Bericht

Software Design (BA-MECH-22)

Abschlussprojekt

Bachelor Mechatronik, Design & Innovation

3. Semester

Lehrveranstaltungsleiter: Julian Huber

Jahrgang: BA-MECH-22-2B

Studenten: Vella Nedelcheva, Pavlo Slyvka, Benedikt Reimeir

01. März 2024

Inhaltsverzeichnis

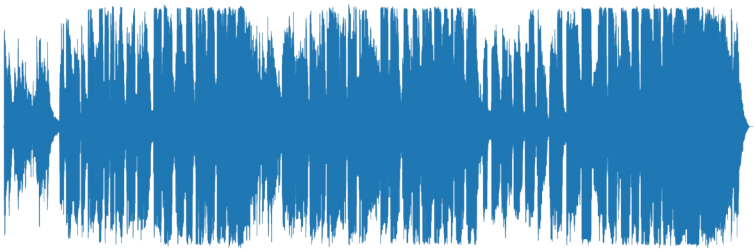
1. Einführung

Es soll eine Anwendung mit **Web-UI** entwickelt werden, mit der **Musikstücke identifiziert** werden können, indem nur ein kurzer Abschnitt des Musikstückes untersucht wird → wie ***Shazam***, ***Soundhound***, etc.

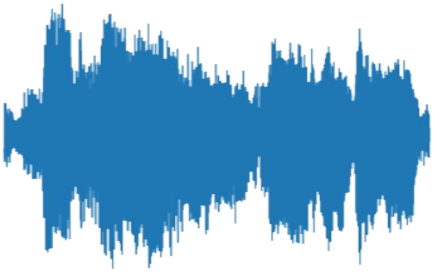
- Die Umsetzung erfolgt in **Python** mit Objektorientierung
- Die Web-UI soll mit streamlit umgesetzt werden
- Die Datenbank soll mit tinydb umgesetzt werden

1. Funktionsweise

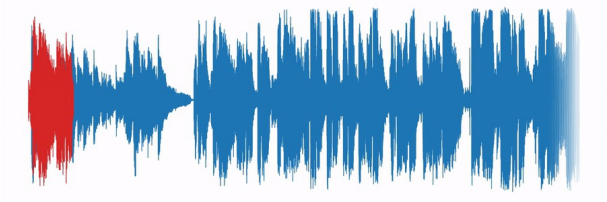
- Die gesamte Erklärung kann in [2] entnommen werden
- Digitale Musik ist ein **zeitdiskretes Signal**
- Kann durch seine Waveform dargestellt werden



- Wenn wir ein Musikstück identifizieren wollen, haben wir normalerweise nur kleine Abschnitte zur Verfügung
- Wo passt dieser Schnipsel in das gesamte Musikstück?



- Simple Lösung: **Brute Force**
- Wir vergleichen den Schnipsel mit allen möglichen Positionen im Musikstück

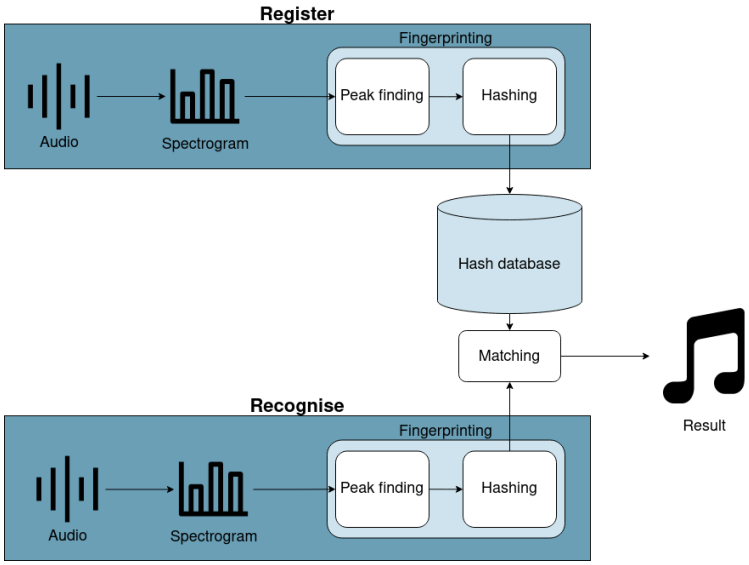


- Wenn wir eine Übereinstimmung finden, haben wir das Musikstück identifiziert
- Müssen dies für **alle** Musikstücke machen die in unserer Datenbank liegen[1]

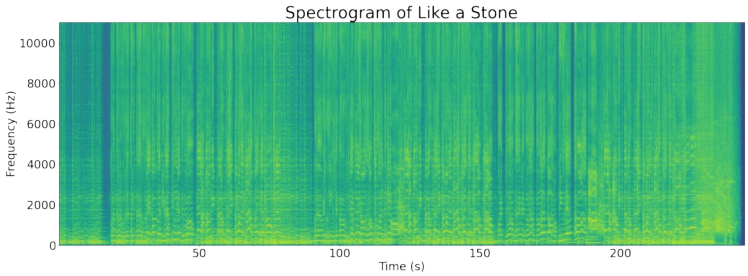
1

[2]1. Einführung

- **Brute Force** ist sehr ineffizient → nicht praktikabel
- Wir wollen **Fingerprint** für unser Musikstück erstellen und diese vergleichen
- Workflow in 2 Teilen →**Register** und **Recognise**



- Es werden Spektrogramme der Musikstücke erstellt
- Trägt die Frequenz über die Zeit auf
- Farbe entspricht der Signalstärke
- Peaks im Spektrogramm **verteilt über Zeit und Frequenz** als Fingerprints



- Die Spektrogramme werden mit **Fensterung** erstellt → idealerweise auf eine etablierte Bibliothek zurückgreifen
- Es wird davon ausgegeangen, dass die Musikstücke die selbe **Sampling Rate** haben
- Die Musikstücke im Examples-Ordner haben alle eine (unübliche) Sampling Rate von 22050Hz → häufiger wäre die Sampling Rate 44100H

1. Anforderungen der Aufgabenstellung

[3]1.1. Anforderungen der minimalen Aufgabenstellung

- Softwareprojekt in **Python** mit **Objektorientierung**
- Versionskontrolle über **git** & **GitHub**
- Anwendung mit Web-UI:
- **Register workflow** → Musikstücke einlernen
- **Recognise workflow** → Musikstücke identifizieren
- Hochgeladenes Musikstück muss in Browser wiedergegeben werden können
- Dokumentation - Zwei Möglichkeiten
- requrirements.txt-Datei mit allen packages
- **README.md** im Repository mit Anleitung zur Installation und Ausführung, Umgesetzten Erweiterungen, UML-Diagrammen der Softwarestruktur, Quellen zu verwendeten Inhalten etc.
  ODER
- Kurzer **Bericht als pdf-Dokument** mit selbem Inhalt

1.2. Anforderungen der möglichen Erweiterungen

- Aufzeichnung über Mikrofon
- Input über YouTube-Link
- Link zu identifiziertem Musikstück auf Spotfity/YouTube
- Recommendation System → ähnliche Fingerprints bedeutet höchstwahrscheinlich ähnliche Musik
  - Graphenbasiert
  - Basierend auf History etc.
- Meta-Daten zu Musikstücken (Interpret, Album, etc.) bestimmen
- Albumcover des Musikstückes bestimmen und anzeigen
  - z.B.: mit DuckDuckGo Such-API
- Bestimmung & Visualiserung der beats-per-minute des Musikstückes
- History der letzten identifizierten Musikstücke
- Visualisierungen:
  - Datenbank
  - Fingerprints
  - Histogramm der Matches
  - Waveform des Musikstückes plotten
  - etc.
- Benchmarking der Anwendung:
  - Wie lange dauert das Einlernen?
  - Wie lange dauert die Identifikation?
  - Wie hängt dies von der Größe der Datenbank ab?
- Deployment auf **eigenem** Server
- z.B.: Oracle Cloud Free Tier

3

[4]2. Dokumentation der minimalen Aufgabenstellung und der Erweiterungen

2.1. Ornder, UI Datei und Modulen

Zuerst wurde den Beispiel von Github mit dem Befehl **git clone** geklont. Danach wurde in **shazam_2/build/lib/abracadabra** den Ordner **classes** und die Datei **User_Interface(UI)** erstellt. Um den Projekt erfolgreich zu starten, wurden die Modulen tinydb, pydub, streamlit, numpy, pandas, os und sys installiert und importiert. Die Daten wurden in **music_hashes.json** gespeichert.

2.2. In User Interface

Teach song

In der bereits genannten Datei **(UI)** wurden die Teile Teach Song, Recognise Song und About implentiert. Das Hochladen von einer Audiodatei geschieht über die Bibliothekfunktion von streamlit **file_uploader.** Um den richtigen Format zu gewährleisten wurde **audio** von dem gleichen Modul verwendet. Wenn die **selected_file** nicht None ist, dann wird der Code von **temp_dir** bis zum **f.write** implementiert. Er dient dazu, eine hochgeladene Datei vorübergehend in einem temporären Verzeichnis zu speichern, um sie später zu verarbeiten oder zu manipulieren. Ohne den Teil wird den Pfad der Datei von der **fingerprint_file** der Klasse **Fingerprinting** nicht erkannt, die zur Erzeugung von einer Liste von hashes dient.

fingerprint_file(): In dieser Methode werden vier andere Funktionen der Klasse Fingerprinting aufgerufen. In **file_to_spectrogram** wird die spectogram der Audiodatei berechnet . Dann werden drei Werte Zeit, Frequenz und **Sxx** bekommen. Der dritte wird an **find_peaks** übergeben, um die **peaks (**Spitzwerte**)** in dem Spectrogram zu finden und eine Liste aus diesen zurückzugeben. Dieses Ergebnis, die Zeit und die Frequenz werden von **idxs_to_tf_pairs** genommen, um diese in Werte umzuwandeln. Mit der Methode **hash_points** werden alle Hashes für eine Liste von Peaks generiert. Die Funktion iteriert durch diese und erzeugt einen Hash für jeden Peak innerhalb seiner Zielzone (des Peaks). Es gibt eine Liste von Tupeln (**Hash, Zeitversatz, song_id**) zurück.

store_song(): Um die Daten des Musikstücks in der Datenbank zu speichern wird die Methode **store_song** der **db_manager** aufgerufen. In dieser wird eine neue id des Songs generiert und dies mit der verfügbaren in der Query verglichen. Wenn dieser id in der Datenbank nicht existiert, dann werden die new hashes, id, title, artist und album dementsprechend an **__insert_fingerpint** und **__insert_song_info** weitergegeben und dadurch erfolgreich in der Tabelle gespeichert. Nach einem erfolgreichen Importieren wird eine Nachricht “Added song to library” angezeigt, die für 1 bis 2 Sekunden schnell links auf den Bildschirm rauskommt.

Recognise song

In diesem Teil des Codes in **UI** werden die Datei auf die gleiche Weise hochgeladen. Mit der Methode **recognise_song** der **recogniser_class** wird den besten Match des Musikstücks gefunden. Für das Ziel werden die Hashes der Audiodatei wieder so generiert und danach an **get_matches** der db_manager übergeben. Diese nimmt eine Liste von Tupeln **new_hashes_data** entgegen, wobei jedes Tupel aus einem Hashwert **h,** einer Zeit **t** und einem nicht verwendeten Wert besteht.

4

2. Dokumentation der minimalen Aufgabenstellung und der Erweiterungen

Die Funktion erstellt zunächst ein Dictionary **h_dict**, das den **h** auf die entsprechende Zeit **t** abbildet, und fügt in der Liste **new_hashes** den neuen Hashwert hinzu. Dann wird es für Matches **(von alle h)** in der Datenbank gesucht, indem zwei Bibliothekmethode aufgerufen werden **search** und **Query.** Zum Schluss wird es überprüft, ob genüg Übereinstimmungen gefunden werden,wobei die Schwelle durch **threshold** gleich 800 definiert ist. Wenn die Anzahl der gefundenen Ergebnisse kleiner als dieser Variable ist, wird None zurückgegeben. Anderenfalls wird ein leeres **result_dict** erstellt. In einer Schleife über die gefundenen Einträge **results** werden diese nach der ID gruppiert und in **result_dict** gespeichert. Schließlich wird **result_dict** zurückgegeben, das die gefundenen Ergebnisse nach ID gruppiert enthält. So werden die neue matches bekommen. Wenn es keine gefunden werden, wird None[5] zurückgegeben. Anderenfalls wird **best_matches** aufgerufen.

best_matches(): Sie durchläuft die Übereinstimmungen und wählt den Song mit der höchsten Punktzahl bzw. score aus, wobei die Punktzahl durch die Funktion **score_match** bestimmt wird. Wenn dieser größer ist als der letzten **best_score,** wird ihr (der Variable **best_score**) der neue Wert zugewissen, als auch die **song_id** an der **matched_song .**

recognise_song(): In der **recognise_song** wird in einer If Bedingung überprüft,ob **matched_song True** ist. Wenn das der Fall ist, wird die Information aus der Datenbank system von **db_manager** an **info** übergeben, sonst wird None zurück. Falls **info** nicht **None** ist, wird **info** zurückgegeben.

Recognise_song_in UI: Sollte es keine Audiodatei gefunden, wird die Nachricht “Teach a song!” angezeigt. Sonst werden die Daten: **id, title, Artist und Album** angezeigt. Nach gewissener Zeit mit der Hilfe der Bibliothekfunktion **librosa, matplotlib.pyplot** und **plotly.graph_objs** werden die Fingerprints und das Histrogram erledigt.

Die Klasse Waveform: da befindet sich enkapsuliert die Methode **plot_waveform**. Diese wird verwendet, um das Wellenformdiagramm eines Musikstücks zu erstellen und anzuzeigen. Sie übernimmt als Parameter die Audiodaten **self.audio_data** und die Abtastfrequenz **self.sr**. Zuerst werden die Zeitwerte für die x-Achse berechnet, indem die Länge der Audiodaten durch die Abtastfrequenz geteilt wird. Dies ermöglicht es, die Audiodaten im zeitlichen Verlauf darzustellen. Anschließend wird ein Plot des Wellenformdiagramms erstellt, indem die bereits genannte Daten gegen die berechneten Zeitwerte aufgetragen werden. Dies wird mithilfe der **plotly** realisiert. Falls während des Plottens ein Fehler auftritt, wird eine entsprechende Fehlermeldung angezeigt, um den Benutzer über das Problem zu informieren.

Die Klasse Histrogram: Danach wird es des entsprechenden Songs implemenitiert, was über zwei Methoden geschieht. Die Funktion **process_audio_for_histogram** übernimmt Audiodaten und wandelt diese in einem **numpy** Array um. Nach dem berechnet das Histogramm, der mit einer vordefinierten Anzahl von Amplituden(laut dem Modul sind **bins)** ist und gibt das berechnete Histogramm sowie die Bin-Kanten zurück. Diese Funktion wird von numpy verwendet. Prinzipiell sollten die Amplituden so erhalten. Im Falle eines Fehlers während der Verarbeitung wird dieser behandelt, indem sowohl für das Histogramm als auch für die Bin-Kanten der Wert **None** zurückgegeben wird. In der Funktion **read_audio_file** wird eine durch den Eingabepfad angegebene Audiodatei gelesen. Dabei wird die **librosa-**Bibliothek verwendet, um die Audiodatei

5

mit einer Abtastrate von **44100 Hz** zu laden. Anschließend gibt sie die geladenen Audiodaten zurück. Sollte es beim Lesen zu einem Fehler kommen, wird eine entsprechende Fehlermeldung ausgegeben und es wird der Wert **None** zurückgegeben. Dann wird in der User_Interface über ein **If** Bedingung das Histogram nur dann gezeichnet und angezeigt wird, wenn sowohl dieses selbst, als auch die Bin-Kanten erfolgreich berechnet wurden. Wenn dies der Fall ist, wird es mithilfe von **matplotlib** geplottet und angezeigt. Damit werden die Verteilung der Amplitudenfrequenzen des erkannten Songs visualisiert. Mit der Funktion **search_youtube** im Recognise from recording werden videos im Internet dem Match gefunden. Den gleichen Logik wird benutzt, damit es mit der microfon.recording_function ein Song zu erkennen.

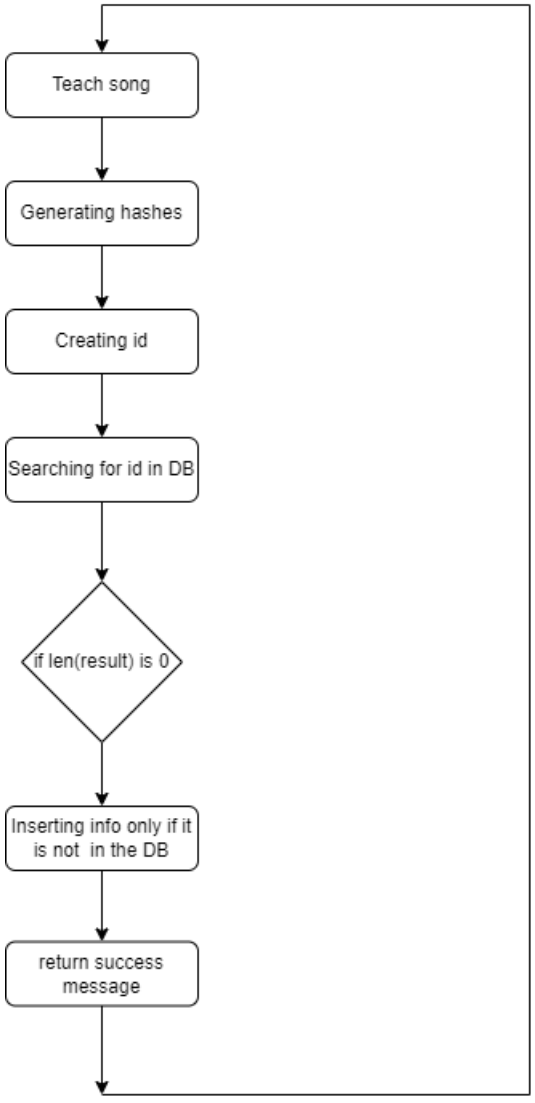


Abbildung 1. Diagramm von Teach song

[7]3.2.Diagramm von Recognise Song

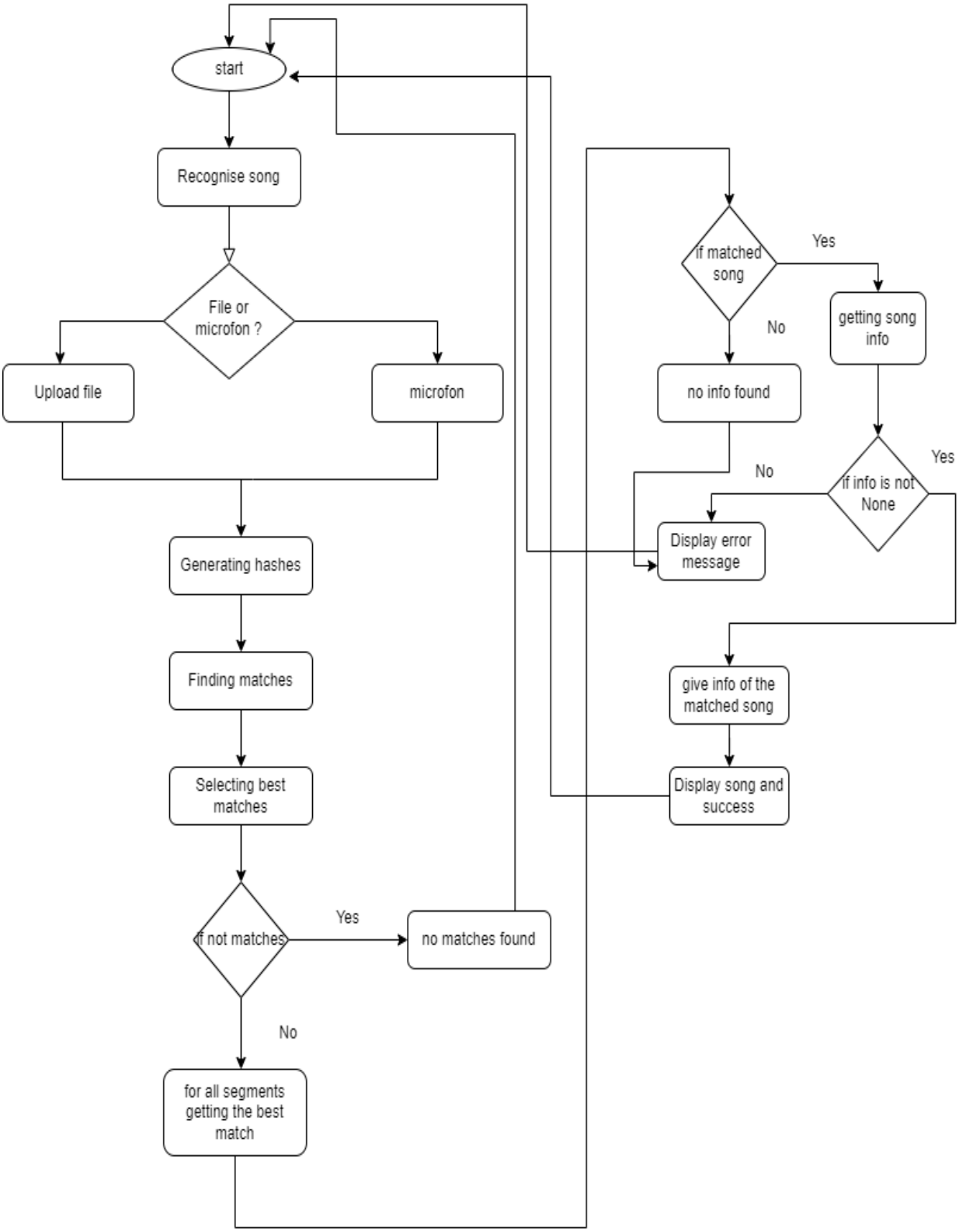


Abbildung 2. Diagramm von Recognise song

7

Abbildungen der Diagramme

Literaturverzeichnis

[1] *An Industrial-Strength Audio Search Algorithm*, Avery Li-Chun Wang, 2003: https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf

[2] *abracadabra: How does Shazam work?*, Cameron MacLeod, 2022:

https://www.cameronmacleod.com/blog/how-does-shazam-work

[3] *Audio Fingerprinting with Python and Numpy*, Will Drevo, 2013:

https://willdrevo.com/fingerprinting-and-audio-recognition-with-python/

[4] w3schools python tutorials:

https://www.w3schools.com/python/

[5] Black Box AI:

https://www.blackbox.ai/

[6] Deepl Translator:

https://www.deepl.com/translator