is a comma delimited list of all the intervals consisting the chord, relative to the root note. This list can have an arbitrary length and the extra-intervals are included inside a parenthesis. Their propose is it to either indicate additional intervals added to a common chord when combined with a shorthand (Fig. 2.22 2nd case) or specify a chord that is not included yet in the shorthand chord types. Additionally, a special 'omit interval' symbol with the asterisk character (*), can be added before an interval, to denote a missing interval from a shorthand notated chord.

TABLE 2.2 Definition of shorthand notations for common chords, with their interval lists

Chordtype		Shorthand	Interval List
Triad Chords:	Major	maj	(1, 3, 5)
	Minor	min	(1, b3, 5)
	Diminished	dim	(1, b3, b5)
	Augmented	aug	(1, 3, #5)
Seventh Chords:	Major Seventh	maj7	(1, 3, 5, 7)
	Minor Seventh	min7	(1, b3, 5, b7)
	Seventh	7	(1, 3, 5, b7)
	Diminished Seventh	dim7	(1, b3, b5, bb7)
	Half Diminished Seventh	hdim7	(1, b3, b5, b7)
	Minor Major Seventh	minmaj7	(1, b3, 5, 7)
Sixth Chords:	Major Sixth	maj6	(1, 3, 5, 6)
	Minor Sixth	min6	(1, b3, 5, 6)
Extended Chords:	Ninth	9	(1, 3, 5, b7, 9)
	Major Ninth	maj9	(1, 3, 5, 7, 9)
	Minor Ninth	min9	(1, b3, 5, b7, 9)
Suspended Chords:	Suspended 2nd	sus2	(1, 2, 5)
	Suspended 4th	sus4	(1, 4, 5)

Finally, if the bass note is different from the root, it can be specified with the respective interval after the slash (‘/’) character. This interval is used to identify chord inversions, and is similar to the slash chords in standard music notation (Harte, 2010).

TRANSCRIPTION FILE FORMATS

The transcription file format that is mostly used for comparing and evaluating beat and chord recognition systems is an ASCII text format. For beat recognition systems this results in a file with the ‘.txt’ extension, that contains in each line the *beat-time* in seconds. In the case of downbeat recognition systems, a similar format is used, that either specifies the *downbeat-time* in seconds, or it includes every beat with its metrical position within the bar, which results in each line containing the *downbeat-time* followed by its *beat-metrical-position* (*beat-metrical-position* is represented by an integer, with 1 indicating the first beat). For chords, the format

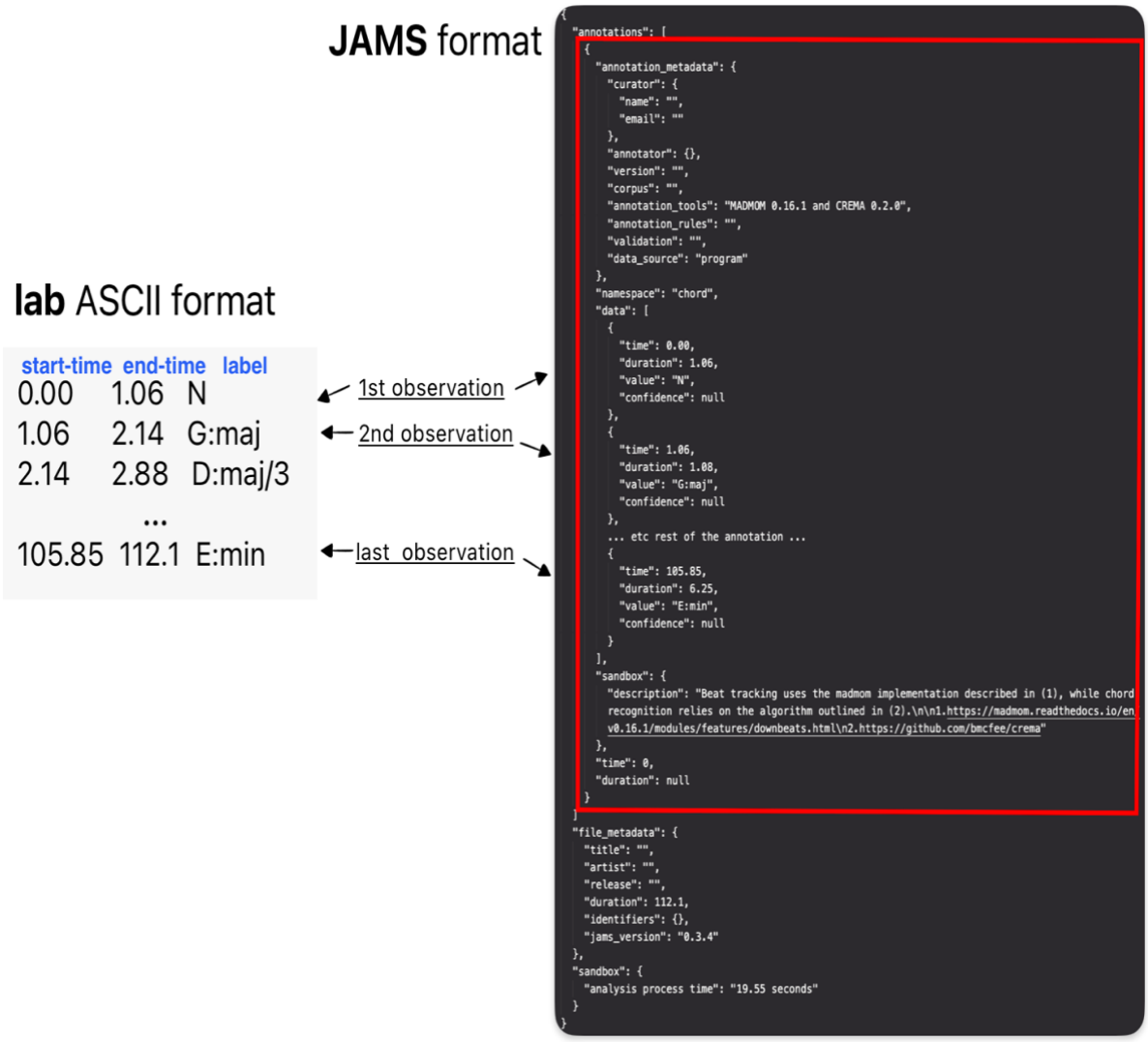


Fig. 2.23 ACE Transcription file formats: lab format (left), and JAMS format (right)

typically employs a '.lab' extension, where each line contains annotations that are segmented by *start-time end-time label*, with times in seconds and labels conforming to the syntax outlined previously (see Fig. 2.23 left part) (Harte, 2010).

An alternative file format that offers a structured JSON[1] schema for diverse MIR tasks and supports multiple annotations per file, is the JSON Annotated Music Specification (JAMS) format[2]. A JAMS file, can be either created or converted from a .lab file by incorporating the additional required fields.

The JAMS structure consists of three basic properties:

1. **file_metadata**: describing the audio file with the following fields:
    a. title: Name of the audio file.
    b. artist: Artist of the audio file
    c. release: Release details like album or date
    d. durations: Total length of the audio file
    e. identifiers: Unique identifiers for the audio file (e.g., MusicBrainz ID)
    f. jams_version: Version of the JAMS format used
2. **annotations**: an array of all the annotations for the given file. Each annotation contains:
    a. namespace: Category label for the annotation type
    b. data: Details of the annotation. Each observation contains an object with the following keys (time, duration, value, confidence)
    c. annotation_metadata: Information about the annotation process (corpus, version, curator, annotator, annotation_tools, annotation_rules, validation, data_source)
    d. sandbox: Field for additional, unstructured annotation data
    e. time: Start time of the annotation in the audio.
    f. duration: Length of the annotated segment
3. **sandbox**: an unrestricted place to store any additional data

An example of the above properties can be seen in Fig. 2.23, which presents a JAMS file produced by the Web application of the present thesis and contains a chord recognition output aligned at beat positions (further discussion in sections 3.2 and 4.1) (Humphrey et al., 2014).

[1] JavaScript Object Notation (JSON) offers a human-readable and easily parseable syntax that seamlessly integrates with Web technologies and various programming languages

[2] The reader can delve more in the documentation in https://jams.readthedocs.io/en/stable/index.html

Key fields in each JAMS annotation[1] are the **namespace** and **data**. **Namespace** specifies the annotation type (e.g., chord, beat, key_mode, lyrics), and **data** holds the annotation segments. While in the '.lab' chord format the arrangement of the data happens in each line as following: *start-time end-time label*, in JAMS an array contains multiple *objects* separated by a comma, and each object is having four properties: *time, duration, value and confidence*, that describe each observation as following (JAMS 0.3.2 documentation, 2021):

- The *time* field is a non-negative number denoting the time of the observation (in seconds) similar to the *start-time* or *end-time* for chord annotations or the *beat-time* in the beat annotations
- the *duration* is a non-negative number denoting the duration of the observation (in seconds), and matches the addition of the *start-time* plus the *end-time* for chord annotations while it is typically zero in the case of beat annotations
- the *value* depends on the namespace and carries the actual annotation (e.g., beat, chord, lyric, segment label). In the case of the 'chord' namespace, it contains chord symbols in the syntax described previously, while in the case of 'beat' namespace its value field is either null or it contains a number that indicates the metrical position within the bar of the observed beat, which is useful for the task of downbeat estimation
- the *confidence* gives an additional value about the certainty of each observation

2.6 Discussion

This section addresses the limitations and challenges encountered by beat and chord recognition algorithms, as documented in prior studies through observations from various researchers. It also covers important findings from other related research initiatives, providing a full picture of what is known in this area and suggesting directions for future work.

The *madmom* Python library for beat recognition reviewed in subsection 2.3.3 provides superior performance compared to alternative approaches (Böck & Davies, 2020). Moreover, MADMOM is publicly available, which was the reason it was used in the application of the present thesis. However, despite its performance, a common problem in beat tracking algorithms is that of dealing with expressive music performance, which comprises deviations

[1] An example of a JAMS annotation is outlined with the red rectangle in Fig. 2.23

of beat times and non-standard tempo changes, due to rubato, accelerando, etc. Additionally, the effectiveness of these algorithms generally declines when they encounter musical content that differs from the tracks used in their training datasets.

Pinto et al. (2021), proposed a solution to those problems, with a user-centric approach that involved minimal manual annotation by a hypothetical end-user to fine-tune (or specifically adjust certain weights of) an existing state of the art algorithm, in order to adapt to the musical piece being analyzed. As a result, they achieved high accuracy on individual challenging pieces where the current state-of-the-art systems are not effective, which demonstrated that the use of in situ fine-tuning over a small annotated region as a straightforward means to adapt a state-of-the-art beat-tracking system for this type of content (Pinto et al., 2021).

It is also important to highlight that the beat tracking systems described in subsections 2.3.2 and 2.3.3 are intended for offline use. Nonetheless, certain applications, such as liveBPM, require algorithms designed for real-time operation. An example of this is the Online Beat TrAckINg (OBTAIN) system, which showcases the effectiveness and reliability of an advanced algorithm implemented on a Raspberry Pi 3 for real-time beat tracking (Mottaghi et al., 2017).

In the ACE task, the systematic review of Pauwels et al. (2019) discussed key challenges and potential solutions for ACE systems. In their study, they found out that there is a lack of a comparative evaluation of the effects of the input features in deep learning models and they suggested using linear spectrograms over logarithmic ones to preserve harmonic integrity, or the creation of dual-input systems that utilize both CQT and Fourier spectrograms for enhanced analysis. Moreover, they highlighted that chord recognition systems should exploit relationships between chords and related musical concepts such as beats, key, genre, and melody, which has been proven to improve accuracy, pointing out that the main problem for the creation of such systems is the scarcity of multi-concept annotated data for the same audio file.

Additional challenges they highlighted is the skewed distribution of chord types in existing datasets, which biases performance assessments and necessitates new training and evaluation methods. A proposed solution is to augment data with less common chords and avoid the creation of datasets focused primarily on pop songs. Furthermore, it has been observed that chord recognition datasets suffer from ambiguity and subjectivity in their annotations. This is supported by a study by Koops et al. in 2019, which revealed that the agreement among

annotators on chord roots is as low as 76%, and 59% for the most complex MIREX chord vocabulary, reduced by an additional 5% when taking into account chord inversions. This disagreement is influenced by factors such as the musical background of the annotator, and personal preferences in transcription detail. This problem highlights the need for innovative assessment methods in ACE, where annotator subjectivity should be taken into account. Solutions could be personalized chord labels tuned to the idiosyncrasies of each user or analysis of annotations in a more functional level where annotators could agree on the function that chord has in the context of the key of the song (Pauwels et al., 2019).

A further noteworthy research initiative is the system developed by Carsault and Nika in 2022, that extracts chord labels from live audio streams and predicts possible continuations of the extracted chord sequences[1]. To achieve this, they added additional information in the training of the network by incorporating key and downbeat positions of the input chord sequence. Moreover, they introduced additional musical knowledge by training the chord extraction model using distances reflecting the relationships between chords (Carsault & Nika, 2022). Their approach differs from traditional ACE systems, which primarily focuses on the estimation of chords without considering their future progression or their harmonic function within their musical context and appears to be more closely related to the field of human machine music co-improvisation. However, their system integrates many of the ideas discussed in prior works and utilizes a new evaluation system[2] for functional chord assessment, advancing beyond right/wrong chord prediction, which is the standard in the MIREX ACE evaluation community. This evaluation system could be explored in ACE.

Finally, Korzeniowski & Widmer (2017) supported that training temporal models on a time-frame basis as the input representation, is 'futile on principle' and serves merely as a method to smooth predictions without contributing to the understanding of musical structure. They argued that frame-level analysis is insufficient because first-order models are restricted to predicting the next chord symbol within a brief 10 ms frame - a time span in which chords usually do not change. Moreover, even when chord changes do occur, they often involve sequences that extend beyond two chords, a complexity that first-order models are not designed to handle. Furthermore, even with more advanced systems like Recurrent Neural Networks

[1] https://github.com/carsault/chord_extraction_prediction_lib?tab=readme-ov-file

[2] https://github.com/jnika/ACE_Analyzer

(RNNs) that exceed simple first-order models, the benefits are marginal, because the dominance of self-transitions and the inherent difficulties in predicting chord changes without knowing when the chord change might occur, denies the musical modeling of the underlying structure. The suggested solution is to transition to hierarchical language models that operate at a more abstract level, such as the chord level, where the duration of a chord is reduced into a single instance in the final sequence. By adopting this approach, models can interpret and forecast musical structures in a manner that resembles human cognition, which naturally discerns patterns and predicts progressions over broader sections of music. Implementing such hierarchical models could significantly advance the field of ACE, making it more reflective of actual musical comprehension and responsive to the inherent structures within compositions.

3 Implementation

This chapter presents the architecture of the Play-Along-Together (PAT)[1] application and the development of its components. Its source code is made openly available through GitHub[2][3].

3.1 General Architecture

The application automates the analysis of chords and beats via a script residing on the MusiCoLab server. The result of the analysis is delivered to the client-side application (see Fig. 3.1). This processed information is then visually overlaid on the waveform display of the analyzed musical composition. The visualization of the chords is achieved with color-coded regions (audio segments) and the respective beats with markers (vertical lines) containing a label on the top with the respective chord symbol, thus providing an intuitive content representation to the user.

Following the automated analysis, users have the ability to refine the results for enhanced accuracy by correcting possible errors from the derived annotations. This is possible with the annotation tools and waveform editing tools provided, that can enable fine-tuning of beats by moving beat markers, insertion of new beats, chord modification or deletion using respective mouse events and the buttons provided.

Besides annotation tools, the application offers a click track to provide an audio representation of the detected beats, a snap-on-beat function for cursor positioning akin to a grid system in Digital Audio Workstations (DAWs), and additional audio player utilities such as playback control, zoom, volume adjustment, among others.

Finally, in an effort to enhance user experience, the PAT application incorporates several usability aids such as description tooltips, context-aware toolbars, context-sensitive and action-triggered icon buttons, visual skip cues, and keyboard shortcuts. These elements contribute significantly to the intuitive nature of the application, enabling users to navigate and use the app more effectively. To conclude, all these components will be explored in greater depth in the following subsections, as they constitute the core focus of this thesis.

[1] Play-Along-Together Web Application: https://musicolab.hmu.gr/apprepository/playalong3/

[2] Content Analysis Implementation & Evaluation: https://github.com/NeoklisMimidis/chord_and_beat_analysis

[3] Play-Along-Together Web Application Implementation: https://github.com/NeoklisMimidis/playalong3

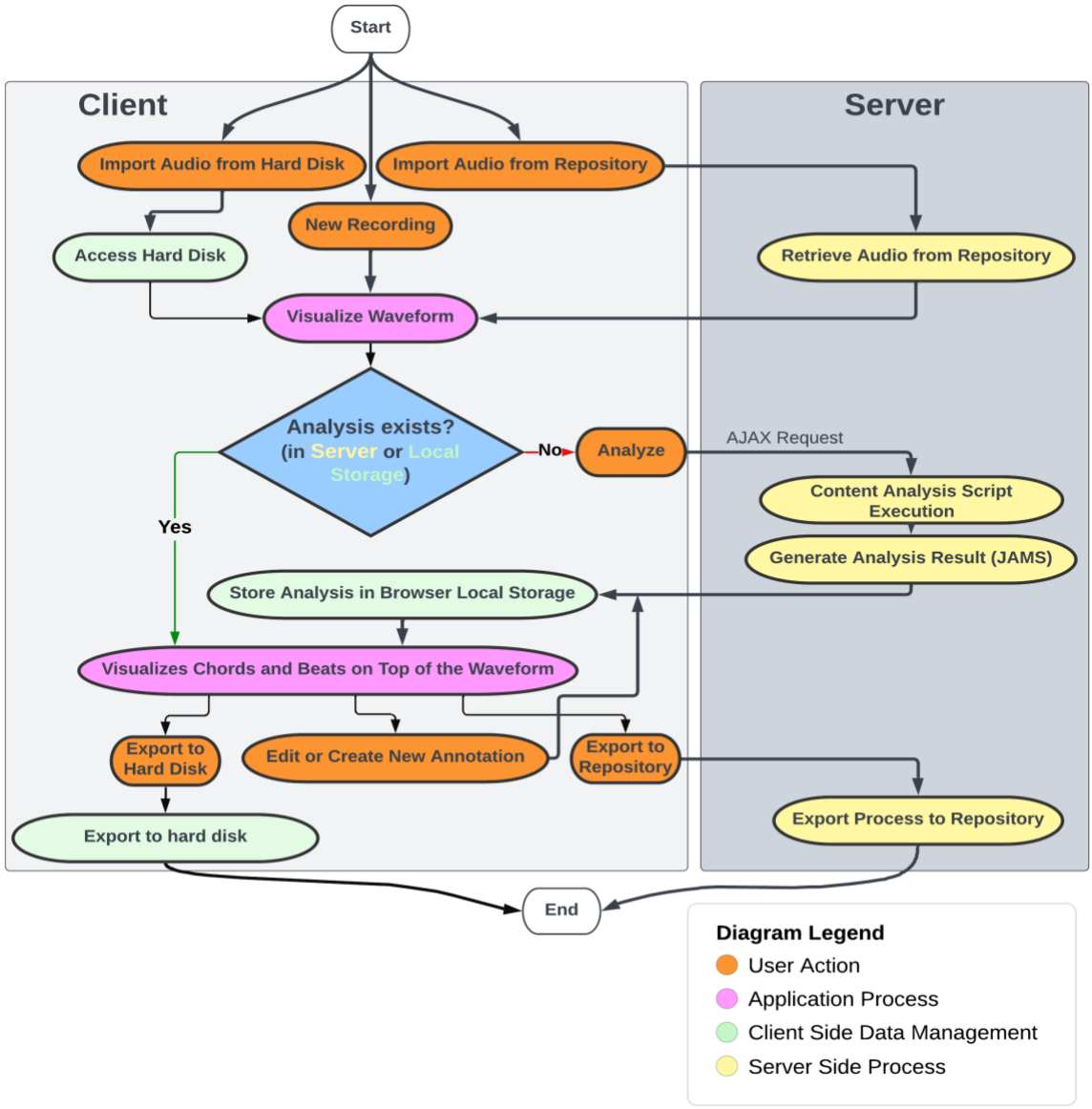


Fig. 3.1 Application architecture diagram presenting client-server communication for audio content analysis and visualization

3.2 Content Analysis

Chord and beat analysis in the PAT web application is invoked through an Asynchronous JavaScript and XML (AJAX) request from the client to the server. This client-server interaction necessitates the audio data to be available on the server, enabling the analysis script, which is written in Python, to execute. Depending on the originator of the backing track, i.e., a new recording, an audio file from the user's hard disk, or a recording imported from the MusiCoLab repository, the analysis process requires some extra time for transferring the audio file to the server.

3.2.1 Command-line Configuration

The analysis script is designed for command-line execution and accepts one positional argument along with five optional arguments. These arguments are detailed in TABLE 3.1, which provides a brief breakdown of the command-line arguments used. Note that Python 3.7 is a prerequisite for running the content analysis script due to dependencies imposed by the *madmom* library. Additional requirements are summarized in TABLE 5.1 of APPENDIX A.

TABLE 3.1 Breakdown of command-line arguments for chords and beats analysis script

	Component	**Explanation**
Interpreter	Python 3 Environment Path	Specifies the path to the Python 3.7 executable environment.
	Example	***/path/to/environment/py-3.7/python.exe***
Script	Analysis Script Path	Points to the location of the Python script for chord and beat analysis.
	Example	***/path/to/script/chordBeatAnalysis.py***
---		**Arguments**
Positional Argument	Path to the Input File	Specifies the audio file to be analyzed (type: string).
	Example	***'./path/to/test.wav'***
Optional Arguments	Output Path, Alignment Percent, Title, Artist, Method	Allows you to specify the output path, alignment percent, title of the track, artist of the track, and method for chord extraction (‘crema’ or ‘madmom’)
	Example	***--output_path '/path/to/export/location' --alignment_percent '0.41' --title 'My title' --artist 'Artist name' --method 'madmom'***
	Help	Shows the help message and exits the program.
	Example	***-h, --help***

3.2.2 Script Workflow

Implemented in Python, the script comprises three main functions: **`extract_chords()`**, **`extract_beats()`**, and **`add_chords_to_beats()`**. These functions perform chord and beat analysis and map chords to respective beat times (refer to TABLE 3.2 for further details).

TABLE 3.2 Summarizing key functions, methods, and outputs of the audio analysis script

Core functions	**Details (Method used & output)**	
`extract_beats``(data)` data: audio time series (loaded from *librosa*)	*MADMOM Functions*	`RNNDownBeatProcessor(), DBNDownBeatTrackingProcessor( beats_per_bar=[3, 4], min_bpm=55.0, max_bpm=215.0, fps=100, transition_lambda=100)`
	Output	**beats** (Numpy array, shape: (num_beats, 2), [**beats_times**, beats_numbers])
`extract_chords``(data, sr, method='crema')` data: audio time series (loaded from *librosa*) sr: sampling rate of given data method: either 'madmom' or 'crema'	*CREMA Functions*	`analyze(y=data, sr=sr)`
	MADMOM Functions	`CNNChordFeatureProcessor(), CRFChordRecognitionProcessor (fps=10)`
	Output	**chord_intervals**, **chord_labels** (Numpy array, [start_time, end_time], List, [chord_label])
`add_chords_to_beats``( beats_times, chord_intervals, chord_labels, percent=0.5)` **beats_times**: extract_beats() output **chord_intervals**: extract_chords() output **chord_labels**: extract_chords() output percent: choose the inner-beat alignment percentage to use	Output	**annotation_object** (jams.Annotation object with the 'chord' namespace)

3.2.3 Chord Analysis

Two machine learning models are available for chord analysis, both of which are implemented as open-source Python packages. The model selection is facilitated through an optional command-line argument, allowing users to choose the best package according to their requirements. CREMA is the package used by default for chord analysis and implements the research approach presented by McFee & Bello (2017). This model provides a comprehensive chord vocabulary, featuring 14 distinct chord variations. The alternative model used by the chord analysis script is implemented by the *madmom* library, which is based on the research approach presented by Korzeniowski & Widmer (2016). The vocabulary of this model includes only major and minor chords. Detailed information about the architectures and parameter settings of these chord models has been previously outlined in subsection 2.4.3 of this thesis.

Invoking these models through their Python packages is easy and straightforward. In the CREMA package, the `analyze()` function serves to initiate the chord recognition process. Equivalently, in the MADMOM package, the `CNNChordFeatureProcessor()` function is invoked to extract chord related features, followed by the `CRFChordRecognitionProcessor()` function to decode the extracted chord sequence and generate the output.

3.2.4 Beat Analysis

In the case of beat analysis, the script uses a specific sequence of function calls for the loaded audio track. Utilizing the `RNNDownBeatProcessor()` function, the system initially calculates the joint activation function of the respective beats and downbeats positions. Then, the `DBNDownBeatTrackingProcessor()` function is invoked to finalize the downbeat tracking, and generate the final output.

This process[1] identifies both beats and downbeat positions, but only the beat data is currently used in the content analysis of the PAT application.

[1] The selection of the downbeat tracking algorithm over a purely beat tracking one, was because it performed better (compared with other beat recognition models available in the Madmom library).

3.2.5 Heuristically Driven Chord-Beat Alignment

For the final required `add_chords_to_beats()` function (refer to TABLE 3.2), a heuristic method was used to align chords with beat times. This method involves a nested loop structure that examines each beat to determine the matching chord interval. The key to this approach is the **next_chord_matching_percentage** variable, which was established based on empirical findings and four main observations:

1. Beats occur more frequently than chord transitions and act as markers for chords.
2. Compared to template matching systems, machine learning algorithms provide chord data less frequently than beat data.
3. Even in well-annotated sequences, a discernable time lag exists between chord initiation and beat occurrence.
4. The selected algorithms (used in this thesis) set the chord resolution at 100 milliseconds, whereas the beat resolution was at 10 milliseconds

Intuitively, beats that are covered entirely by a single chord are directly matched to that chord. However, when chords extend beyond the duration of a single beat, the task becomes to accurately determine the alignment of each chord with the appropriate beat. Should the chord match the preceding or the following beat? Or should the decision be based on whether the duration of a chord within a beat exceeds 50% of the duration of that beat? Trial and error tests with a dataset of 415 audio files, revealed that setting the alignment at 0.41 (i.e. the percentage input argument of the add_chords_to_beats() function) provided the best results for the CREMA chord model and the MADMOM beat tracking algorithm. This effectively aligns a chord with the preceding beat if it starts within 41% of the duration of that beat. This choice and its effectiveness are further discussed in the 4.1.3 subsection of the thesis.

Subsequently, this function instantiates a jams.Annotation object using the 'chord'[1] namespace (a vital component of the Top-level JAMS object structure as discussed in section 2.5 Transcription File Formats), and the alignment of chords to beats are appended in the jams.Annotation object. Finally, the script was designed for command-line execution using argparse Python module, enabling parameter customization at runtime.

[1] The 'chord' namespace was selected instead of the 'beat' namespace (both options of the created JAMS object), due to the allowance for string-based chord labels in the value field. The 'beat' namespace only permits numerical entries and does not align with project requirements.

3.3 User Interaction and Graphical User Interface

This section focuses on the ‘Backing Track Audio Player’, a key part of the Play-Along-Together (PAT) web application. As seen in Fig. 3.2, PAT has three main areas: ‘PAT Controls’, the ‘Backing Track Audio Player’, and the ‘Multitrack Recordings’. This section focuses on the ‘Backing Track Audio Player’ and how it interacts with the remaining PAT components.

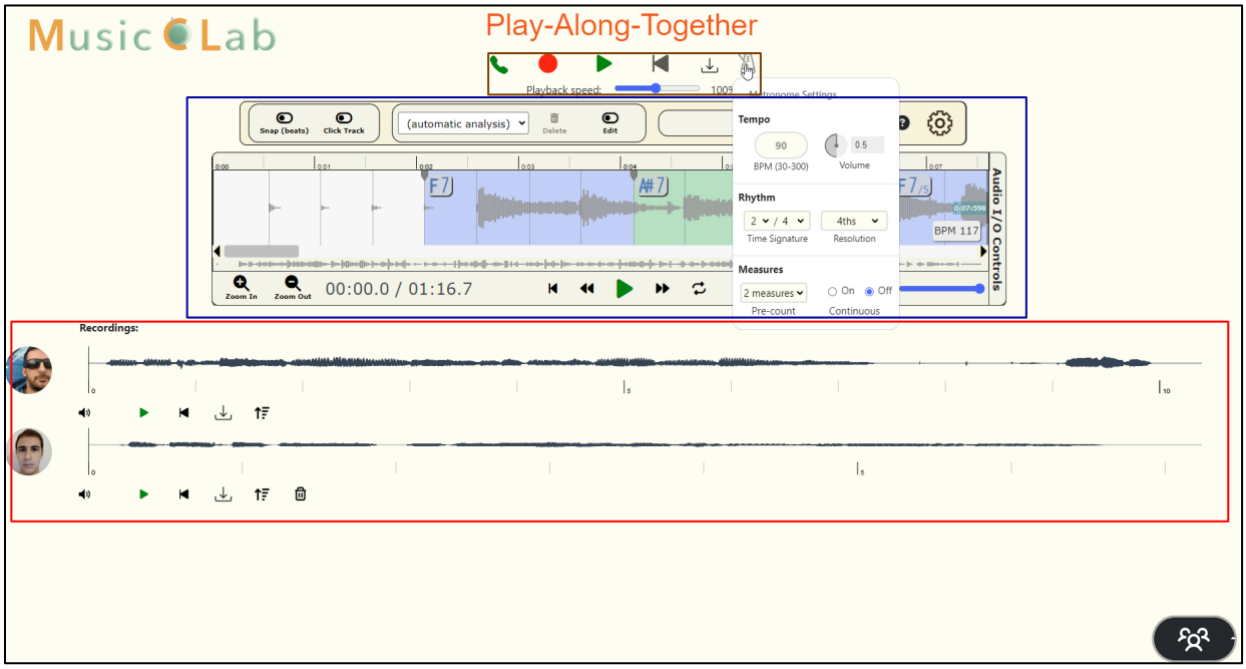


Fig. 3.2 The three main parts of PAT. Brown: ‘PAT Controls’; Blue: ‘Backing Track Audio Player’; Red: ‘Multitrack Recordings’

The ‘Backing Track Audio Player’ consists of two main components: the ‘Audio Player’ and the ‘Context-aware Toolbar’. Subsection 3.3.1 presents the ‘Audio Player’, while 3.3.2 focuses on the ‘Context-aware Toolbar’. Subsection 3.3.3 presents ‘Annotation Editing’ and describes the editing tools available within the player. Finally, subsection 3.3.4, entitled ‘Additional Features and System Integration’, presents supplementary noteworthy features and how they integrate with the rest of the PAT application.

Throughout this section, screenshots and images are provided to describe the functionality of the Graphical User Interface (GUI). Packages dependencies for the web application scripts are summarized in TABLE 5.2 of APPENDIX A.

3.3.1 Audio Player

The audio player is composed of three parts: 1) the ‘Audio I/O Controls’, a collapsible sidebar for managing file Input and Output; 2) ‘Visual Representations of Musical Content’, containing the main waveform display of the audio file, which later visualizes beats and chords upon successful analysis, as well as a timeline above the waveform and a minimap situated below; and 3) the ‘Audio Player Controls’, which enable user interaction with the audio file playback. These parts are illustrated in Fig. 3.3.

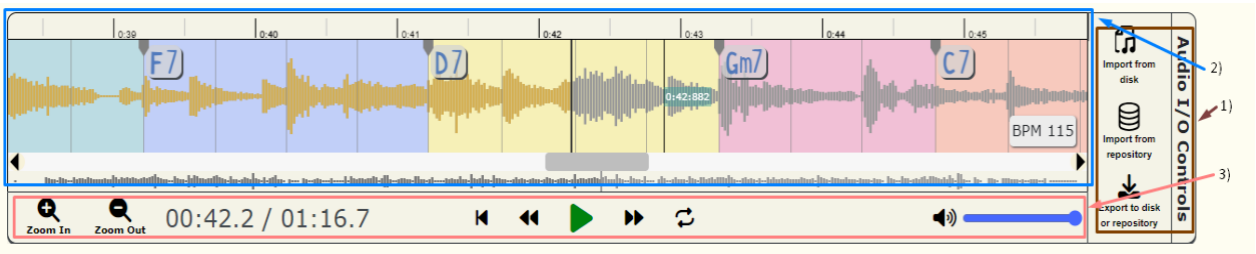


Fig. 3.3 The three parts of the ‘Audio Player’

AUDIO I/O CONTROLS

The ‘Audio I/O Controls’ is a collapsible sidebar and it is in a closed state upon initial application launch (refer to Fig. 3.4a). Hovering over this sidebar, a tooltip emerges providing a concise explanation of its functionalities (refer to Fig. 3.4c).

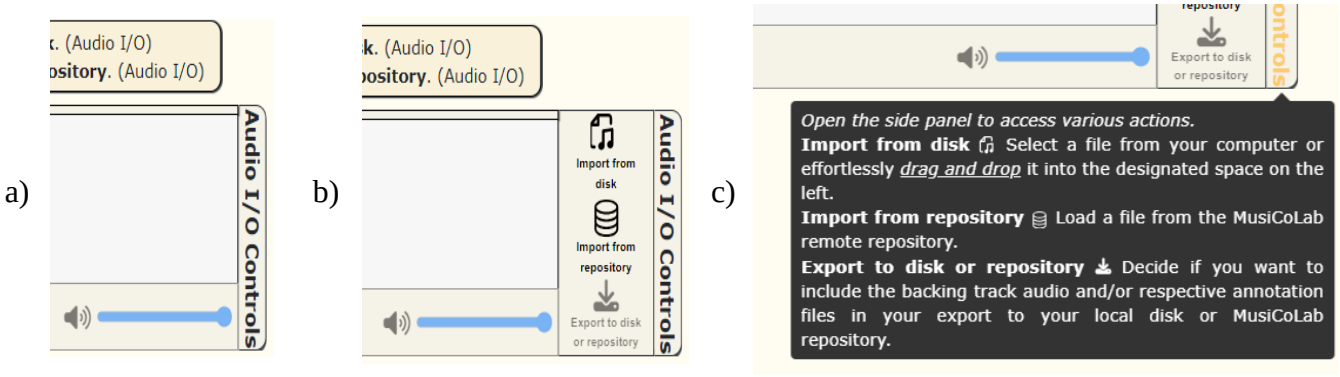


Fig. 3.4 ‘Audio I/O Controls’ sidebar in: a) closed state b) opened state c) description tooltip

Upon expanding this sidebar, users gain access to a suite of file input and output controls that offer the following capabilities (refer to Fig. 3.4b and Fig. 3.5 for additional visualizations):

1. **Import from Disk**: Users have the option to manually select a file from their local computer storage for upload. Alternatively, they can use the drag-and-drop interface situated on the sidebar’s left-hand section for a more effortless upload process.

2. **Import from Repository**: Users can import assets directly from MusiCoLab's remote repository through a searchable list, offering access to public, private, or course-specific collections. Importing from private collections is restricted in collaborative sessions, and course-specific files are accessible only when the app is launched from a course.
3. **Export to Disk or Repository**: This feature allows users to export the loaded backing track, with or without the analysis file, to local storage or the MusiCoLab repository. Exporting to a course-specific repository is possible only when the app is initiated from a course. Obviously, Export to Disk or Repository, is disabled until a file is loaded (Stage 2: Audio Loaded), and exporting an annotation file is restricted until an analysis is completed (Stage 3: Analysis Loaded).

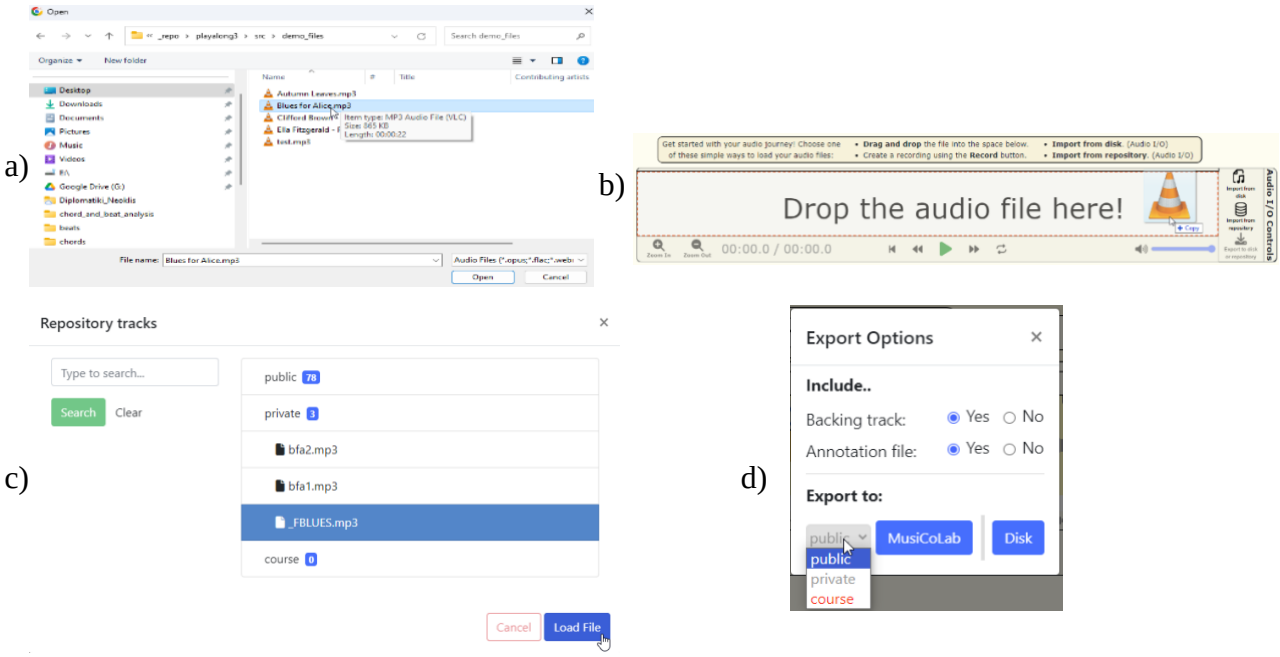


Fig. 3.5 Audio I/O Controls capabilities: a) 'Import from Disk' via file Select, b) 'Import from Disk' via drag and drop, c) 'Import from Repository', and d) 'Export to Disk or Repository'

Finally, the process of loading an audio file is visually represented through a progress bar in the waveform container (see Fig. 3.6), except in the case of Importing a file from the repository, where a loading spinner and a cancel option are displayed instead, in the relative menu.

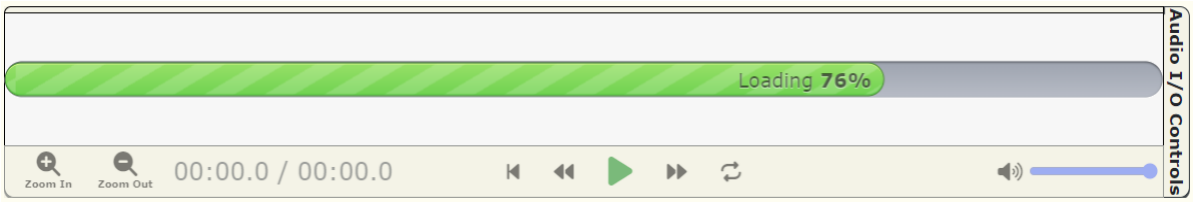


Fig. 3.6 Loading progress bar in the waveform container

In the initial stage (Stage 1: Initial, see Fig. 3.7a), before any audio signal is loaded, the waveform container is blank.

Supplementary visual components for waveform navigation, i.e. the waveform, the minimap and the timeline become visible – inside, below and above the waveform container, upon loading an audio signal (see Fig. 3.7b) The minimap serves as a condensed representation of the entire audio file and functions as a scrollbar for the main waveform, while, the waveform container displays a portion of the audio, adjustable by the zoom controls. The timeline provides a time-oriented view, displaying time in either minute-second-decisecond format (e.g., 2:52.5) or minute-second format, based on the current zoom level. Also, hovering the mouse over the waveform displays the current time position.

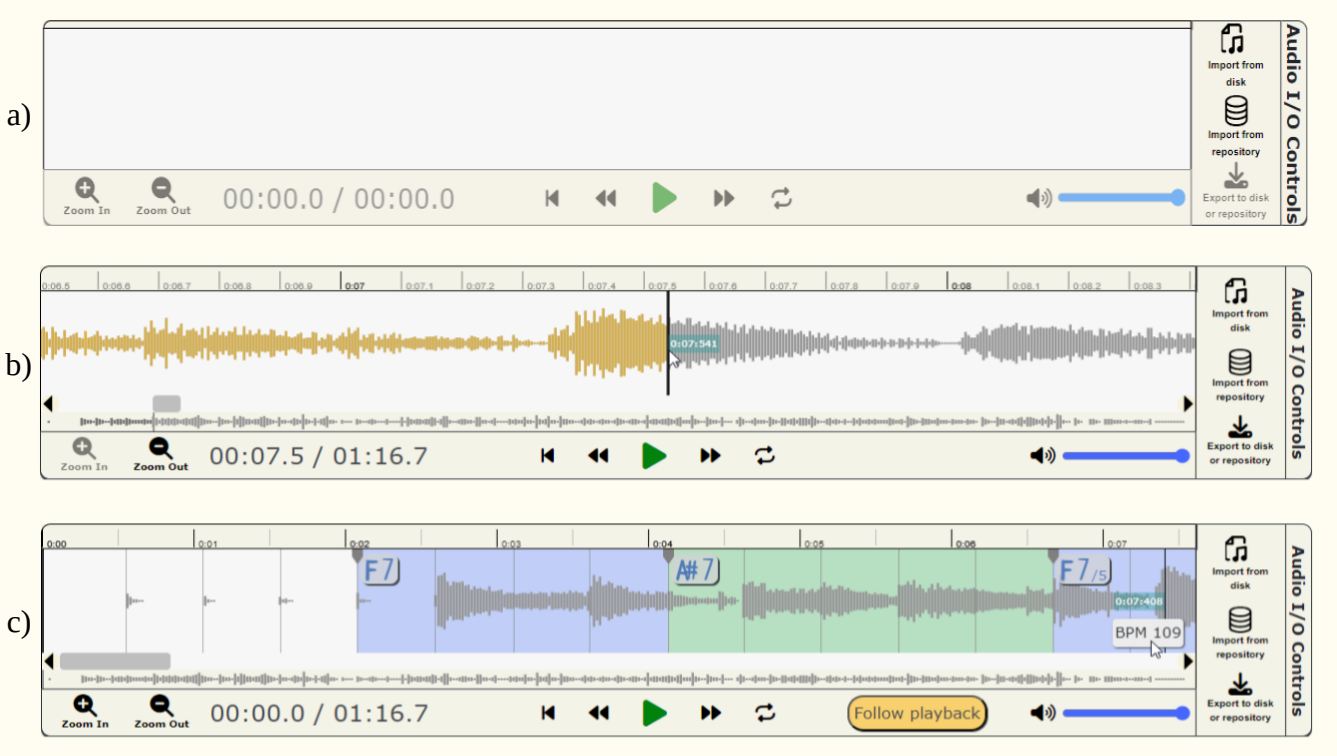


Fig. 3.7 Audio Player stages: a) Initial, b) Audio Loaded c) Analysis Loaded

Upon successful analysis (Stage 3: Analysis Loaded), beats and chords are visualized through Wavesurfer's markers and regions plugins. This is achieved by fetching the server-side JAMS-format annotation data (refer to 3.2 section), and utilizing for each observation the *time* and *value* fields - which correspond to the chord and beat segments. For further details on the annotation format and encoded labels, please refer to section 2.5 Transcription File Formats.

The *time* fields, are used directly to represent the beats through vertical lines on top of the waveform, via the markers plugin of the Wavesurfer.js library. The *value* fields, carry the chord symbols in an extended version of the syntax defined by Harte et al.: {Root: Shorthand (extra intervals) / bass note} and are modified before usage. This modification involves encoding them using a custom OpenType Font (OTF) with SVG symbols. The font for the representation of Roots and Shorthands is provided by the software development team of the Genius Jam Tracks (GJT) [1] applications, who are collaborating partners of the MusiCoLab research project.

In the application, the *root value* field serves a dual purpose: first, it displays the matched chord root at each beat using Wavesurfer's markers plugin, and second, it influences chord colorization through the Wavesurfer's regions plugin, enabling easier chord differentiation, as seen in Fig. 3.8.



Fig. 3.8 Representation of the 23 chord regions comprised from the total amount of root notes plus special symbols (14 colors are unique due to enharmonic notes are sharing color)

The *shorthand value*, a mnemonic for chord types, converts into one of the 63 GJT-provided SVG symbols. This conversion, exceeds the 14 templates in the CREMA model used for analysis. To address this discrepancy, additional shorthands were created based on MIREX standards, following Harte's methodology, allowing for wider chord representation and future compatibility in annotation evaluations (until an extended task vocabulary is developed, they will be simplified to Tetrads and subsets). In the source code, an array called Variations consisting of a total of 67 chord type objects was created. Each object contained properties for 'shorthand', 'simplified', 'description', and 'encoded' fields (see Fig. 3.9a) (where the 'encoded' property matched the SVG font). The 'description' property provides the full name of the chord and serves as a tooltip - when the user hovers over the chord region, as seen in Fig. 3.9b. The TABLE 5.3 of APPENDIX B, contains all the necessary information for various matching processes among shorthand, simplified, description, and encoded formats of chord variations. Finally, the *bass* note, appearing after the slash (/), serves as an optional interval to clarify the chord inversion (in Fig. 3.12b, the chord F7/5 indicates a 2nd inversion - note C is in the bass).

[1] https://geniusjamtracks.com

Additionally, as illustrated in Fig. 3.9b, if a beat (vertical marker) contains a chord identical to the previous one, the chord label is not displayed, providing a cleaner user interface. Similarly, in cases where there is no chord, only the beat is displayed. Furthermore, the BPM (Beats Per Minute) indicator, located in the bottom-right corner of the waveform (as shown in Fig. 3.7c), provides a real-time measurement of the tempo. It is calculated by dividing 60 seconds by the time between two consecutive beats and is updated continuously during playback.

Activating the Edit toggle modifies some of the visual representations of the musical content. These changes will be presented in subsection 3.3.3, 'Annotation Editing'.

a)

```
{
  shorthand: 'min7',
  simplified: 'm7',
  description: 'Minor seventh',
  encoded: 'b',
},
```

b)



Fig. 3.9 a) One object in the 67-Item variations array; b) Blue: Tooltip according to description field; Red: Repeated chords; Brown: 'No chord' case

AUDIO PLAYER CONTROLS

In the initial stage (Stage 1: Initial), the audio player's controls are disabled, until the audio is successfully loaded (Stage 2: Audio Loaded). The control layout, from left to right, includes:

1. **Zoom in/out buttons**: adjusting the zoom level.
2. **Time display**: showing the current playback cursor position and total audio duration.
3. **Standard playback controls**: Stop, Skip backwards and forwards (with visual cues on the waveform when pressed), Play/Pause
4. **Loop button**: enabling looping in two ways; looping the entire audio or a selected area.
5. **Follow Playback button**: visually reverts to the current playback position, if the user is scrolled away (displayed only when required).
6. **Mute/Unmute button**: toggling on/off sound.
7. **Volume slider**: adjusting audio track volume.

Additionally, for an enhanced user experience and easier audio playback, all controls are equipped with shortcuts and display tooltips when hovered.

Upon entering Stage 3 (Analysis Loaded), the buttons retain their functions, but some gain additional capabilities when combined with the 'Snap (beats)' toggle. Specifically, the Skip backward and forward buttons are skipping to the previous and next beats respectively, and the selected area aligns to the closest beat positions when the loop is enabled.

3.3.2 Context-aware Toolbar

In Stage 1: Initial before loading an audio track, the bar above the waveform representation, guides users on necessary startup actions, serving as a preface instructions menu, toward the subsequent steps (see Fig. 3.10a).

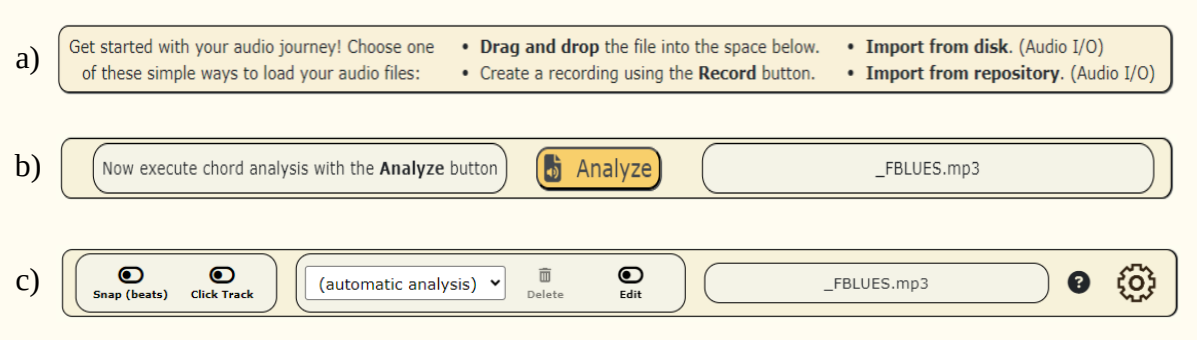


Fig. 3.10 Context-aware toolbar across different application stages: a) Initial; b) Audio Loaded; c) Analysis Loaded

After successfully loading the audio track (Stage 2: Audio Loaded), the bar transforms and features a centrally-located ‘Analyze’ button. Adjacent to this button, a brief instructional text on the left and the filename of the uploaded audio file on the right are displayed (see Fig. 3.10b). Upon clicking ‘Analyze’, a modal window appears, informing the user that the analysis process will take some time (see Fig. 3.11a). Affirmative selection initiates the content analysis processes as detailed in 3.2 section via an AJAX request to the server. Analysis time required is estimated from past analysis results using linear regression and is shown as a progress bar on the client interface (see Fig. 3.11b). It is important to highlight that if the audio has been imported from the repository, the application will first ascertain whether an existing annotation file (named audio_filename.jams) is available. If so, the process delineated for this stage is bypassed, and the interface progresses directly to the next stage (Stage 3: Analysis Loaded). The same occurs if a prior annotation is found in the browser’s local storage, where then the user is prompted to load it, as illustrated in Fig. 3.11c.

Upon successful analysis, the bar transitions once more (Stage 3: Analysis Loaded). This time, it becomes a toolbar replete with annotation tools divided into two modes, dependent upon the status of the ‘Edit’ toggle. As illustrated in Fig. 3.10c, the left-hand side features a ‘Snap (beats)’ toggle and a ‘Click Track’ toggle. Centrally, there is the ‘Annotation List’, accompanied by a ‘Delete’ icon and an ‘Edit’ toggle. On the toolbar’s right side, the loaded audio filename, a ‘Question’ icon, and a ‘Settings’ menu icon, are displayed.

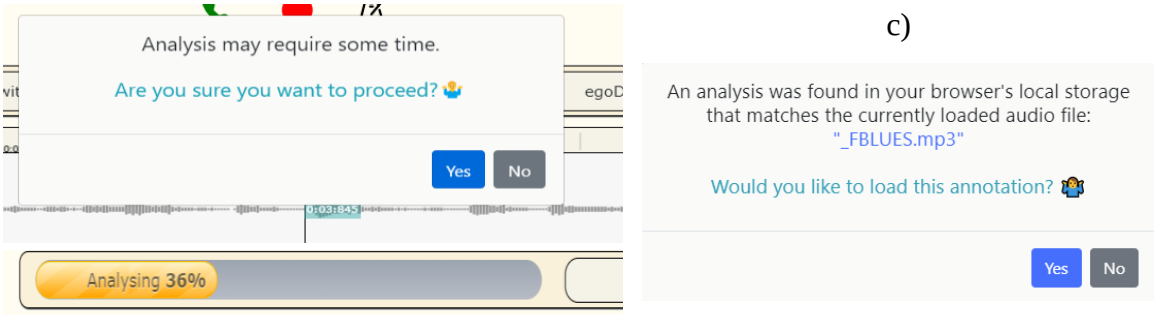


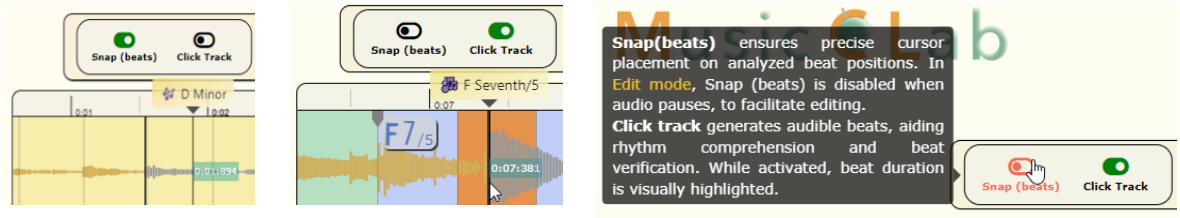
Fig. 3.11 Analysis confirmation; b) Analysis progress bar; c) Local storage analysis load

Furthermore, the toolbar's context shifts again upon activating Edit mode (when the 'Edit' toggle is enabled), a topic detailed in the dedicated 3.3.3 Annotation Editing subsection.

SNAP (BEATS) & CLICK TRACK

Located on the toolbar's far left, 'Snap (beats)' and 'Click Track' enhance user interaction with beat-tracking analysis. Hover-over tooltips explain their functions, as shown in Fig. 3.12c. The 'Snap (beats)' when activated, ensures precise cursor alignment with the analyzed beat positions (vertical lines on top of the waveform), when the user clicks near them (a feature which is found in Digital Audio Workstations - DAWs). This feature deactivates in Edit mode when audio is paused, allowing other click-dependent features to function freely. Additionally, this tool modifies some audio player functionalities like looping and skipping forward or backward, as presented in 3.3.1 subsection.

Conversely, 'Click Track' generates audible clicks for each analyzed beat, providing essential auditory cues for evaluation of beat accuracy. When active, each beat's duration is visually emphasized on the waveform, as in Fig. 3.12b. Unlike traditional metronomes, 'Click Track' follows the tempo of the backing track and its potential deviations.



a) b) c)

Fig. 3.12 a) 'Snap (beats)' activated; b) 'Click Track' activated; c) Respective tooltip

ANNOTATION LIST, DELETE

At the center of the toolbar lies the annotation list, designed for visualizing the selected annotation of the current audio file. This feature was inspired by commercial applications like

Chordify[1], and it leverages the JAMS file structure, which can include multiple annotations within a single .jams file. The default annotation, named ‘automatic analysis’, is always displayed in the list and is generated by the Python script executed on the server. For instance, as seen in Fig. 3.13, there are two annotations: one labeled ‘automatic analysis’ and another named ‘Edit by Neoklis’, which was created after the original annotation was edited.

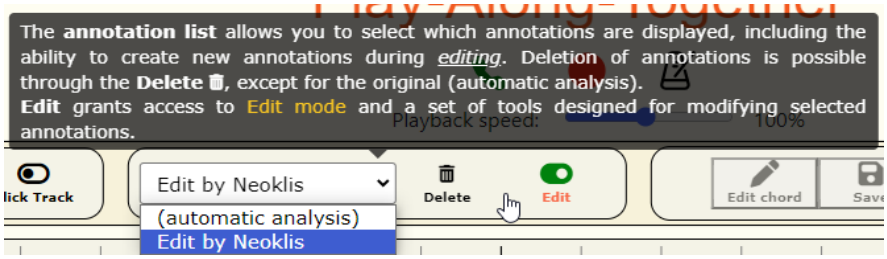


Fig. 3.13 Toolbar center featuring the Annotation list, Delete button, and Edit toggle

PLAYER SETTINGS MENU

The Settings Menu provides features that aim to improve usability and offer educational value:

1. **Follow Playback**: Offers two modes which manipulate the playback cursor’s position and behavior when playing the audio file. It provides a more customized playback experience enhancing navigation and focus, while allowing easier chord reading. The modes are:
   - Scroll Mode: The waveform scrolls at user’s defined scroll point.
   - Page Turn Mode: The waveform moves at the reset point, when reaching the turn point.
2. **Chord Editor Categories**: Shows or hides the chord categories, like Major, Minor, Dominant, etc. in the chord editor during Edit chord. Grouping chords by category aids in quick recognition and learning. The Chord Editor can be seen in Fig. 3.17

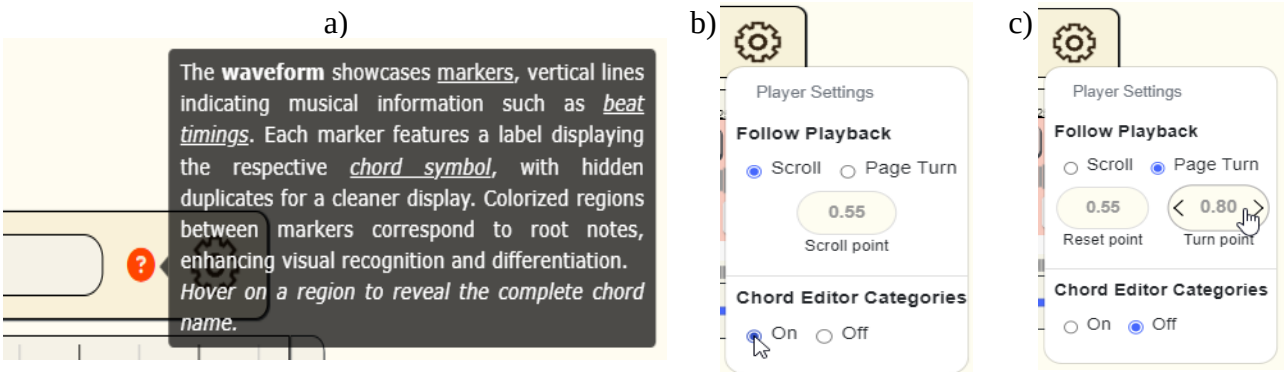


Fig. 3.14 a) The question icon tooltip explains how the beats and chords are visualized in the waveform. b) The available options of the Player Settings menu

[1] https://chordify.net/

3.3.3 Annotation Editing

In edit mode, the display of the main waveform in the ‘Audio Player’ and the ‘Context-aware Toolbar’ is transformed to allow for editing. In this mode, users can make various changes like Adjust Beat Timing, Edit Chord, Add Beat, or Delete Beat. Any modification sets the application into ‘Editing’ status, which remains active until the user confirms changes with ‘Save’ or reverts by selecting ‘Cancel’.

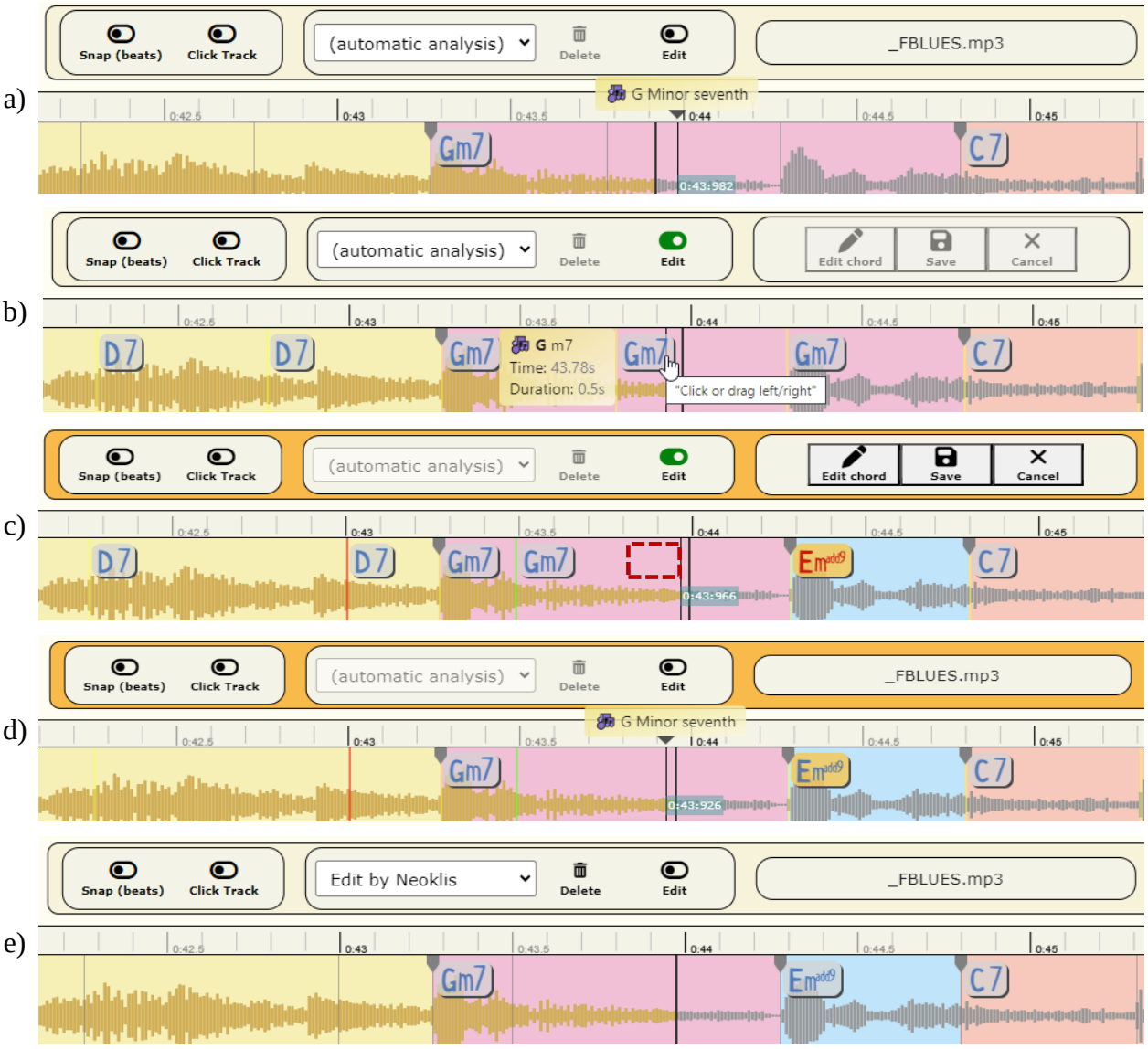


Fig. 3.15 Comparing Backing Track Player States - a) Edit Mode and Editing Status both off (before any modification); b) Edit Mode on, Editing Status off; c) Edit Mode and Editing Status both on; d) Edit Mode off, Editing Status on; e) Edit Mode and Editing Status both off, (after successful modification).

AUDIO PLAYER'S MAIN WAVEFORM IN EDIT MODE

- In the main waveform, beat lines change to yellow, indicating they are editable. All marker labels show up and can be selected for modification (see Fig. 3.15b).
- Waveform marker tooltips now display information based on the 'simplified column' in table FGF, along with the beat's starting time and total duration, aiding the annotation editing process.

In Edit mode, user actions are as follows:

1. **Adjust Beat Timing**: Dragging and dropping a marker changes its timing and turns the vertical line red (see Fig. 3.15c 2nd marker).
2. **Add Beat**: Double-clicking on the waveform adds a new beat, automatically assigning the same chord type as the previous (see Fig. 3.15c 4th marker).
3. **Delete Beat**: Right-clicking on a marker shows a popup menu offering the option to delete the selected marker and its corresponding chord (see Fig. 3.16 and Fig. 3.15 red rectangle, after deletion).
4. **Edit Chord**: Selecting a marker label activates the 'Edit Chord' button and turns the chord symbol red (see Fig. 3.15c 5th marker). Clicking this button opens the chord editor, which offers 67 chord types and roots (see Fig. 3.17). To confirm the changes, the user can press the 'Apply' button on the chord editor's bottom-right corner. A chord that has been changed is illustrated in the waveform by a light orange background color on the marker label.

With these functionalities and visual identifications, the application tries to offer a user-friendly experience for editing annotations.

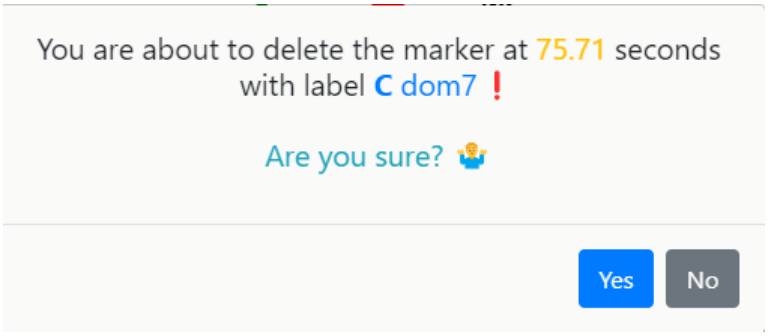


Fig. 3.16 Popup menu showcasing 'Delete Beat' action

a)



b)

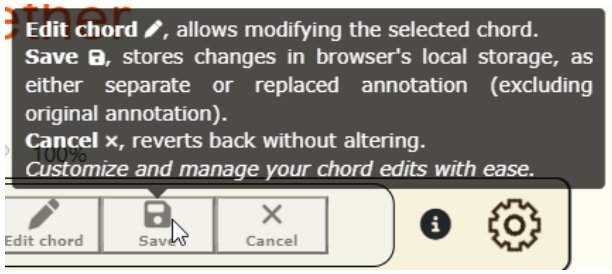


Fig. 3.17 Chord editor (a) without chord categories (b) with chord categories. The chord editor, accessible during chord editing, can display chord categories based on user selection in settings menu. Additionally, it offers tooltip descriptions for hovered symbols. The fonts for chord types, roots, and accidentals align with those used in marker labels.

CONTEXT-AWARE TOOLBAR IN EDIT MODE

- When Edit mode is activated, the 'Context-aware Toolbar' changes its context. Instead of displaying the filename and a 'question' icon (see Fig. 3.14), it now shows a sequence of annotation editing tools 'Edit Chord', 'Save', 'Cancel', and an 'info' icon. Similarly, these tools and the info icon feature hover-over tooltips, as depicted in Fig. 3.18, offering detailed insights into available functionalities.

a)



b)

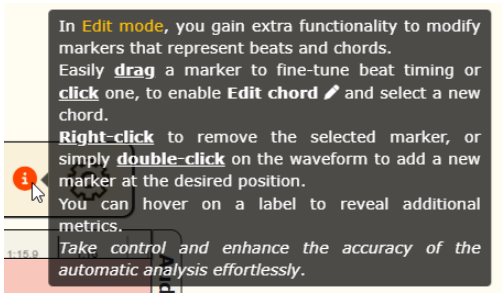


Fig. 3.18 Tooltips for Edit Mode: (a) editing tools and (b) Info icon

❖ After Editing status is enabled:

- The ‘Save’ and ‘Cancel’ buttons become active, whereas the ‘annotation list’ and ‘Delete’ annotation button are disabled because the user is editing the selected annotation.
- The bar’s color turns orange, indicating the application is in ‘Editing’ status.

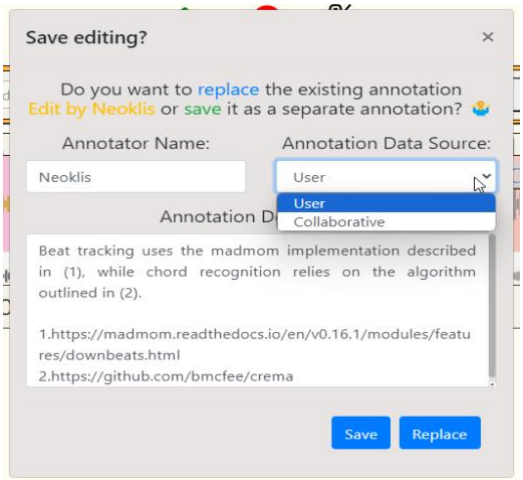
Toolbar actions are as follows:

1. **Edit Chord**: This button functions as previously described for editing the selected chord.
2. **Save**: Clicking this opens a modal prompt for entering ‘Annotator Name’, ‘Annotation Data Source’, and ‘Annotation Description’. If the user saves - either as a separate or replaced annotation - the corresponding JAMS file and the browser’s local storage are updated (Fig. 3.19a).
3. **Cancel**: Cancels all the changes made during the editing session (see Fig. 3.19b)

The default analysis – server request, named ‘automatic analysis’, is a special case that has additional logic applied. This annotation cannot be replaced during the ‘Save Editing’ process nor deleted using the ‘Delete’ button. Additionally, the ‘Annotation Description’ includes information about the models selected for the analysis.

Finally, it is important to emphasize that the user during ‘Editing’ can switch back and forth between the Edit mode using the Edit toggle (see Fig. 3.15d). This allows for utilization of both displays until the user finalizes changes by pressing either the ‘Save’ or ‘Cancel’ button. Only upon a successful ‘Save’ or ‘Cancel’ does the ‘Editing’ status become disabled, reverting the ‘Context-aware Toolbar’ back to its default color (see Fig. 3.15e).

a)



b)

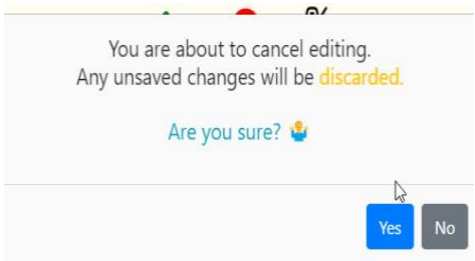


Fig. 3.19 Popup menus showcasing (a) ‘Save’ and (b) ‘Cancel’ actions

3.3.4 Additional Features and System Integration

This subsection briefly demonstrates how the backing track player integrates with the broader PAT application and contributes to a collaborative, multifaceted user experience.

INTEGRATED FEATURES IN PLAY-ALONG-TOGETHER (PAT)

The broader PAT application offers several functionalities that directly or indirectly affect the backing track player. These include:

- 'Playback Speed' slider: Allows users to modify the speed of the tracks.
- 'Multitrack Playback' buttons: Provides options to play, pause, or stop all tracks.
- 'Mix and Download' button: Enables users to combine all tracks into a mixdown

Additionally, the PAT's metronome tempo is influenced by the beat tracking analysis of the backing track. Moreover, a newly recorded track can be used as the backing track and gain access to all integrated functionalities, such as analysis and editing.

COLLABORATIVE FEATURES

The PAT application is designed to support collaborative music-making through the Yjs[1] shared editing framework:

- Track Sharing: All tracks, including the backing track, are shared among collaborators
- Collaborative Editing: Editing changes made are instantly shared to all users. Also, while editing functionalities are limited to one user at a time, the system allows with the 'Edit' mode toggle, a seamless switch between collaborators.

SERVER-SIDE OPERATIONS

To support the functionalities of the backing track player, specific PHP scripts were implemented in the server of the MusiCoLab (backend of the application):

- Content-analysis Script: Responsible for running the analysis on the backing track.
- File Management Scripts: Handling tasks such as exporting or deleting necessary files from the server repository and database

[1] https://yjs.dev/

3.4 Using the PAT Application in Music Education

This section attempts to highlight the potential use of the PAT application in an educational context. In the following, three case studies are presented, which emphasize the benefits of using the PAT application to carry out teaching and learning tasks, over conventional software applications for the same purpose.

3.4.1 Asynchronous Student Practice

This scenario highlights a student's individual learning journey, emphasizing the PAT application's analysis, playback, and editing capabilities.

Ella, an enthusiastic music student, discovers a new song on the radio that captivates her. Eager to practice it, she finds the song in audio file format and imports it into the PAT application. Using the 'Analyze' button, she executes the content analysis feature, which provides chord symbols for each beat of the track. When Ella needs more detailed information about a chord, she hovers over the detected chord on the waveform and tooltips display the full chord names. Additionally, Ella uses the playback speed control to adjust the tempo to her learning pace and with the help of the loop and zoom controls she focuses on specific sections of the song. During her practice, Ella notices a few errors in the analysis. In order to correct those inaccuracies (in chords or beats), she uses the 'Edit' functionalities and the 'Click track' and 'Snap beats' features, which finds particularly helpful for precise editing. Upon completing her session, Ella saves her newly annotated version and export it to her private collection within the MusiCoLab repository, for future reference.

3.4.2 Asynchronous Assignments combined with Synchronous Lessons

This scenario underscores the teacher-student dynamic in enhancing students' aural skills development, highlighting the PAT application's effectiveness in combining analysis with educational feedback in a group learning context.

Dimitris, a dedicated music teacher, aims to enhance the aural skills of his students. As a supplement to traditional ear training methods, he assigns his students an engaging exercise: they are to select various audio tracks of their choice and load them into the PAT application. The students then perform an analysis of these tracks, focusing on identifying and correcting annotated chords and beats. Finally, they save their revised annotations and subsequently export the audio track with the annotation, to the public collection of the MusiCoLab

repository. This exercise not only challenges the students but also allows Dimitri to use his expertise in reviewing their submissions. He discusses their modifications in subsequent collaborative lessons, offering guidance on their mistakes. For his beginner students, he simplifies the task by selecting the track and marking incorrect analyzed chords with an 'X' (unknown chord), guiding them to focus on specific areas for improvement.

3.4.3 Synchronous Group Improvisation Session on a Jazz Piece

This scenario focuses on a synchronous group improvisation session, led by a jazz music teacher, utilizing the teleconferencing and collaborative features of the PAT application. It demonstrates the effective orchestration of a live, interactive music lesson, integrating preparation, real-time improvisation, and the creation of a final mixed recording.

Antonis, a jazz music teacher, plans to conduct a group improvisation session with his students. In preparation for the improvisation session, he carefully selects an appropriate backing track to upload to the PAT application, or alternatively, he records a new track and uses it as the session's backing track. He then uses the 'Analyze' button to obtain the content analysis of chords and beats for the chosen track. Once analyzed, he saves and exports this track to the course files of the MusiCoLab repository. He also provides a direct link to the application with the pre-selected track on the Moodle-based Learning Management System of the MusiCoLab platform, which they use weekly for their online lessons.

When the lesson time arrives, Antonis and his students connect via PAT's integrated teleconferencing feature, Jitsi Meet, to collaborate on the backing track. The annotated track, already analyzed by Antonis, is automatically loaded, streamlining the session's start. The students proceed to record their improvisations, aided by the metronome, which also serves as a pre-count. After completing their recordings, they wrap up the session and participants have the option to individually combine these improvisations into a final mix.



4 Evaluation

This chapter presents the evaluation of the PAT application using objectives and subjective measures. The first section presents a computational evaluation of the beat and chord analysis algorithms as well as the accuracy of alignment of the detected beats and chords. The section that follows provides a user experience evaluation gained through a three user experiments throughout synchronous music lessons. Finally, the third section consolidates the findings of this evaluation with respect to its capability to support the requirements of music teaching and learning.

4.1 Evaluation of algorithmic output

The evaluation of the algorithmic output used in this thesis application comprises three parts:

1. assessment of the beat recognition model,
2. analysis of the selected chord recognition models, and
3. investigation of the best chord alignment at beats with varying alignment percentages.

The source code for automating this evaluation is available on GitHub[1]. Library dependencies for the evaluation script are summarized in TABLE 5.1 of APPENDIX A.

This evaluation measures the performance of algorithms from open-source libraries and serves as a practical exercise in standard evaluation methods. It should be noted that results may have a bias due to the fact that the evaluation used some tracks from the dataset that was used for training the selected algorithms. Several evaluation measures have been implemented in programming toolkits to automate the task of evaluating on large datasets.

mir_eval[2] and *madmom*[3] or task-specific evaluators like the *Beat-Tracking-Evaluation-Toolbox*[4] and *MuOOEvaluator*[5] for the chord evaluation are examples of such toolkits. In the following, the evaluation functions provided by the *madmom* library have been used and

[1] Content Analysis Implementation & Evaluation: https://github.com/NeoklisMimidis/chord_and_beat_analysis

[2] https://craffel.github.io/mir_eval/

[3] https://madmom.readthedocs.io/en/v0.16.1/modules/evaluation.html#submodules

[4] https://github.com/adamstark/Beat-Tracking-Evaluation-Toolbox

[5] https://github.com/jpauwels/MusOOEvaluator

extended to accommodate chord evaluations for the chords produced by the CREMA package, which uses several chord type variations not supported by the original *madmom* library.

4.1.1 Beats

The evaluation of beat tracking algorithms includes subjective listening tests and objective, quantitative methods. The subjective approach involves human assessments of synthesized beats mixed with the original audio track (similar to the 'Click Track' feature of the PAT application), but it is time-consuming and lacks consistency. The objective evaluation uses a range of metrics, to quantitatively assess beat tracking accuracy, a practice also used in the MIREX Audio Beat Tracking (AB) community (Davies et al., 2009). This subsection describes the datasets and metrics typically employed in such evaluations and applies them to the assessment of the beat recognition model used in this thesis.

For more in-depth insights and intuitive understandings, readers are directed to resources like the MIREX Audio Beat Tracking website[1] and the interactive notebook online tutorials with integrated code examples, such as the 'Tempo, Beat, and Downbeat Estimation'[2] and 'Chapter 6: Tempo and Beat Tracking'[3].

A common visualization of the estimated beats in an audio file, involves overlaying vertical lines on the waveform display, as seen in Fig. 4.1.

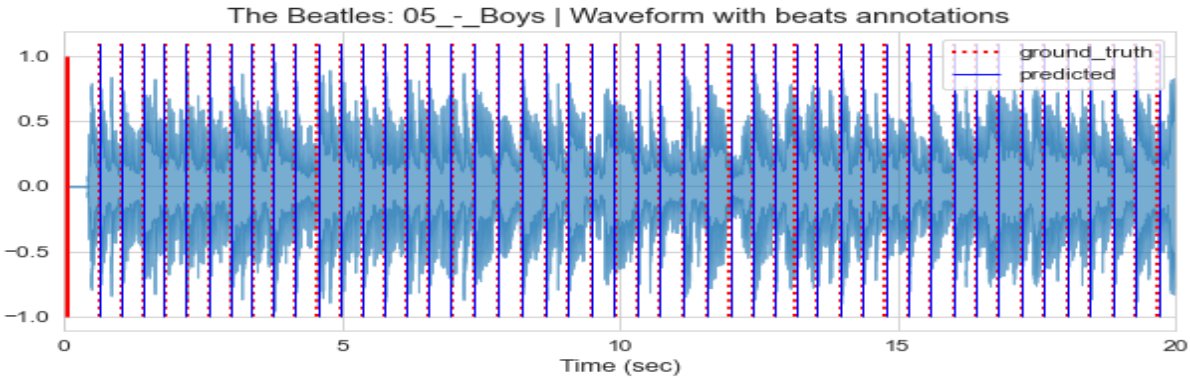


Fig. 4.1 Waveform with ground truth (dotted lines) and predicted (solid lines) beat annotations

[1] https://www.music-ir.org/mirex/wiki/2021:Audio_Beat_Tracking

[2] https://tempobeatdownbeat.github.io/tutorial/ch2_basics/evaluate.html

[3] https://www.audiolabs-erlangen.de/resources/MIR/FMP/C6/C6.html

DATASET

The datasets commonly used in for the beat evaluation in the MIREX AB contest, include the SMC[1], MAZ and MCK datasets. Additional datasets for the task of beat tracking can be found in the *mir_data*[2] Python library.

METRICS & RESULTS

The beat tracking model was evaluated using a dataset of 179 audio files from the 12 Beatles albums included in the Isophonics dataset[3]. This dataset was selected for its use in this thesis's chord recognition evaluation analysis, and the audio files were already available at hand. The results of this evaluation are presented in TABLE 4.1, and are interpreted in the subsequent paragraphs.

TABLE 4.1 Descriptive statistics for performance metrics in the quantitative assessment of beat tracking

	fmeasure	pscore	cemgil	goto	cmlc	cmlt	amlc	amlt	information gain	global information gain
count	179	179	179	179	179	179	179	179	179	179
mean	**0.871**	**0.896**	**0.758**	**0.816**	**0.743**	**0.850**	**0.794**	**0.910**	**3.154**	**1.617**
std	0.262	0.213	0.263	0.389	0.354	0.303	0.303	0.201	0.600	-
min	0.000	0.000	0.003	0.000	0.000	0.000	0.002	0.003	0.485	-
25%	0.923	0.944	0.652	1.000	0.489	0.932	0.615	0.937	2.868	-
50%	0.986	0.984	0.873	1.000	0.972	0.979	0.979	0.983	3.307	-
75%	0.996	0.994	0.944	1.000	0.994	0.994	0.995	0.995	3.541	-
max	1.000	1.000	0.979	1.000	1.000	1.000	1.000	1.000	4.151	-

A. F-measure

The F-measure in beat tracking calculates accuracy using three key elements: true positives (correct beats within a ±70ms window), false negatives (missed detections), and false positives (extra detections). It calculates and then combines, *precision* (proportion of the identified beats

[1] https://joserzapata.github.io/project/smc-beat-tracker-dataset/

[2] https://mirdata.readthedocs.io/en/stable/source/quick_reference.html

[3] http://www.isophonics.net/content/reference-annotations-beatles

which are correct) and *recall* (proportion of the actual beats which are correct), for the final accuracy score. The measure penalizes beats tapped off-beat, relative to the annotations or at incorrect metrical levels, by assigning a score of zero.

In our case the mean of **0.871**, suggests good overall accuracy, with most beats correctly identified within the window of ±70ms.

B. PScore

The PScore evaluates beat tracking by comparing the detected beats against ground truth annotations, employing cross-correlation as the similarity measure. It involves quantizing beats and annotations into impulse trains, and then calculating the cross-correlation within a specified window.

In our case the mean of **0.896**, indicates strong alignment with ground truth annotations.

C. Cemgil

The Cemgil method measures beat accuracy with a Gaussian error function, scoring beats on how close they are to the correct annotations. It uses a 40ms standard deviation tolerance window, to assess this proximity.

In our case the mean of **0.758**, which is lower compared to F-measure, indicates that the beat estimates are slightly poorly localized.

D. Goto

The Goto method categorizes beat tracking as either ‘correct’ or ‘incorrect’ based on the statistical analysis of beat errors. It determines this by comparing the time differences between each annotation and the closest beat. The method involves normalizing the error and evaluating a sequence under specific conditions related to timing and error statistics

In our case the mean of **0.816** implies a balanced rate of correct/incorrect categorizations.

E. Continuity-based

Continuity-based evaluations in beat tracking adjust the tolerance window in relation to the beat tempo: faster tempos have smaller windows; slower tempos have larger ones. These methods focus on how continuously beats are tracked in relation to the annotated beats.

- CMLc (Correct Metrical Level, continuity required): Measures the longest continuously correct segment at the correct metrical level.
- CMLt (Correct Metrical Level, continuity not required): Sums all continuously correct segments at the correct metrical level.
- AMLc (Allowed Metrical Levels, continuity required): Similar to CMLc but includes *allowed** metrical levels.
- AMLt (Allowed Metrical Levels, continuity not required): Similar to CMLt but includes *allowed** metrical levels.

*: *allowed* metrical levels indicates that annotation beats can be resampled at double or half the metrical level and off-beat tapping.

The minimal performance increase observed in AMLc (**0.794**) and AMLt (**0.910**) compared to CMLc (**0.743**) and CMLt (**0.850**) respectively (+ ~0.05), suggests that the beat tracking system identifies correctly the metrical levels in most instances. It is important to note that these observations are specific to the dataset under evaluation, and performance may vary in other contexts or with different musical compositions.

Moreover, the differences between metrics where continuity is required or not, may indicate instances of momentarily odd mis-placed beats due to variations in tempo or the complexity of musical passages.

F. Information Gain/Global Information Gain

Information Gain in beat tracking assesses a beat tracker's accuracy by comparing its predictions to a worst-case scenario (uniform distribution of errors). It uses the Kullback-Leibler divergence, measured in bits, to quantify the difference between the tracker's error distribution and a theoretical uniform distribution. Global Information Gain expands upon this by combining data from various files to evaluate the tracker's overall accuracy in a test database.

The average of **3.154** bits for Information Gain indicates a significant deviation from the worst-case scenario, showing successful beat prediction. The Global Information Gain of **1.617**, which is lower than the individual Information Gain average, indicates a varied performance when considering all files together. This could highlight inconsistencies in performance across different musical pieces or styles in the database (Davies et al., 2009, 2021)

4.1.2 Chords

Evaluating the performance of a chord recognition algorithm necessitates comparison with a ground truth, usually established by human annotators resulting in the creation of reference datasets. This subsection introduces the datasets and metrics commonly used in such evaluations, and then presents the results for the chord recognition models of this thesis.

For a detailed analysis and hands-on examples, readers are encouraged to explore the interactive FMP notebook 'Chapter 5: Chord Recognition'[1] and the source code[2] of the chord recognition model that was primally selected in this thesis and previously described, CREMA.

DATASET

The accuracy of chord analysis is typically measured by comparing the algorithmic output with a ground truth crafted by one or more human annotators. Datasets featuring chord annotations in the MIREX ACE[3] contest include the Isophonics[4], McGill Billboard[5], USPOP2002Chords - subset of the USPOP2002[6] with only chord annotations, the RWC-Popular[7] and artist-specific datasets such as those for Robbie Williams[8] and Jay Chou. One more noteworthy addition, is the recent Jazz Audio-Aligned Harmony (JAAH)[9] dataset which focuses on the jazz genre, and expands the existing datasets of the task.

[1] https://www.audiolabs-erlangen.de/resources/MIR/FMP/C5/C5.html

[2] https://github.com/bmcfee/ismir2017_chords/tree/master

[3] https://www.music-ir.org/mirex/wiki/2021:Audio_Chord_Estimation

[4] http://www.isophonics.net/datasets

[5] https://ddmal.music.mcgill.ca/research/The_McGill_Billboard_Project_(Chord_Analysis_Dataset)/

[6] https://labrosa.ee.columbia.edu/projects/musicsim/uspop2002.html

[7] https://github.com/tmc323/Chord-Annotations

[8] https://www.researchgate.net/publication/260399240_Chord_and_Harmony_annotations_of_the_first_five_albums_by_Robbie_Williams

[9] https://github.com/MTG/JAAH

METRICS

A. Chord Symbol Recall

The standard quality measure to evaluate the quality of an automatic transcription is Chord Symbol Recall (CSR). Initially, MIREX utilized uniform length chord symbol frames, typically 10ms long, for sampling chord sequences. This approach changed into a segment-based evaluation, which treats both the ground-truth and predicted annotations as continuous audio segmentations and calculates the cumulative length of correctly overlapping segments (summed duration of time periods where the correct chord has been identified, normalized by the total duration of the song), offering a more precise and computationally more efficient CSR method. Finally, in cases of evaluating multiple files, the overall duration of each song is considered, rather than just averaging all files. This approach weights each song by its length, treating the collection as a single extended audio example. The resulting metric is referred to as Weighted Chord Symbol Recall (WCSR).

B. Vocabularies

The accuracy of the correctly overlapping segments is assessed based on how well the chord qualities match the chords contained in specific chord vocabularies. In MIREX, there are five different chord vocabularies used, including MajMin and Sevenths, highlighted in grey in TABLE 4.2. For instance, the 'Dominant Seventh' chord (shown in row 13 of the table) is evaluated as a major chord in the MajMin vocabulary. This is because when a specific chord quality does not have a direct metric, it's evaluated under a simpler quality. On the other hand, the same chord is directly evaluated in the Sevenths vocabulary. Additionally, MIREX includes the MajMinBass and SeventhsBass vocabularies as part of their evaluation scheme (extending the MajMin and Sevenths vocabularies with chord inversions), and the Roots vocabulary, which focuses only on chord roots.

TABLE 4.2 Chord qualities and their evaluation in chord vocabularies

	Description	Chord Quality (Shorthand) / Intervals List	Vocabularies			
			MajMin	Triads	Sevenths	Tetrads
1	No chord	*N*	✅	✅	✅	✅
2	Unknown chord	*X*	✅	✅	✅	✅

3	Major	*maj* (1,3,5)	✅	✅	✅	✅
4	Minor	*min* (1, b3, 5)	✅	✅	✅	✅
5	Augmented	*aug* (1, 3, #5)	🚫	✅	🚫	✅
6	Diminished	*dim* (1, b3, b5)	🚫	✅	🚫	✅
7	Suspended Second	*sus2* (1, 2, 5)	🚫	✅	🚫	✅
8	Suspended Fourth	*sus4* (1, 4, 5)	🚫	✅	🚫	✅
9	Major Sixth	*maj6* (1, 3, 5, 6)	☑ (maj)	☑ (maj)	🚫	✅
10	Minor Sixth	*min6* (1, b3, 5, 6)	☑ (min)	☑ (min)	🚫	✅
11	Major Seventh	*maj7* (1, 3, 5, 7)	☑ (maj)	☑ (maj)	✅	✅
12	Minor Seventh	*min7* (1, b3, 5, b7)	☑ (min)	☑ (min)	✅	✅
13	Dominant Seventh	7 (1, 3, 5, b7)	☑ (maj)	☑ (maj)	✅	✅
14	Diminished Seventh	*dim7* (1, b3, b5, bb7)	🚫	☑ (dim)	🚫	✅
15	Half-Diminished Seventh	*hdim7* (1, b3, b5, b7)	🚫	☑ (dim)	🚫	✅
16	Minor Major Seventh	*minmaj7* (1, b3, 5, 7)	☑ (min)	☑ (min)	🚫	✅
17	Only Root*	*1* (1)	🚫	✅	🚫	✅
18	Power Chord*	*5* (1, 5)	🚫	✅	🚫	✅

* *Root* and *Power Chord* are not classified as 'true' chords in the literature

This table outlines chord qualities with their shorthand notations and interval structures. The 'Vocabularies' columns (MajMin, Triads, Sevenths, and Tetrads) use symbols to specify the evaluation procedure of the respective shorthand:

- ✅ direct evaluation;
- ☑ mapped to subsets (if root and bass note match);
- 🚫 excluded from the evaluation;

C. Segmentation

The Chord Symbol Recall metric effectively measures an algorithm's accuracy in identifying the correct chord at any given moment. However, it doesn't reflect the algorithm's consistency. This can be illustrated with the annotated sequence and the two predicted sequences in TABLE 4.3. You can notice that both predictions have a recall value of 0.66, with 8 out of 12 beats matching correctly.

However, Prediction 1 is musically more useful, because it is less fragmented compared to Prediction 2, and identifies correctly the chord changes. A musician could then listen and correct the wrong chords in the second bar of that prediction and then use it as a lead sheet, but the same doesn't apply for the Prediction 2 where he will have to put more effort.

TABLE 4.3 Segmentation value equals to the lowest between under- and over-segmentation

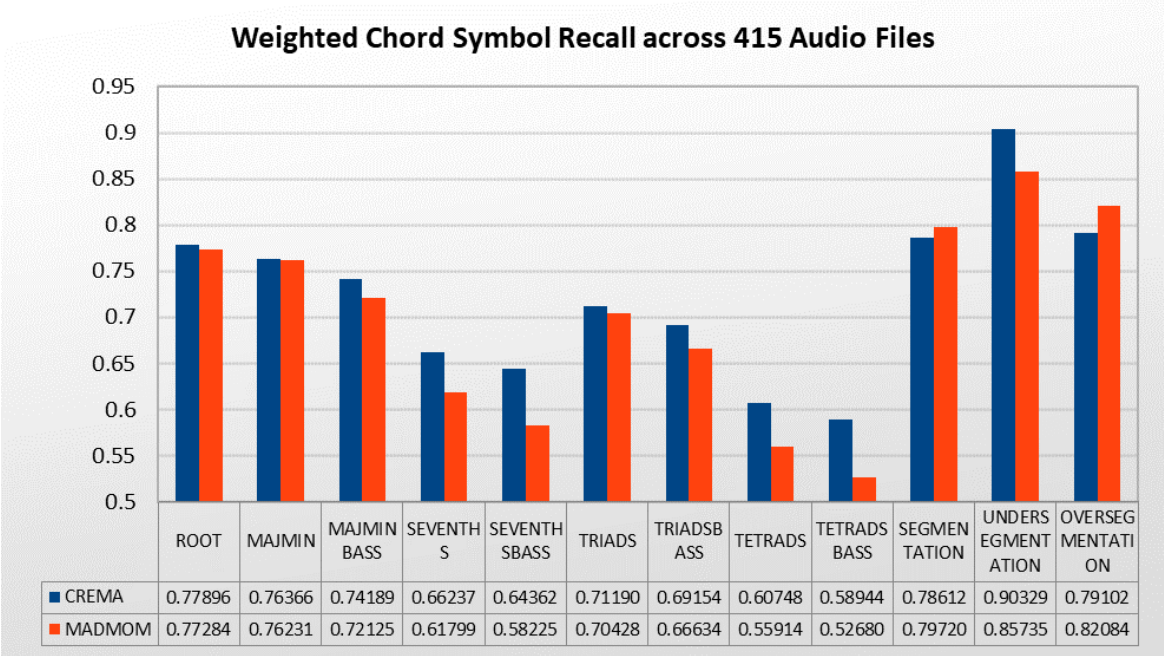
Ground Truth	C				A		F		G			
Prediction 1	C				F		A		G			
Prediction 2	C	A	C		F	A	F		F	G		C
Bar:Beat	1:1	1:2	1:3	1:4	2:1	2:2	2:3	2:4	3:1	3:2	3:3	3:4

To address this issue, the evaluation employs the directional Hamming distance, which finds for each annotated segment, the overlap in the other annotation and then totals their differences. These metrics highlight whether there is over or under-segmentation, and when used in both ways, it offers a thorough assessment of the overall quality. (Fard, 2020; Harte, 2010). In our previous example Prediction 1 will score a perfect segmentation, while Prediction 2 is clearly over-segmented.

RESULTS

Chord recognition systems, are usually evaluated with methods that employ cross-validation on split train-test datasets. This thesis differs in that aspect (because it uses existing implementations) and employs a simple comparative analysis of the CREMA and MADMOM chord recognition algorithms, on a compound dataset of several well-known dataset collections. The compound dataset includes 415 audio files, with selected works from The Beatles (180), Carole King (7), and Queen (20) within the Isophonics dataset; a substantial portion of the USPOP2002Chords dataset (179); and the Jay Chou (29) collection.

The Weighted Chord Symbol Recall (WCSR) across multiple chord vocabularies and segmentation metrics, of the two models are visualized on the bar graph of Fig. 4.2.



	ROOT	MAJMIN	MAJMIN BASS	SEVENTHS	SEVENTH SBASS	TRIADS	TRIADSBASS	TETRADS	TETRADS BASS	SEGMENTATION	UNDERSEGMENTATION	OVERSEGMENTATION
CREMA	0.77896	0.76366	0.74189	0.66237	0.64362	0.71190	0.69154	0.60748	0.58944	0.78612	0.90329	0.79102
MADMOM	0.77284	0.76231	0.72125	0.61799	0.58225	0.70428	0.66634	0.55914	0.52680	0.79720	0.85735	0.82084

Fig. 4.2 WCSR between MADMOM and CREMA chord recognition models

From the graph it can be seen that the performance between the two models in the Root, MajMin, and Triads vocabularies was close, with CREMA slightly outperforming MADMOM. This result contradicts earlier selection experiments, that involved around a smaller dataset of 28 Beatles songs, and MADMOM[1] was outperforming CREMA.

However, as the chord complexity increased within the vocabularies, CREMA demonstrated a significant lead, particularly within the TetradsBass[2] vocabulary, where it surpassed MADMOM by over 6% (this was expected because CREMA was trained in complex chords and supports chord inversions). The support of complex chords, aligned well with the needs of the MusiCoLab platform's user base, were most musicians had a jazz background.

Finally, in the segmentation metrics, MADMOM leaned towards over-segmentation, while CREMA tended towards under-segmentation, resulting in a lower overall segmentation metric, a non-issue as chords were later aligned to beats in subsequent processing.

[1] This was also one of the reasons that the MADMOM chord recognition model was integrated into the server-side content analysis script

[2] The TetradsBass extends the Tetrads vocabulary in TABLE 4.2, with the addition of chord inversions.

4.1.3 Chords at Beats

The final output of the content analysis - as described in 3.2.5, was the chords aligned at beat times.

This alignment process used beats from the beat recognition algorithm evaluated in 4.1.1, and chords from the CREMA chord recognition model[1] evaluated in 4.1.2. The alignment process underwent extensive testing with various values, and its performance was assessed using the ground truth annotations of the chords. After trial and error, the optimal performance was achieved with an alignment value of 0.41 percent[2].

However, it was observed that the combination of the chords and beats did not result in a performance enhancement, even at the best alignment percentage. This outcome was further verified in Fig. 4.3.

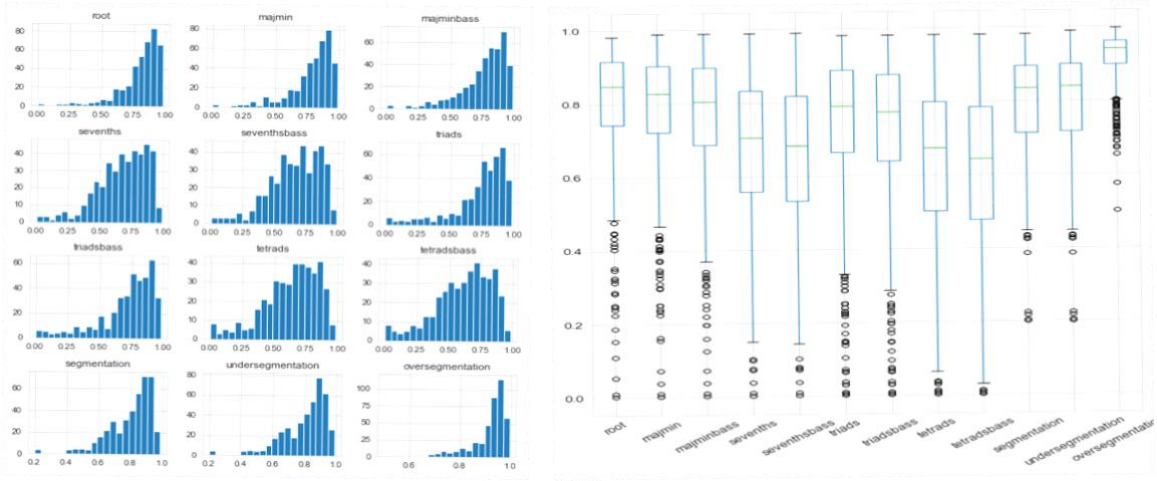
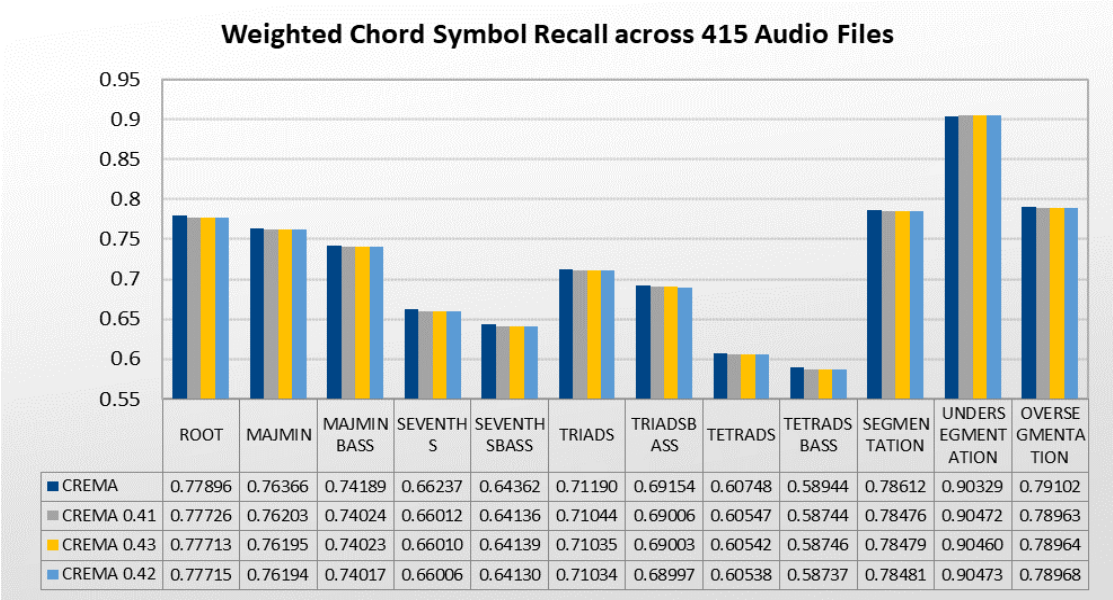


Fig. 4.3 Left: Histograms showing the distribution of WCSR scores for the best-performing alignment (CREMA 0.41) across various evaluation metrics for a dataset of 415 audio files. Right: Corresponding boxplots illustrating the median, interquartile range, and outliers, offering a statistical summary for each metric

The best aligned model performed similarly to the only chords CREMA model. Specifically, in the complex categories such as 'sevenths' and 'tetrads', the model's recall scores were having a higher level of difficulty (bigger variability), while in simpler categories like 'root' and 'majmin,' the model showed a more reliable chord recognition capability.

[1] CREMA was preferred over MADMOM for its diverse chord vocabulary

[2] Similar tests with the MADMOM model demonstrated a significantly higher alignment value, at 0.9 percent



	ROOT	MAJMIN	MAJMIN BASS	SEVENTHS	SEVENTHSBASS	TRIADS	TRIADSBASS	TETRADS	TETRADSBASS	SEGMENTATION	UNDERSEGMENTATION	OVERSEGMENTATION
CREMA	0.77896	0.76366	0.74189	0.66237	0.64362	0.71190	0.69154	0.60748	0.58944	0.78612	0.90329	0.79102
CREMA 0.41	0.77726	0.76203	0.74024	0.66012	0.64136	0.71044	0.69006	0.60547	0.58744	0.78476	0.90472	0.78963
CREMA 0.43	0.77713	0.76195	0.74023	0.66010	0.64139	0.71035	0.69003	0.60542	0.58746	0.78479	0.90460	0.78964
CREMA 0.42	0.77715	0.76194	0.74017	0.66006	0.64130	0.71034	0.68997	0.60538	0.58737	0.78481	0.90473	0.78968

Fig. 4.4 Bar chart comparing the Weighted Chord Symbol Recall (WCSR) between the CREMA model and the top three beat alignment derivatives (CREMA 0.41||0.43||0.42)

Similar observations were made from the Bar chart in Fig. 4.4 and the Histogram of Differences in Fig. 4.5, which illustrated only a marginal disparity between the two approaches.

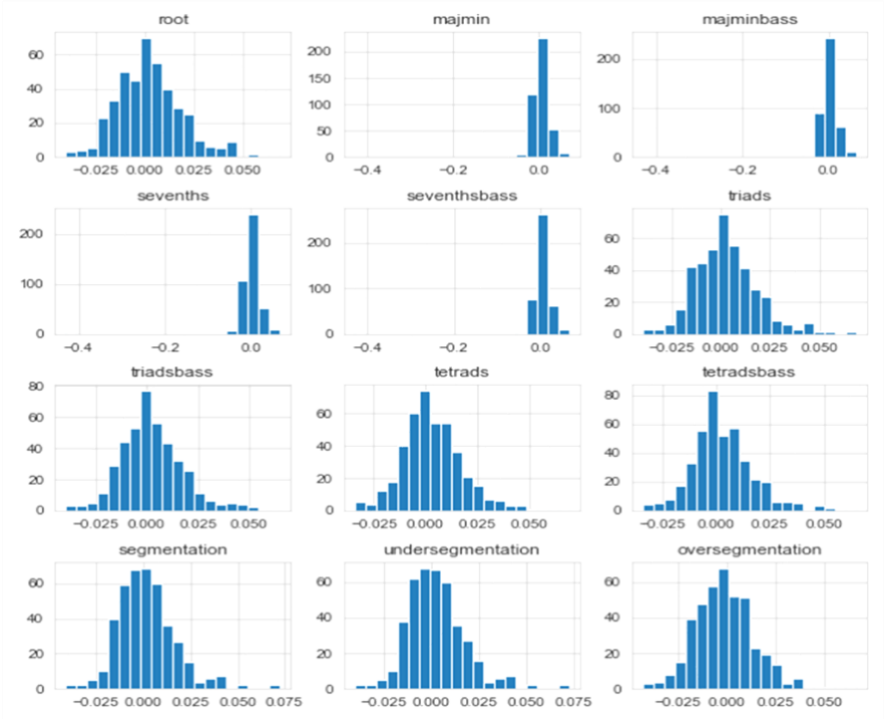


Fig. 4.5 Histogram of differences between the CREMA model and the top alignment

4.2 Evaluation of usability

In the context of the MusiCoLab research project, the PAT application was evaluated in three pilot experiments that involved online music lessons. Valuable feedback was acquired by collecting answers through a post-experiment questionnaire. In addition to the questionnaire, screen recordings allowed observing the collaboration of users throughout the experiments and identifying potential problems that could be later resolved. This section presents a brief description of each experiment and the answers obtained through the questionnaire.

Experiment 1 focused on jazz improvisation, involving five participants, including an observer, two instructors and two students (acoustic guitar and drums/accordion) (see Fig. 4.6). The findings of this experiment contributed to the presentation of the MusiCoLab research project at the *4th International Symposium on the Internet of Sounds*[1].

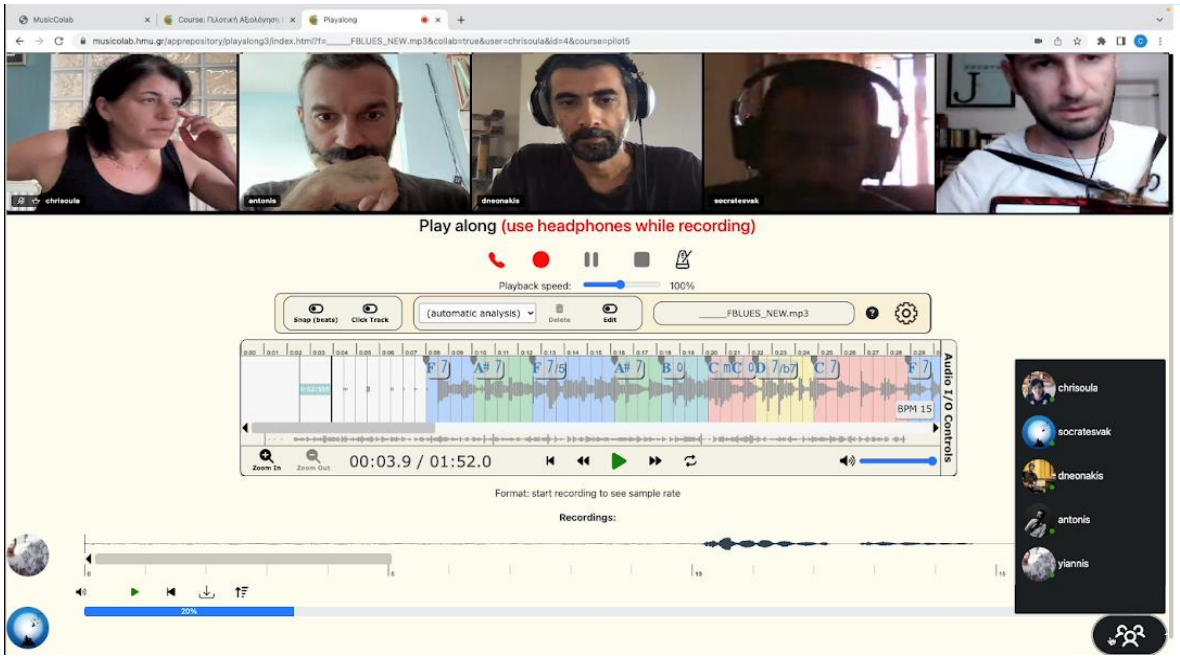


Fig. 4.6 Screenshot from the screen recording of experiment 1

After the experiment, participants were asked to answer a questionnaire evaluating the content analysis and editing functionalities of the PAT application. These features were assessed for their effectiveness and importance in online music education, with positive results, as shown in Fig. 4.7. It is important to note that these features are not typical in online music lessons and were unfamiliar to the participants (Alexandraki et al., 2023).

[1] https://internetofsounds.net/is2_2023/

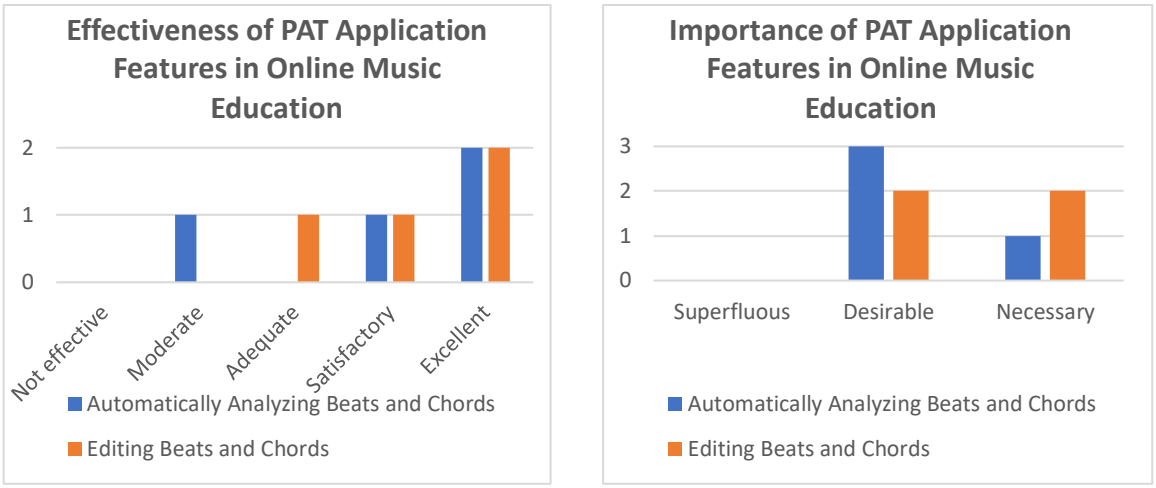


Fig. 4.7 PAT application effectiveness and importance of content analysis and curation

Experiment 2, focused on blues improvisation, also involved five participants with a similar format to experiment 1. The two students participated with their singing/voice and keyboard performances (see Fig. 4.8). Through this experiment, missing functionalities and potential features that could be added as future extensions of the application were suggested.



Fig. 4.8 Screenshot from the screen recording of the experiment 2

Experiment 3, was a guitar lesson between a teacher -member of the MusiCoLab research team, and a young student. This experiment was designed to include exercises for two guitars and to use as most of the application affordances for synchronous and asynchronous collaboration (see Fig. 4.9). Through the experiment, no significant issues were encountered, and the young

student was particularly enthusiastic when using the available tools, which proved to greatly motivate the engagement of the student.



Fig. 4.9 Screenshot from the screen recording of the experiment 3

As opposed to experiment 1, experiments 2 and 3 used the same questionnaire which investigates three aspects: a) whether the facility of music content analysis supports the learning process b) whether tooltips are intuitive and helpful, and c) whether application enhancements such as tonality detection and structural analysis would be considered important for the music lesson.

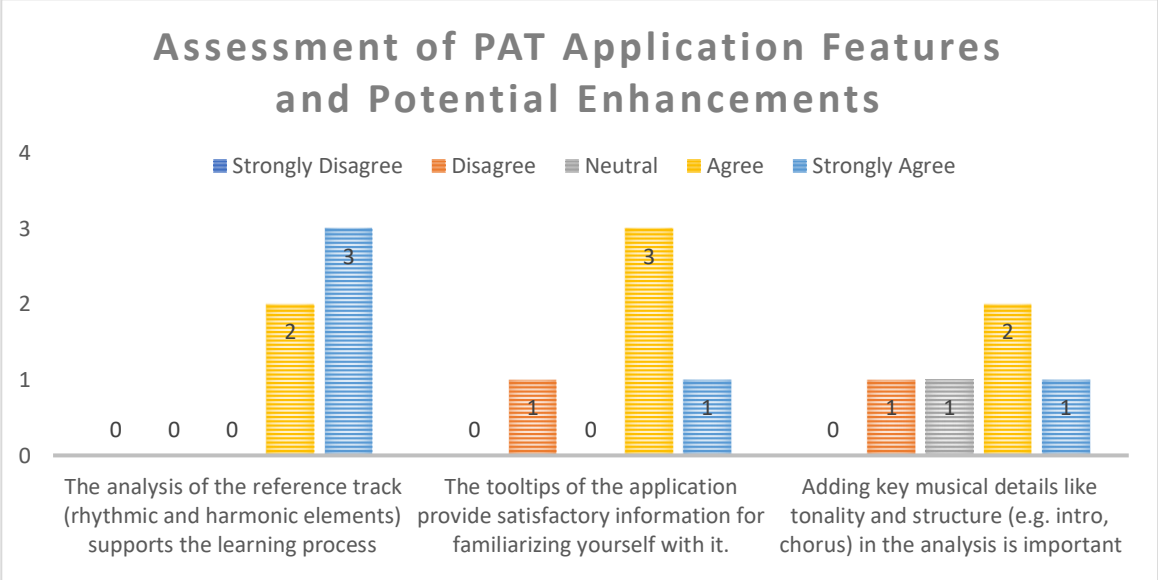


Fig. 4.10 PAT application features assessment and future enhancements

As depicted on Fig. 4.10 users were very positive concerning the importance of music content analysis for music education. In majority, tooltips were found to be intuitive. Finally, concerning the retrieval of further information from the backing track, the responses varied but were predominantly positive, indicating musicians' willingness to adopt intelligent music content analysis tools in their lessons.

4.3 Consolidation

The evaluation of the algorithmic performance revealed that the algorithms used in the application for beat tracking and chord estimation, as well as the combination of the two, achieves satisfying performance, according to the benchmark metrics used by the MIREX community. In terms of user experience, the PAT application was rated positively highlighting that intelligent, novel music analysis tools (i.e., automatic analysis of rhythmic and harmonic content) although not familiar to users, enriches user experience and user engagement in online music education.

According to Maharaj & Gill (2023) the current availability of technological tools for music education allows students to develop specific skills by adopting self-paced learning practices that are supplementary to classroom learning. These tools not only allow individuals learners to progress at their own pace, but also enable educators to focus on developing other essential skills in their students. Furthermore, while the use of AI-enhanced educational applications is currently limited, the development of digital assistants that create music exercises and offer instant feedback, such as aural skills training apps, is a promising direction for future work.

5 Conclusions

5.1 Summary

This thesis focuses on the capabilities of state-of-the-art music content analysis tools to assist and support online music learning. Specifically, a web application has been developed for the study of music recordings. The application makes use of current achievements in the research area of MIR to automatically analyze the rhythmic and harmonic content of a music recording. The result of the analysis is presented in an intuitive visualization and provides various tools for editing the automatically derived content, not only to correct errors of the automatic analysis process but also to aid the participants of a music lesson to better understand the content of the music piece being studied.

The developed application is referred to as Play-Along-Together (PAT) and is part of the MusiCoLab research and development project for online music education. The development of PAT required in-depth study of the libraries and the programming tools currently available for automatic music content retrieval from musical sounds. Specifically, the tools that were studied focus on rhythm-tempo estimation and beat tracking as well as on chord estimation. In combination, these tools allow for providing a comprehensive semantic representation of music signals.

Furthermore, the development of the application required the investigation of current technologies for developing web applications. Such technologies allow capturing, displaying and processing audio waveforms in a web browser environment, refraining from the requirement of application installation and from problems related to platform compatibility. This provides music teachers and their students with improved accessibility to the application features, but also permits the implementation of collaborative interactions through a dedicated Graphical User Interface. Additionally, the PAT application incorporates several usability aids such as description tooltips, context-aware toolbars, context-sensitive and action-triggered icon buttons, visual skip cues, and keyboard shortcuts. These elements contribute significantly to the intuitive nature of the application, enabling users to navigate and use the app more effectively.

Moreover, this thesis provides an evaluation of the PAT application involving both the computational performance of the MIR algorithms as well as an assessment of user experience through a series of pilot experiments. This evaluation demonstrates the efficiency of the PAT

application and the capabilities of recent research achievements in intelligent computing, i.e., achievements of MIR research, to enhance user engagement and the user experience in music learning and teaching.

5.2 Contributions and Outcomes

Briefly, the work carried out in this thesis has produced outcomes in the following directions:

- The development of a collaborative web-based application, named PAT, which is aimed at enriching music education experience through facilitating both synchronous and asynchronous music learning and teaching scenarios.
- The integration of intelligent, novel music analysis tools to enhance teaching and learning, demonstrating practical applications and benefits.
- My contribution and active participation in the MusiCoLab research project[1], due to which I was able to co-present my developments at the *4th International Symposium on the Internet of Sounds*[2].
- The release of open-source software which includes server-side processing tools for music analysis[3] and the web application[4].

5.3 Future Perspectives

The web application developed in the context of the present thesis can be used to automatically analyze, visualize and curate musical content, using intelligent tools aimed at the recognition of beats and chords. However, the MIR tools available today offer numerous additional capabilities, e.g., structural segmentation, audio transcriptions, melody recognition and more, which may be useful for musicians and musicologists alike. One of the reasons that these tools are not widely used for music education is because their integration into meaningful applications has been delayed. At present, a limited number of applications that make use of intelligent music analysis are being developed, most of which are offered as proprietary software and are not usually tailored to the requirements of music education.

[1] https://musicolab.hmu.gr/musicolab/project_en.php

[2] https://internetofsounds.net/is2_2023/

[3] https://github.com/NeoklisMimidis/chord_and_beat_analysis

[4] https://github.com/NeoklisMimidis/playalong3 & https://musicolab.hmu.gr/apprepository/playalong3/

On the other hand, numerous open-source programming libraries are currently available to be used for application development. Such programming libraries offer the possibility to improve the current effort in future plans by incorporating relationships between chords and related musical concepts such as beats, key, genre, and melody, improving the final accuracy and providing multiple tools of analysis to relevant applications. For instance, with the information of the musical key, the application could appropriately select the correct chords instead of their enharmonic equivalent. In the case of the utilization of the downbeat positions, the application could showcase the current playback cursor in a format based on bars and beats. Finally, providing users with information on different song sections, such as verses, choruses, and bridges, the application could significantly improve track navigation, making it easier for them to move through different parts of the song.

Building upon the implemented features, the application could enhance user interaction by introducing new controls to overcome current constraints. For instance, enabling users to configure beat and chord recognition settings (via a modal window) before analysis, such as selecting a piece's metrical structure or tempo, could significantly enhance the accuracy of the underlying algorithm. Moreover, expanding the chord editor to include additional musical notations like double sharps, double flats, and chord inversions could give users more options for curating the underlying musical content. Furthermore, future updates could introduce various chord visualization tools, such as fretboard diagrams for string instruments (e.g., guitar, ukulele, mandolin) or piano keyboard displays, which aim to display the musical content in a way that aligns more naturally with the users' experiences of playing and understanding music.

As this thesis explores the use of music technology in music education, it proposes the use of MIR research advancements in daily musical practice by seamlessly integrating intelligent tools as a helpful addition to conventional teaching methods. Despite technical challenges, such as communication latencies in online music lessons, the ongoing progress in MIR and other research areas, e.g., Network Music Performance and the Internet of Sounds, promises improved online services not only for music consumers but also for musicians, musicologists, and the music education community at large.

References

Alexandraki, C., Mimidis, N., Viglis, Y., Nousias, A., Milios, D., & Tsioutas, K. (2023). Collaborative Playalong Practices in Online Music Lessons: The MusiCoLab Toolset. *2023 4th International Symposium on the Internet of Sounds*, 1–10. https://doi.org/10.1109/IEEECONF59510.2023.10335236

Bello, J. P., & Pickens, J. (2005). A Robust Mid-level Representation for Harmonic Content in Music Signals. *Proceedings of the 6th International Society for Music Information Retrieval (ISMIR)*, 304–311.

Böck, S., & Davies, M. E. P. (2020). Deconstruct, Analyse, Reconstruct: How to Improve Tempo, Beat, and Downbeat Estimation. *Proceedings of the 21st International Society for Music Information Retrieval (ISMIR)*.

Böck, S., Korzeniowski, F., Schlüter, J., Krebs, F., & Widmer, G. (2016). madmom: A new Python Audio and Music Signal Processing Library (arXiv:1605.07008). arXiv. http://arxiv.org/abs/1605.07008

Böck, S., Krebs, F., & Widmer, G. (2016). Joint Beat and Downbeat Tracking with Recurrent Neural Networks. *Proceedings of the 17th International Society for Music Information Retrieval (ISMIR)*.

Bogdanov, D., Wack, N., Gómez, E., Gulati, S., Herrera, P., Mayor, O., Roma, G., Salamon, J., Zapata, J., & Serra, X. (2013). ESSENTIA: An Audio Analysis Library for Music Information Retrieval. *Proceedings of the 14th International Society for Music Information Retrieval (ISMIR).*

Brown, J. C. (1991). Calculation of a Constant *Q* Spectral Transform. *The Journal of the Acoustical Society of America*, *89*(1), 425–434. https://doi.org/10.1121/1.400476

Carsault, T., & Nika, J. (2022). MERCI WP DD. Deep Discovery, Deliverable DD. 1 (V1) Report Embedding spaces and multivariate time [1].

Cho, T. (2014). *Improved Techniques for Automatic Chord Recognition from Music Audio Signals* [PhD Thesis].

Correya, A., Marcos-Fernández, J., Joglar-Ongay, L., Alonso-Jiménez, P., Serra, X., & Bogdanov, D. (2021). Audio and Music Analysis on the Web using Essentia.js *Transactions of the International Society for Music Information Retrieval (ISMIR)*

Davies, M., Böck, S., & Magdalena, M. (2021, November). Tempo, Beat and Downbeat Estimation. Retrieved from https://tempobeatdownbeat.github.io/tutorial/intro.html

Davies, M., Degara Quintela, N., & Plumbley, M. (2009). Evaluation Methods for Musical Audio Beat Tracking Algorithms.

Downie, J. S. (2008). The music information retrieval evaluation exchange (2005-2007): A window into music information retrieval research. *Acoustical Science and Technology*, *29*(4), 247–255. https://doi.org/10.1250/ast.29.247

Fard, H. H. (2020). *Automatic Chord Recognition with Fully Convolutional Neural Networks* [Master's Thesis].

Fujishima, T. (1999). Realtime Chord Recognition of Musical Sound: A System Using Common Lisp Music. *Proceedings of the 1999 International Computer Music Conference (ICMC)*. https://quod.lib.umich.edu/i/icmc/bbp2372.1999.446/1

Gómez, E. (2006). Tonal Description of Polyphonic Audio for Music Content Processing. *INFORMS Journal on Computing*, *18*(3), 294–304. https://doi.org/10.1287/ijoc.1040.0126

Harte, C. (2010). *Towards automatic extraction of harmony information from music signals* [PhD Thesis]. https://qmro.qmul.ac.uk/xmlui/handle/123456789/534

Harte, C., & Sandler, M. (2009). Automatic Chord Recognition Using Quantised Chroma And Harmonic Change Segmentation.

Humphrey, E. J., & Bello, J. P. (2012). Rethinking Automatic Chord Recognition with Convolutional Neural Networks. *Proceedings of the 11th International Conference on Machine Learning and Applications (ICMLA)*, 357–362. https://doi.org/10.1109/ICMLA.2012.220

Humphrey, E. J., Salamon, J., Nieto, O., Forsyth, J., Bittner, R. M., & Bello, J. P. (2014). JAMS: A JSON Annotated Music Specification for Reproducible MIR Research. *Proceedings of the 15th International Society for Music Information Retrieval (ISMIR)*, 591–596.

JAMS. (2021, January 29). JAMS 0.3.2 documentation. Retrieved from https://jams.readthedocs.io/en/stable/index.html

Koops, H. V., de Haas, W. B., Burgoyne, J. A., Bransen, J., Kent-Muller, A., & Volk, A. (2019). Annotator subjectivity in harmony annotations of popular music. *Journal of New Music Research*, *48*(3), 232–252. https://doi.org/10.1080/09298215.2019.1613436

Korzeniowski, F. (2018). *Harmonic Analysis of Musical Audio using Deep Neural Networks* [PhD Thesis]. Ph. D. Dissertation. Johannes Kepler University Linz, Linz, Austria.

Korzeniowski, F., & Widmer, G. (2016, December 15). Feature Learning for Chord Recognition: The Deep Chroma Extractor. *Proceedings of the 17th International Society for Music Information Retrieval (ISMIR)*. http://arxiv.org/abs/1612.05065

Korzeniowski, F., & Widmer, G. (2017, February 1). On the Futility of Learning Complex Frame-Level Language Models for Chord Recognition. arXiv.Org. https://doi.org/10.17743/aesconf.2017.978-1-942220-15-2

Maharaj, A., & Gill, A. (2023). Technology in Music Education. Canadian Journal of Learning and Technology, 49(2), 1-15.

Mauch, M., & Dixon, S. (2010). Approximate Note Transcription for the Improved Identification of Difficult Chords. Proceedings of the 11th International Society for Music Information Retrieval (ISMIR), 135–140.

McFee, B., & Bello, J. P. (2017). Structured Training For Large-Vocabulary Chord Recognition. *Proceedings of the 18th International Society for Music Information Retrieval (ISMIR)*.

McFee, B., Raffel, C., Liang, D., Ellis, D., McVicar, M., Battenberg, E., & Nieto, O. (2015). librosa: Audio and Music Signal Analysis in Python. *Proceedings of the 14th Python in Science Conference (SciPy '15)*, 18–24. https://doi.org/10.25080/Majora-7b98e3ed-003

McVicar, M., Santos-Rodriguez, R., Ni, Y., & Bie, T. D. (2014). Automatic Chord Estimation from Audio: A Review of the State of the Art. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *22*(2), 556–575. https://doi.org/10.1109/TASLP.2013.2294580

Mottaghi, A., Behdin, K., Esmaeili, A., Heydari, M., Marvasti, F., & Advanced Communications Research Institute (ACRI), Tehran, Iran. (2017). OBTAIN: Real-Time Beat Tracking in Audio Signals. *International Journal of Signal Processing Systems*, *5*(4), 123–129. https://doi.org/10.18178/ijsps.5.4.123-129

Müller, M. (2015). *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer International Publishing. https://doi.org/10.1007/978-3-319-21945-5

Müller, M., & Ewert, S. (2011). Chroma Toolbox: Matlab Implementations for Extracting Variants of Chroma-Based Audio Features. *Proceedings of the 12th International Society for Music Information Retrieval (ISMIR)*.

Müller, M., Kurth, F., & Roder, T. (2004). Towards an efficient algorithm for automatic score-to-audio synchronization. *Proceedings of the 5th International Society for Music Information Retrieval (ISMIR)*.

Nawab, S. H., Ayyash, S. A., & Wotiz, R. (2001). Identification of Musical Chords using CONSTANT-Q Spectra. *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, *5*, 3373–3376. https://doi.org/10.1109/ICASSP.2001.940382

Odekerken, D., Koops, H. V., & Volk, A. (2020). DECIBEL: Improving Audio Chord Estimation for Popular Music by Alignment and Integration of Crowd-Sourced Symbolic Representations (arXiv:2002.09748). arXiv. http://arxiv.org/abs/2002.09748

Ono, N., Miyamoto, K., Le Roux, J., Kameoka, H., & Sagayama, S. (2008). Separation of a monaural audio signal into harmonic/percussive components by complementary diffusion on spectrogram. *IEEE 2008 16th European Signal Processing Conference*.

Pauwels, J., O'Hanlon, K., Gómez, E., & Sandler, M. B. (2019). 20 Years of Automatic Chord Recognition from Audio. *Proceedings of the 20th International Society for Music Information Retrieval (ISMIR)*, 54–63.

Pinto, A., Böck, S., Cardoso, J., & Davies, M. (2021). User-Driven Fine-Tuning for Beat Tracking. *Electronics*, *10*(13), 1518. https://doi.org/10.3390/electronics10131518

Reed, J., Ueda, Y., Siniscalchi, M., Uchiyama, Y., Sagayama, S., & Lee, C.-H. (2009). Minimum Classification Error Training to Improve Isolated Chord Recognition. *Proceedings of the 10th International Society for Music Information Retrieval (ISMIR)*, 609–614.

Schörkhuber, C., & Klapuri, A. (2010). Constant-Q Transform Toolbox for Music Processing. *Proceedings of the 7th Sound and Music Computing (SMC)*, 3–64.

Sheh, A., & Ellis, D. P. W. (2003). Chord Segmentation and Recognition using EM-Trained Hidden Markov Models. *Proceedings of the 4th International Society for Music Information Retrieval (ISMIR)*.

Varewyck, M., Pauwels, J., & Martens, J. P. (2008). A Novel Chroma Representation of Polyphonic Music Based on Multiple Pitch Tracking Techniques. *Proceedings of the 2008 ACM International Conference on Multimedia*, 667–670. https://doi.org/10.1145/1459359.1459455

Κούτρας, Α., Koutras, A., Αλεξανδράκη, Χ., Alexandraki, C., Ζαρούχας, Θ., Zarouchas, T., Ζέρβας, Π., Zervas, P., Χατζηαντωνίου, Π., & Chatziantoniou, P. (2023). *Επεξεργασία και ανάλυση ήχου, ομιλίας και μουσικής* https://doi.org/10.57713/kallipos-347

Appendix A

Availability and software dependencies

This appendix provides details on the availability and software dependencies of the PAT application, server-side Python libraries and client-side JavaScript packages.

Availability

The Play-Along-Together (PAT) Web Application is accessible from the MusiCoLab platform[1] and its source code is openly available through GitHub[2]. Unregistered users can use most of the PAT application features, including the automatic music content analysis. However, the import and export functionalities of the application that allow users to transfer audio files from and to the MusiCoLab repository and the application usage in a collaborative scenario, are only available to registered users. To access all features, users need to first register on the MusiCoLab platform[3], which also grants them access to educational courses and the VHVCoLab application for synchronous collaboration and asynchronous interaction with musical score files.

Software dependencies

TABLE 5.1 Library dependencies for the scripts of music content analysis and evaluation

Library	Version
Content analysis script dependencies	
numpy	1.21.6
librosa	0.10.1
jams	0.3.4
crema	0.2.0
madmom*	0.16.1
Evaluation of annotation retrieval additional script dependencies	
pandas	1.3.5
matplotlib	3.5.3
seaborn	0.12.2
**Python 3.7 is required*	

[1] https://musicolab.hmu.gr/apprepository/playalong3/

[2] https://github.com/NeoklisMimidis/playalong3

[3] https://musicolab.hmu.gr/lms/login/index.php

TABLE 5.2 Packages dependencies for the web application development and interface

Package Name	Version	Source
buffer	5.7.1	npm (Imported through node package manager)
process	0.11.10	
vite	4.3.9	
@fortawesome/fontawesome-free	6.4.0	
@fortawesome/fontawesome-svg-core	6.4.0	
@fortawesome/free-solid-svg-icons	6.4.0	
bootstrap	4.6	
jquery	3.7.0	
jquery-resizable-dom	0.35.0	
lib0	0.2.78	
lit-html	2.7.4	
nexusui	2.1.6	
tippy.js	6.3.7	
wavesurfer.js	6.6.3	
y-protocols	1.0.5	
y-websocket	1.5.0	
yjs	13.6.6	
Jitsi (through IFrame API)	-	Direct inclusion
soundtouch	0.1.30	

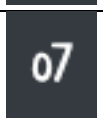
Appendix B

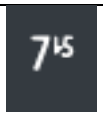
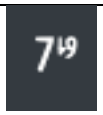
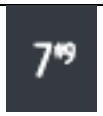
Chord symbols and encodings

The following table outlines the 67 chord variations for encoding between shorthand notations and Genius Jam Tracks (GJT) font. Rows 1-62 detail standard encodings. Rows 63-67 address special cases that required custom symbol assignments to prevent matching errors. Notable symbols include ‘X’ for unknown chords and ‘N’ for no-chord situations. For sus2 and sus4 chords, a combination of the encoded font letter 4 (which equals to sus) and a custom CSS ID ‘disable-font-label’ for the second part was used to achieve the desired appearance without enlarging the OpenType Font (OTF) file.

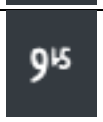
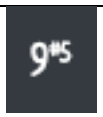
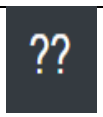
TABLE 5.3 Chord types mapping used between the shorthands in JAMS format to: a) GJT OTF ‘Encoded’ format; or b) ‘Description’ format, which provides the full name of the chord and serves as an extended tooltip; or c) ‘Simplified’ format, serving as tooltip in Edit mode.

#	Shorthand	Chord Symbol	Encoded	Description	Simplified
1	maj7		‘c’	Major seventh	M7
2	maj9		‘d’	Major ninth	M9
3	maj13		‘e’	Major thirteenth	M13
4	min		‘a’	Minor	m
5	min7		‘b’	Minor seventh	m7
6	minmaj7		‘f’	Minor major seventh	mM7
7	min9		‘h’	Minor ninth	m9
8	minmaj7(9)		‘g’	Minor major ninth	mM9

9	min9(11)		‘i’	Minor eleventh	m11
10	min9(11,13)		‘3’	Minor thirteenth	m13
11	aug		‘@’	Augmented	aug
12	dim		‘&gt’ (equals ‘>‘)	Diminished triad	dim
13	dim7		‘8’	Diminished seventh	dim7
14	hdim7		‘W’	Half diminished seventh	hdim7
15	hdim7(b9)		‘X’	Half diminished ninth	hdim9
16	hdim7(b9,11)		‘Y’	Half diminished eleventh	hdim11
17	5		‘5’	Fifth (or harmonic dyad)	5
18	7		‘7’	Seventh	dom7
19	9		‘9’	Ninth	dom9
20	9(11)		‘Q’	Eleventh	dom11
21	9(11,13)		‘U’	Thirteenth	dom13
22	maj6		‘6’	Major sixth	M6
23	maj6(9)		‘k’	Major sixth ninth	M6/9
24	maj(9)		‘=‘	Added ninth	M(add9)

25	min6		‘j’	Minor sixth	m6
26	min(b6)		‘R’	Minor flat sixth	mb6
27	min6(9)		‘Z’	Minor sixth ninth	m6/9
28	min(9)		‘%’	Minor added ninth	m(add9)
29	sus4(b7)		‘[‘	Seventh suspended fourth	7sus4
30	sus4(b7,13)		‘H’	Seventh suspended added thirteenth	7sus4(add13)
31	sus4(b7,9)		‘]’	Ninth suspended fourth	9sus4
32	sus4(b7,9,13)		‘&lt’ (equals ‘<‘)	Thirteenth suspended fourth	13sus4
33	sus4(b9)		‘v’	Seventh suspended flat ninth	7susb9
34	sus4(b13)		‘w’	Seventh suspended flat thirteenth	7susb13
35	7(b5)		‘p’	Seventh flat fifth	7b5
36	7(#5)		‘q’	Seventh sharp fifth	7#5
37	7(b9)		‘S’	Seventh flat ninth	7b9
38	7(#9)		‘s’	Seventh sharp ninth	7#9
39	7(b9,b5)		‘L’	Seventh flat ninth flat fifth	7b5b9

40	7(b9,#5)		‘J’	Seventh flat ninth sharp fifth	7b9#5
41	7(#9,b5)		‘O’ (uppercase o)	Seventh sharp ninth flat fifth	7b5#9
42	7(#9, #5)		‘M’	Seventh sharp ninth sharp fifth	7#5#9
43	7(b9,#9)		‘K’	Seventh flat ninth sharp ninth	7b9#9
44	7(b9,#11)		‘P’	Seventh flat ninth sharp eleventh	7b9#11
45	7(#9,#11)		‘N’	Seventh sharp ninth sharp eleventh	7#9#11
46	7(#11)		‘t’	Seventh sharp eleventh	7#11
47	7(b9,b13)		‘I’ (uppercase i)	Seventh flat ninth flat thirteenth	7b9b13
48	7(#9,b13)		‘0’ (zero)	Seventh sharp ninth flat thirteenth	7#9b13
49	7(b13)		‘m’	Seventh flat thirteenth	7b13
50	7(b7,#5,b9,#9)		‘?’	Altered seventh	7alt
51	min(#5)		‘V’	Minor sharp fifth	m#5
52	maj7(b5)		‘1’	Major seventh flat fifth	M7b5
53	maj7(#5)		‘z’	Major seventh sharp fifth	M7#5
54	maj7(#11)		‘x’	Major seventh sharp eleventh	M7#11

55	maj9(#11)		‘y’	Major ninth sharp eleventh	M9#11
56	9(b5)		‘T’	Ninth flat fifth	9b5
57	9(#5)		‘n’	Ninth sharp fifth	9#5
58	9(#11)		‘r’	Ninth sharp eleventh	9#11
59	9(11,b13)		‘2’	Ninth flat thirteenth	9b13
60	9(#11,13)		‘l’ (lowercase L)	Thirteenth sharp eleventh	13#11
61	7(b9,11,13)		‘u’	Thirteenth flat ninth	13b9
62	7(#9,11,13)		‘o’ (lowercase O)	Thirteenth sharp ninth	13#9
63	maj	**NO SYMBOL**	‘(M)’	Major	**EMPTY TEXT**
64	sus2		‘4<text id=«disable-font-label»>2</text>’	Suspended second	sus2
65	sus4		‘4<text id=«disable-font-label»>4</text>’	Suspended fourth	sus4
66	N		‘<text id=«disable-font-label»>N.C.</text>’	No chord	No Chord
67	X		‘<text id=«disable-font-label»>??</text>’	Unknown	Unknown