Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Застосування інструментів штучного інтелекту для музичних платформ

Засвідчую що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань
Студент гр. ІПЗ-21-2
_____ /В.С. Савчук/

Керівник
кваліфікаційної роботи _____ /Н. Н. Шаповалова/

Завідувач кафедри _____ /А. М. Стрюк/

Кривий Ріг
2025

Криворізький національний університет
Факультет: Інформаційних технологій
Кафедра: Моделювання та програмного забезпечення
Ступінь вищої освіти: бакалавр
Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Зав. кафедри
_____ А.М. Стрюк
«___»_____ 20__ р.

ЗАВДАННЯ
На кваліфікаційну роботу
студенту групи ІПЗ-21-2 Савчуку Владиславу Сергійовичу

1. Тема: Застосування інструментів штучного інтелекту для музичних платформ затверджено наказом по КНУ № 200с від «10» квітня 2025 р.
2. Термін подання студентьом закінченої роботи: «09» червня 2025 р.
3. Вихідні дані по роботі: Розробити моделі які будуть виконувати змістовний аналіз аудіо,
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): Виконати аналіз існуючих інструментів штучного інтелекту, та методів машинного навчання, розробка архітектури програмного забезпечення, обрання методів для аналізу аудіо, розробити обгортку для використання розробленних моделей.
5. Перелік ілюстративного матеріалу: графіки, спектрограми, скриншоти існуючих аналогів, функціональні схеми, схеми топологій нейромереж, графічні приклади роботи k-NN, схематичне зображення інтерфейсу, знімки екранних форм.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літератури за тематикою роботи	20.04.2025 – 26.04.2025
2	Розгляд існуючих аналогів	27.04.2025 – 02.05.2025
3	Підготовка матеріалів першого розділу роботи	03.05.2025 – 09.05.2025
4	Аналіз існуючих алгоритмів обробки аудіо	10.05.2025 – 15.05.2025
5	Розробка архітектури ПЗ	16.05.2025 – 21.05.2025
6	Підготовка матеріалів другого розділу роботи	22.05.2025 – 26.05.2025
7	Початок розробки інтерфейсу програми	27.05.2025 – 30.05.2025
8	Початок розробки API програми	31.05.2025 – 03.06.2025
9	Навчання обраних моделей	04.06.2025 – 06.06.2025
10	Підготовка матеріалів третього розділу роботи	07.06.2025 – 08.06.2025
11	Оформлення пояснювальної записки	08.06.2025 – 09.06.2025

Дата видачі завдання: «11» квітня 2025 р.

Студент _____/В. С. Савчук/

Керівник роботи _____/Н. Н. Шаповалова/

РЕФЕРАТ

АНАЛІЗ АУДІО, МУЗИЧНІ ПЛАТФОРМИ, ШТУЧНИЙ ІНТЕЛЕКТ, МЕТОДИ МАШИННОГО НАВЧАННЯ, КЛАСИФІКАЦІЯ ЖАНРІВ, РОЗПІЗНАВАННЯ ЖАНРУ, ЕМОЦІЙНИЙ АНАЛІЗ МУЗИКИ, ВИЗНАЧЕННЯ ТОНІКИ, НЕЙРОННІ МЕРЕЖІ.

Пояснювальна записка: 51 с., 32 рис., 1 дод., 16 джерел.

Метою кваліфікаційної роботи є дослідження використання моделей машинного навчання на музичних платформах та розробка власних моделей для автоматичної класифікації жанрів музики, емоційного аналізу музичних композицій, визначення тоніки та інших акустичних характеристик.

У роботі проведено аналіз існуючих рішень у галузі автоматичного аналізу музики, виділено їх переваги та недоліки, проаналізовано підходи, які використовуються у комерційних рішеннях музичних платформ. Досліджено методи цифрової обробки аудіосигналів та сучасні підходи до класифікації музичного контенту.

Для розв'язання поставлених завдань використовувалися методи машинного навчання, включаючи алгоритм k-найближчих сусідів (k-NN) та архітектури повнозв'язних нейронних мереж. Також застосовувалися методи для боротьби із перенавчанням.

Для роботи із навчальними моделями було розроблено зручний веб-інтерфейс, який наочно показує всі результати аналізу у вигляді звіту.

Основні положення щодо роботи із методом k-NN, який використовувався для класифікації музичних жанрів, опубліковано у вигляді тез на XVIII Всеукраїнській науково-практичній конференції "Комп'ютерні інформаційні системи та мережі" (КІСМ-2025).

ABSTRACT

AUDIO ANALYSIS, MUSIC PLATFORMS, ARTIFICIAL INTELLIGENCE, MACHINE LEARNING METHODS, GENRE CLASSIFICATION, GENRE RECOGNITION, EMOTIONAL ANALYSIS OF MUSIC, KEY DETECTION, NEURAL NETWORKS.

Thesis in 51 p., 32 fig., 1 app., 16 sources.

The aim of the qualification work is to study the use of machine learning models on music platforms and develop custom models for automatic music genre classification, emotional analysis of musical compositions, key detection and other acoustic characteristics.

The work analyzes existing solutions in the field of automatic music analysis, identifies their advantages and disadvantages, and examines approaches used in commercial solutions of music platforms. Methods of digital audio signal processing and modern approaches to music content classification are studied.

Machine learning methods were used to solve the assigned tasks, including the k-nearest neighbors (k-NN) algorithm and fully connected neural network architectures. Methods for combating overfitting were also applied.

A convenient web interface was developed for working with training models, which visually displays all analysis results in the form of a report.

The main provisions regarding the work with the k-NN method used for music genre classification were published as abstracts at the XVIII All-Ukrainian Scientific and Practical Conference "Computer Information Systems and Networks" (CISM-2025).

ЗМІСТ

ВСТУП

У сучасному світі, цифрові технології продовжують вдосконалюватися забезпечуючи користувачам більш зручний досвід використання цифрових продуктів, програм, інтернет-сервісів, тощо. Останнім часом стрімко розвивається галузь машиного навчання, зокрема штучного інтелекту, та нейромереж. Методи й інструменти штучного інтелекту все частіше інтугрують у різні типи цифрових продуктів.

Стрімко розвиваються і інші напрямки, такі як індустрія музики. Багато стрімінгових сервисів починають використовувати у своїх продуктах, для більш ревалентних рекомендацій користувачам різних треків, задля збільшення охоплення та свого прибутку, для створення плейлистів на смак коритувача. Завдяки таким підходам в наш час, все більше і більше з’являється нових музикантів які до цього були не відомі.

Методи машинного навчання, надають користь від їх використання не тільки звичайним користувачам, а й самим виконавцям. Ці методи можуть бути корисні для аналізу ринку музики, і навіть передбачити успіх від публікації майбутнього треку. Також завдяки таким методам, можно виконувати пошук плагіату для його запобігання, та захисту авторського права музикантів.

Метою даної кваліфікаційної роботи є дослідження інструментарію штучного інтелекту та його практичне використання, для розробки додатку який зможе допомогти музикантам із розумінням жанрової класифікації, та й простим користувачам знаходити схожі за вподобаннями треки.

1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Критичний аналіз літературних джерел за темою

Перед початком розробки будь-якого рішення, необхідно провести аналіз та дослідження за темою роботи. А саме необхідно детальніше розглянути як саме в літературі розв'язуються подібні проблеми класифікації, обробки аудіо, та аналіз характеристик музики.

Щодо класифікації музики основною проблемою є те, що між жанрами немає якогось чіткого визначення та меж. Жанри були сформовані людьми, та просто є сукупністю спільних ознак, які поділяють їхні представники. Що ускладнює структуризацію та класифікацію за допомогою алгоритмів. Представники жанрів розрізняють жанри за допомогою ритмічних структур, звуковим складом, тональністю, та ін. Проте навіть зараз досить часто анотація виконується вручну [1].

Тому щоб виконати класифікацію, для початку, окрім звукових файлів та їх жанрових анотацій, необхідно зробити так щоб, комп'ютер міг обробити ці дані, для цього можна використати методи до прикладу бібліотеки librosa [2] для Python, яка широко застосовується у аналізі аудіо сигналу. Ця бібліотека допомогає дістати із аудіо сигнали, деякі аудіоознаки, зокрема: мел-частотні кепстральні коефіцієнти (MFCC), спектральний центроїд, швидкість перетину нуля [3].

На рисунку 1.1 зображено спектральний центроїд. Спектральний центроїд є однією з основних характеристик аудіо. Він відображає частотний розподіл у момент часу, обчислюється як середнє зважене значення частот, присутніх у спектрі, де вагами виступає амплітуда, кожної з частот. За допомогою цього можна зрозуміти на якій частоті зосереджена основна енергія сигналу.

Моментом часу є вікно яке обрано відповідно до дискретизації, яка була обрана при обробці аудіо файла, за допомогою цього можна побачити динаміку зміни спектрального складу звука протягом всього звучання.

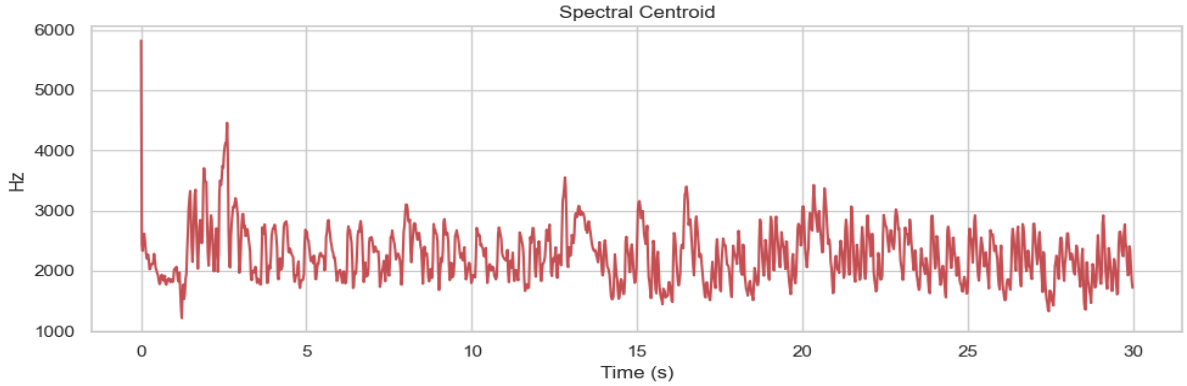

Рисунок 1.1 – Спектральний центроїд

На цьому графіку можна спостерігати короткий стрибок центроїда на початку треку, що вказує на імпульсний звук, чи якиїсь шум, накшталт того що виникає при підключенні електрогітари до підсилювача, короткий високочастотний шум, після чого значення стабілізуються на значенні від 2000 до 3000 Гц. Що свідчить про домінування середніх частот в аудіофрагменті. Про що можна зробити висновок що цей трек відноситься до електроники, поп, чи до року, де присутні синтетичні інструменти та вокал.

На рисунку 1.2 зображено графік швидкості перетину нуля аудіосигналу. Він відображає частоту з якою сигнал змінює свій знак з позитивного на негативний, чи навпаки.

Рисунок 1.2 – Швидкість перетину нуля

За допомогою цих аудіоознак можна виконати классифікацію за жанрами, із використанням машиного навчання, методи якого зможуть уловити закономірності між цими показниками та жанрами, що може дати досить високу точність.

Також класифікувати можна не тільки за числовим представленням цих аудіоознак, а також і за спектрограммою, оскільки вона містить всі ці признаки проте тільки у вигляді зображення. Найчастіше використовуються mel-спектрограми, або log-scaled спектрограми, які найкраще відображають фізіологічне сприймання звуку.

На рисунку 1.3 зображена mel-спектрограма, вона показує енергетичний розподіл сигналу за частотами, ця спектрограма оперує мел-шкалою, яка наближена до людського сприйняття висоти тону. По осі Y, відкладена частота від 0 до 8192 Гц, по осі X, час у секундах, а яскравість відображає гучність у децибелах.

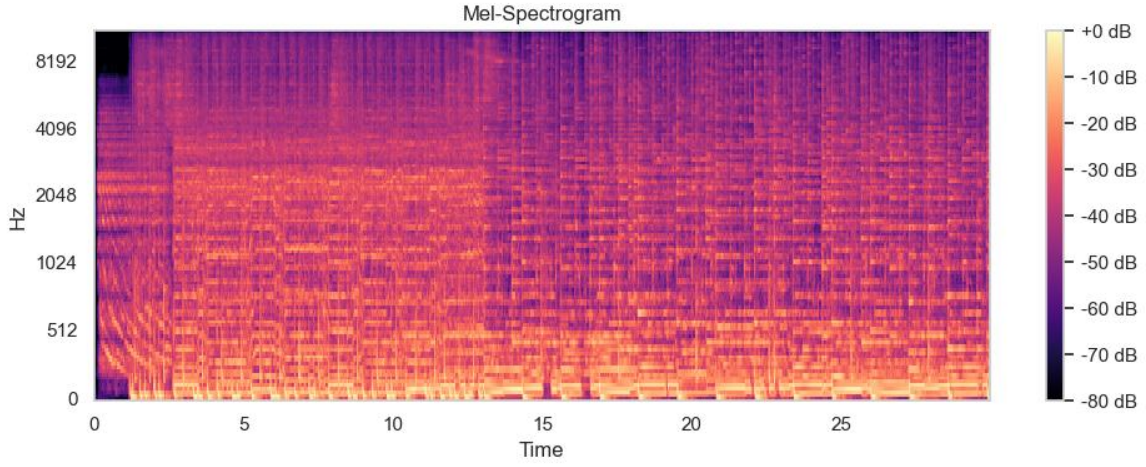

Рисунок 1.3 – Mel-спектрограма

Конкретно у цьому прикладі, можна зробити висновок що найбільша енергія зосередеженна між 2000 та 3000 Гц, що характерно для вокалу, та ударного інструменту, можливо для електрогітари. Завдяки цьому можна зробити висновок до якого жанру належить музика, наприклад хіп-хоп чи електроника.

Тепер необхідно розглянути методи машиного навчання та архітектури нейромереж які використовуються при вирішенні таких задач. Загалом основних задачі всього дві, класифікація жанрів музики та знаходження найбільш схожої на задану.

Для таких задач використовуються конвуляційні або згорткові мережі (CNN), k-найближчих сусідів (k-NN) та повнозв'язні мережі (Dense). Все залежить від підходу до задачі. Можна розглядати весь аудіопоток із деякою дискретизацією, до прикладу відрізками по 10 секунд, передавати його до 1D CNN, виконувати згортки, і отримати від нейромережі досить непоганий результат, проте модель вийде досить складною оскільки розмірність вектору буде дуже великою навіть при 10 секундах. При дискритизації 5000 герц, моно канальний звук, 10 секунд аудіо, розмірність вектору буде дорівнювати 50000 тисяч елементів що дуже багато, проте згортаючи такий вектор можна виокремити найважливіші в ньому елементи, і зробити висновок про жанр на основі цих даних[4].

Розглянемо детальніше роботу зі спектром, тобто з його графічним відображенням, що насправді має бути більш легкою задачею, з точки зору препроцесінга який непотрібен зовсім[5], проте через це збільшується значно складність самої нейронної мережі, що може негативно вплинути на її швидкодію.

Варто також звернути увагу на те що, аудіо може зберігатися у різних форматах, і вони поділяються на: Lossless, тобто без стиснення, до цих форматів відноситься WAV, FLAC, ALAC, які зберігають всю аудіоінформацію, та на Loosy, стиснення із втратами, до таких форматів відноситься MP3, AAC, OGG та ін. Загалом стиснення може призвести до дуже суттєвої втрати даних[6]. Тому щоб в цьому переконатися можна подивитися на рисунок 3.4, на якому зображено три mel-спектрограми у двох форматах FLAC (Lossless) та MP3 із бітрейтом 320 біт (Loosy).

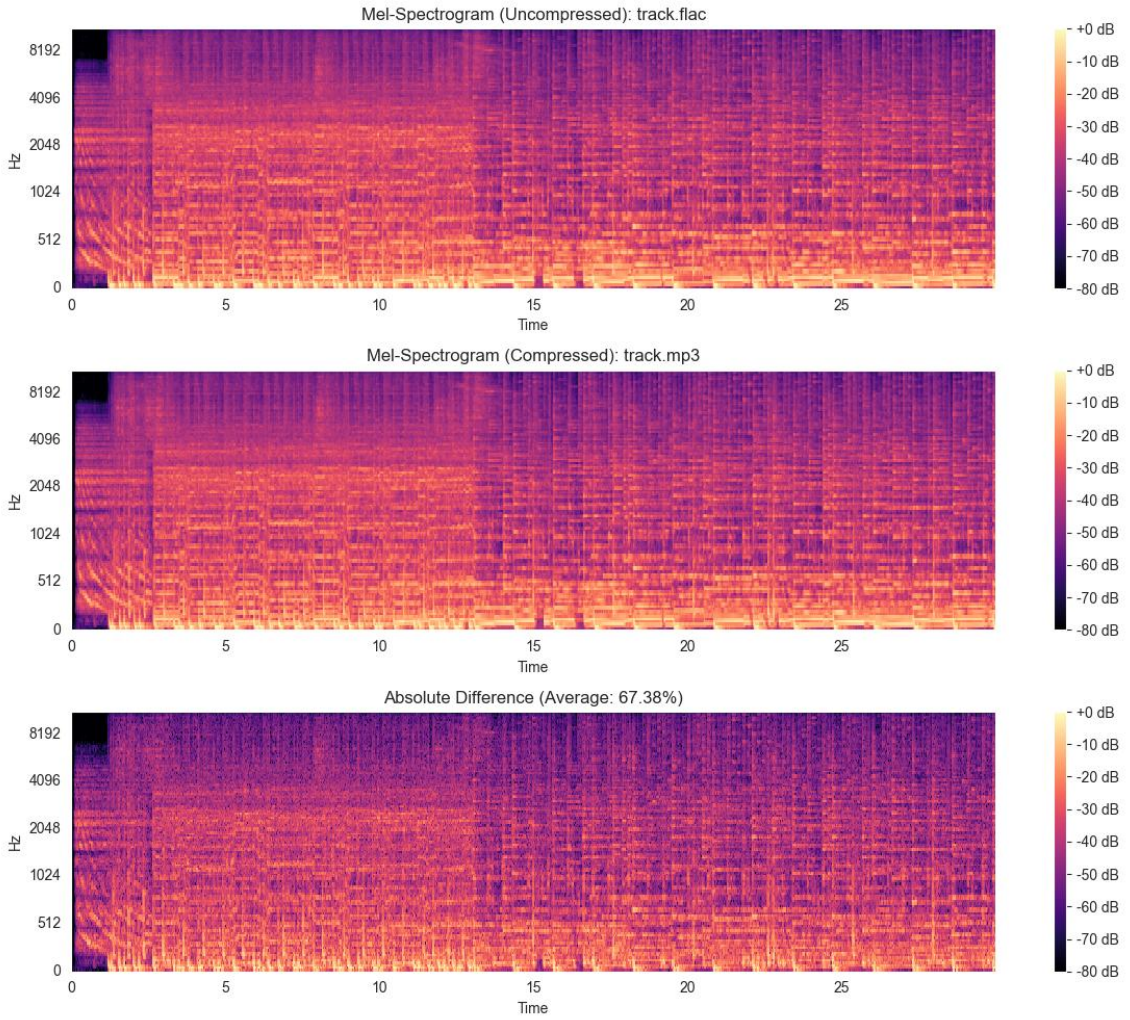

Рисунок 1.4 – Mel-спектрограми MP3 та FLAC, та їх різниці

Візуально обидві спектрограми виглядають дуже подібними, але спектрограма знизу, яка дозволяє оцінити наскільки сильно MP3, виконує стиснення та на скільки велика різниця між цими двома форматами на рівні сигналу. І відмінності є досить великими, звичайна людина майже може не відчути різницю між цими двома аудіодорожками, середня різниця за усіма типами частот становить приблизно 67%. Тобто третину даних втрачається через стиснення. Проте варто зауважити, що при цьому розмір файлу також зменшується кратно, що дає можливість легше обробляти такі аудіофайли, при цьому зберігаються всі важливі характеристики, а також MP3 є більш розповсюдженним форматом ніж FLAC.

Стиснення MP3, відбувається відповідно до психоакустичної моделі [7], тобто інформація яку вухо людини не здатне сприйняти та взагалі розрізнити

видаляється, тому й здається на слух що якість не змінюється взагалі, що можливо буде впливати на побудовану модель за допомогою такого датасету краще. Також все залежить від бітрейту який використовується. На рисунку 1.4 було зображено MP3 файл 320 кбіт/с, якщо ж використати 160, 128 або ж навіть 8 кбіт/с, різниця буде дуже помітною ніж при найкращому бітрейті для цього формату в 320 кілобіт. Також втрати частот розподілені не рівномірно, що відповідає психоакустичній моделі, яка використовується при стисненні.

Проблемою при виконанні роботи також може бути створення свого датасету для аналізу аудіо, оскільки майже всі треки захищені авторським правом. Тому щоб уникнути юридичних проблем, необхідно про це подбати і використовувати аудіо-матеріали тільки в рамках Fair Use. Наприклад генерування нової музики схожої на існуючу із використанням оригіналів може бути порушенням авторського права, проте це питання і досі вивчається і законодавство поки не має якихось обмежень в галузі AI, проте це може змінитися [8].

1.2 Актуальність теми кваліфікаційної роботи

Музичні платформи продовжують активніше впроваджувати у свої сервіси інструментарій із використанням штучного інтелекту. Тому доцільно розглянути деякі з цих сервісів, щоб зробити висновки щодо актуальності роботи, а також подивитися на переваги та недоліки цих сервісів. До популярних сервісів які пов’язані з темою роботи належать: SpotifyS, YouTube Music[10], SHAZAM[11], SoundCloud[12].

В умовах стрімкого розвитку цифрових технологій, і зростання кількості аудіоконтенту від маловідомих авторів, які не мають своєї аудиторії, потреба в ефективних системах рекомендацій, класифікації, пошуку треку стає все актуальнішою. Сучасний користувач стикається з великою кількостью аудіоконтенту, що сильно ускладнює пошук композицій за вподобаннями, проте цю проблему можна вирішити завдяки пошуку схожих композицій, за темпом, спектром, настроєм, семантикою і т.д. Проте щоб виконати такий

ефективний аналіз, необхідно використовувати сукупність класифікаційних нейромереж. Такий підхід може потребувати дуже великих потужностей, тому необхідно балансувати між точністю, і ефективністю.

Тепер розглянемо кожен з сервісів детальніше, і почнемо із Spotify. Він є одним з найбільших світових стрімінгових сервісів, що за передплату пропонує безперешкодне прослуховання в будь-якій точці світу сотні мільйонів треків на будь-який смак, від різних виконавців, жанрів. Проте сервіс можна використовувати і безкоштовно прослуховуючи музику із рекламою. Платформа працює на смартфонах, планшетах, комп'ютерах, телевізорах, та навіть магнітолах у автомобілях, що і надає сервісу таку велику популярність. Проте не тільки через це Spotify популярний, а й через його систему рекомендацій, яка є однією з самих просунутих у музичній сфері.

Як видно з рисунку 1.5, Spotify, аналізує зіставлений мною плейлист, у якому є тільки аніме опенінги.

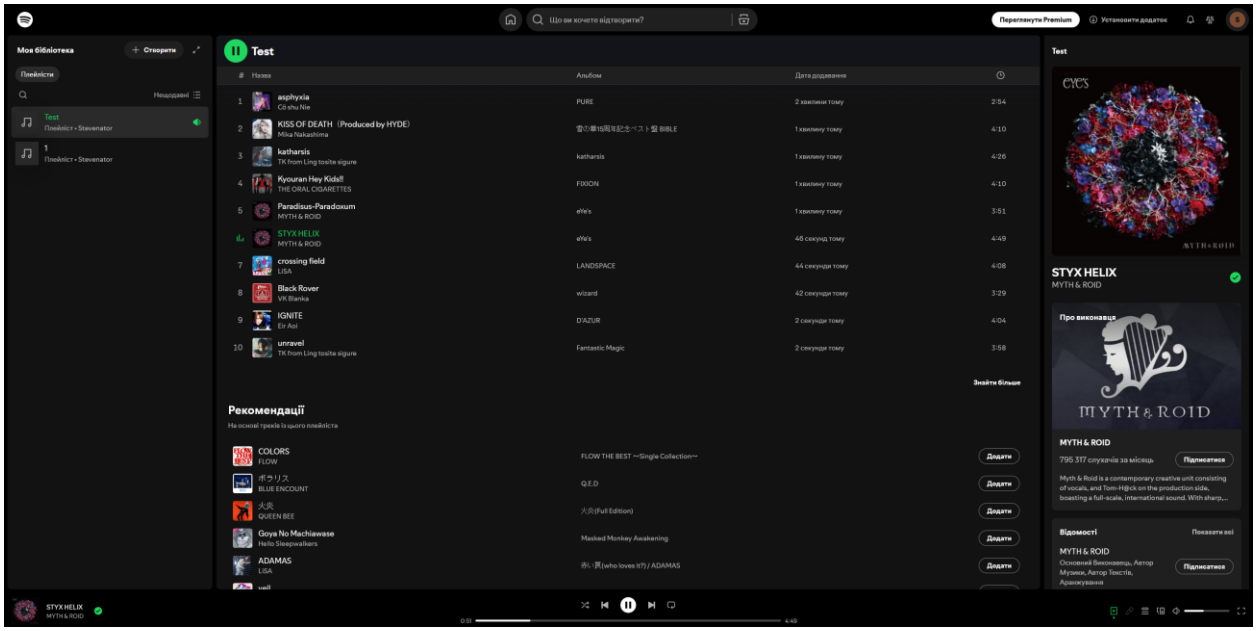

Рисунок 1.5 – Вікно із відкритим плейлистом Spotify

І використовує свою систему із аналізом характеристик треків за API EchoNest, який використовує у собі такі харатеристики, ці аудіоознаки

навідміну від librosa, є високорівневими які включають в себе сукупність базових ознак у аудіо:

- danceability – показує наскільки танцювальним є трек, на основі темпу, ритму і т.п, де 0 це найменш танцювальний, а 1 найбільш;
- energy – параметр від 0 до 1, що відображає інтенсивність треку. Високоенергічні треки є швидкими та гучними;
- loudness – параметр який відображає загальну гучність треку в децибелах. Значення усереднюються по всій довжині треку, типово коливається від -60 до 0 дБ;
- instrumentalness – параметр що оцінює ймовірність відсутності вокалу в треці. В диапазоні від 0 до 1, чим ближче до одиниці, тим більша ймовірність що в треці відсутній вокал;
- key – тональність треку, що визначає основну ноту від 0 до 11. Де 0 це C, а 11 B;
- tempo – BPM композиції, тобто її темп в ударах за хвилину.

Та багато інших, за допомогою цих ознак ШІ який використовується на платформі Spotify, робить ревалентні рекомендації щодо треків що можна побачити на рисунку 1.5, де у рекомендаціях знаходяться лише аніме опенінги, та J-POP музика, що дійсно підходить за тематикою цього плейліста.

Наступною популярною платформою є YouTube Music, вона є не такою поширеною як Spotify, проте на відміну від попередньої платформи, цей сервіс як би не є стрімінгом музики, а саме стрімінгом музикальних відео у фоні. Як би YouTube Music, є тим самим YouTube проте в якому є тільки, музикальні відео. Ця платформа також вміє надавати користувачу рекомендації за музикою в списку відтворення при його створенні, чи надає можливість слухати «Мікс» автоматичний список відтворення створений YouTube за допомогою алгоритмів які аналізують смаки користувача та музику яку він часто прослуховує, і створює на основі цього ці самі списки відтворення. Часто в них з’являється музика яку слухає користувач, та також постійно додається нова, тобто створюється «нескінченна» черга з музики, при тому на відміну від

Spotify, рекомендації YouTube досить часто проштовхують маловідомих авторів, чи навіть можуть запропонувати користувачу музику, жанр якої користувач не слухав ні разу. Алгоритм аналізує як саме користувач себе поводить під час прослухування таких треків чи пропускає їх користувач чи ні і т.д. Також на поведінку алгоритму впливають вподобайки, та дизлайки на конкретні відео.

Як видно з рисунку 1.6 прослуховуючи трек Styx Helix який є японською поп музикою, чи опенінгом, система рекомендації YouTube пропонує схожі варіанти на ті що були у Spotify.

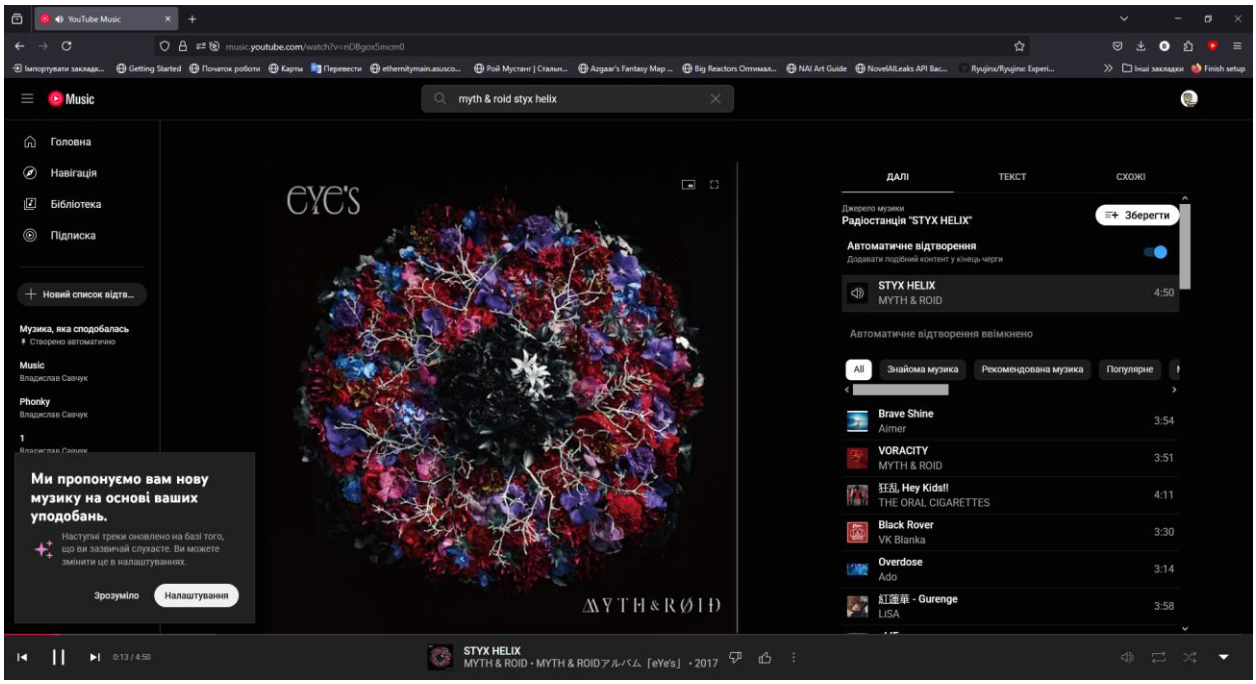

Рисунок 1.6 – Вікно із відкритим плейлистом YouTube Music

Що відображає також дуже ефективну роботу алгоритму. Також навідміну від Spotify, YouTube Music використовує нативну інтеграцію із Google, тому що він є продуктом цієї корпорації, що позитивно впливає на зручність використання цього сервісу на Android пристроях, які в більшості вимагають використання хоча б одного Google акаунту у системі, також YouTube Music не потрібно встановлювати окремо, оскільки додаток YouTube дає можливість використовувати весь функціонал в стандартному додатку.

І також важливою особливістю є те що на YouTube може публіковати свою музику хто завгодно, навідміну від Spotify. Усе що необхідно це просто завантажити свій трек через звичайну сторінку завантаження відео.

Shazam на відміну від двох попередніх сервісів не є стрімінговим сервісом, проте дуже тісно пов’язаний із темою використання ШІ на музичних платформах. Це дуже популярний сервіс із розпізнавання музики яку користувач чує біля себе в будь-який момент часу за допомогою додатку не телефоні, чи на ПК

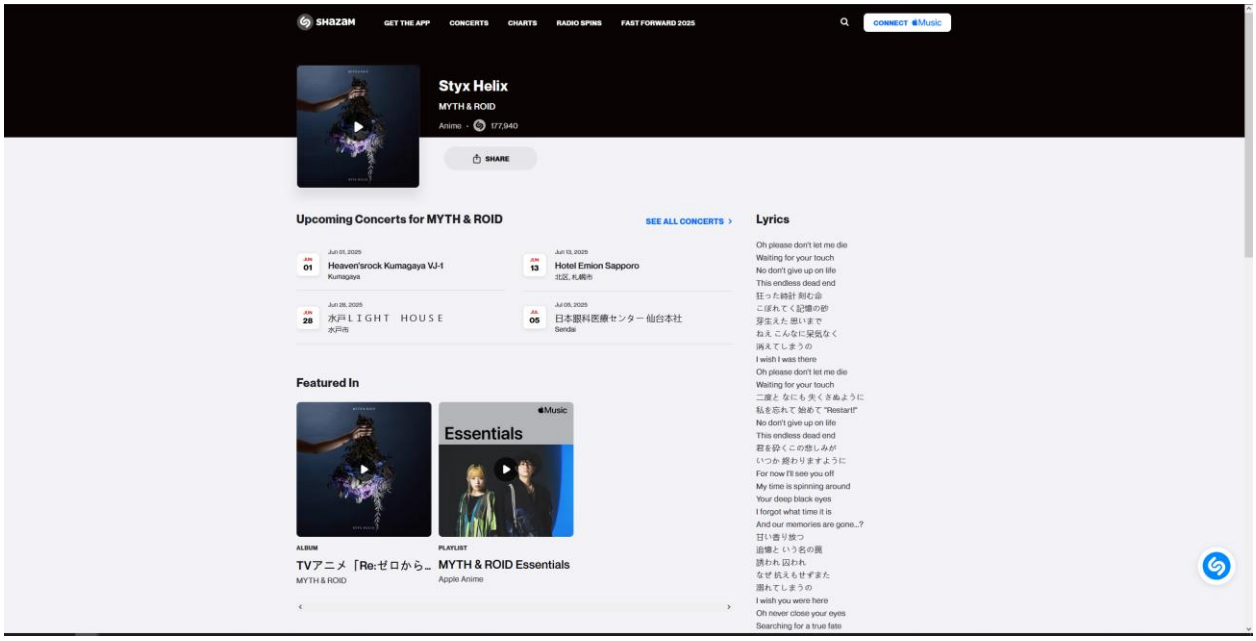

Рисунок 1.7 – Вікно із результатом пошуку за треком Shazam

Тестування проводилося на тому же самому треці, і як видно із результату пошуку на рисунку 1.7 він був визначений корректно, Shazam, використовує аудіо відбиток, базу даних яка містить у собі хеші різних треків, додаток записує невеликий семпл, тривалістю 5-15 секунд, перетворює його на хеш, та порівнює його із своєю базою даних та повертає той який збігається найбільше. Також у Shazam є розклад концертів, текст пісні, а також треки які схожі на цей трек за хешем. Також важливо помітити що навідміну від систем рекомендації у YouTube Music, та Spotify, система у Shazam виділяє схожими треки частіше від однієї і тієї же групи MYTH & ROID, що є логічним оскільки

стилі треків і як висновок їх аудіо відбиток будуть схожими. Системи розпізнавання жанрів сервіс не має, оскільки жанри містяться у базі даних.

SoundCloud популярна музична платформа для незалежних авторів, тут загалом є ліцензована музика захищена авторським правом, так і без авторського права. Також тут є можливість придбати розширені інструменти для творців, які допомогають аналізувати смаки аудиторії тощо. Сервіс також використовує інструментарій штучного інтелекта, для автоматичного визначення жанру музики, рекомендації плейлистів тощо.

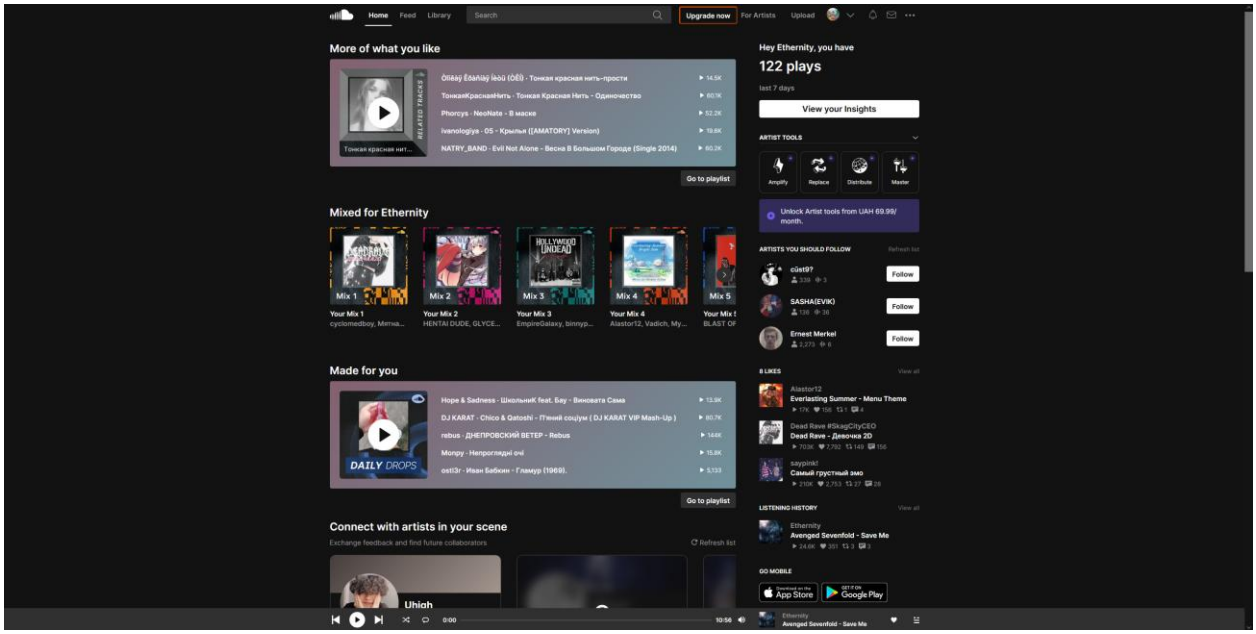

Рисунок 1.8 – Вікно із головною сторінкою SoundCloud

На рисунку 1.8 зображена головна сторінка сервісу, на якому користувач увійшов у свій акаунт, маючи хоча б якусь історію прослуховувань SoundCloud, подібно до YouTube Music запропонує нескінченну чергу із музики за вашими вподобаннями. Також тут є плейлісти за жанрами та «радіостанції» до яких можуть доєднатися люди задля сумісного прослуховування однієї ж і тої черги прослуховування. Для користувачів SoundCloud є повністю безкоштовним і без реклами на відміну від Spotify, та YouTube Music.

Проте варто зазначити що через свою модель безкоштовного використання, якість класифікації жанрів чи підбірки музики гірше ніж у

платформ приведених вище, проте не на багато. Усе ж сучасні імплементації розрізнення жанрів можуть бути дуже ефективними та при цьому не затратними у ресурсах.

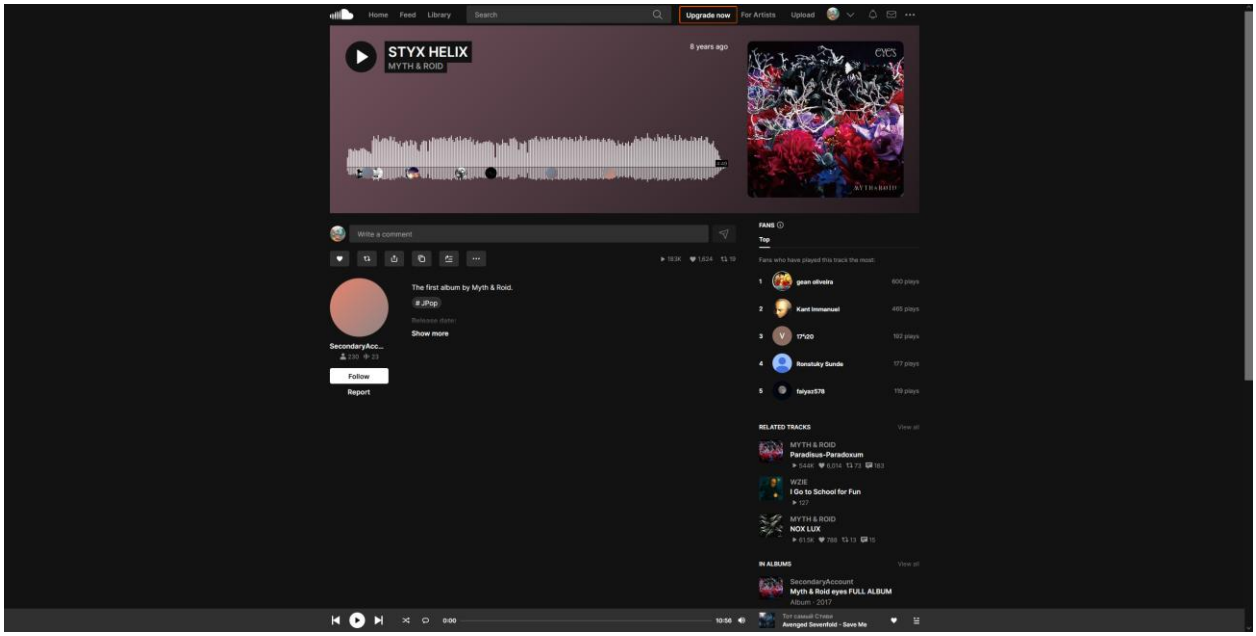

Рисунок 1.9 – Вікно із головною сторінкою SoundCloud

На рисунку 1.9 зображена сторінка треку який використовувався при аналітиці усіх існуючих додатків, поглянувши направо видно пов’язані треки, і проблема точно така ж як і у Shazam рекомендації найчастіше надаються за автором. Також система SoundCloud відмітила цей трек як J-Pop, що є точним результатом.

Підсумовуючи усе вище описан, розробка системи яка зможе розрізняти жанри, шукати музику, вирізняти емоційність музики, та робити рекомендацію є більш ніж актуальною. Жодний з вище приведених сервісів не надає послуги із класифікації жанру без завантаження треку на сайт, Shazam не має у собі можливості додавання нових маловідомих треків що може викликати проблеми, якщо база не встигла оновитися на момент пошуку. Також варто відмітити що хоча система рекомендації у Spotify та YouTube Music є досить високоточними проте є питання щодо використання ресурсів при застосуванні

таких систем. Тобто розроблена система повинна бути досить оптимізована щоб працювати на адекватному апаратному забезпеченні.

1.3 Цілі та завдання кваліфікаційної роботи

Актуальність обраної тематики зумовлює необхідність формування цілей та завдань. Основною метою даної кваліфікаційної роботи є розробка ефективного, оптимізованого програмного забезпечення для аналізу аудіоданих із подальшою класифікацією, пошуком та рекомендацій музичних творів. Для досягнення цієї мети необхідно виконати деякі завдання. Підсумовуючи мета роботи полягає у створенні програмного рішення, яке б було ефективне при використанні на музичних платформах, використовуючи сучасні методи машиного навчання. Для реалізації необхідно виконати такі завдання:

- провести детальний теоретичний аналіз – виконати критичний аналіз літератури за темою, і зробити висновки, які методи будуть найбільш ефетивними при вирішенні даного типу завдання;
- розглянути приклади існуючих сервісів – Виокримити при аналізі цих сервісів інформацію стосовно цієї теми, виявити переваги та недоліки, та використати цю інформацію при побудові своєї системи;
- дослідити методи цифрової обробки аудіофайлів – Дізнатися як саме можна оброблювати аудіосигнал за допомогою мов програмування, дізнатися детальніше про аудіоознаки та їх поняття, дізнатися про їх екстракцію із аудіофайлу;
- обрати мову програмування та середовище розробки – Дослідити питання щодо мови програмування, обрати ту яка найкраще підходить для написання високопродуктивного додатку із використанням ШІ, які фреймворки є для виконання завдань, та знайти до них документацію;
- знайти підготовлений датасет за темою, або ж створити свій датасет – На цьому етапі необхідно знайти датасет який буде відповідати поставленому завданню, бажано щоб він мав збалансованість за

класами, а таком гарну якість аудіо яка забезпечить якісну класифікацію;

- створення модуля для препроцесінгу даних – Перетворення аудіо на ознаки, або на спектрограми, загалом на дані для подальшої обробки за допомогою методів машиного навчання, за допомогою спеціальних бібліотек таких як librosa для Python;
- підготувати можливі моделі машиного навчання – Застосування класичних методів для класифікації та схожості треків таких як k-NN, Random Forest та ін. а також застосування глибоких нейроних мереж таких як CNN, чи CRNN;
- провести серію експерементів із моделями – За результатами експерементів обрати найкращу модель за точністю за іншими метриками якості, балансуючи при цьому між точністю, складністю моделі, та швидкістю;
- розробити інтерфейс користувача – Розробити інтерфейс який буде ергономічним та простим для використання навчанних моделей для звичайних користувачів.

2 РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ

2.1 Розробка та докладний опис функціональної схеми програми

Перед тим як перейти до розробки програмного забезпечення та почати писати код, необхідно побудувати логічну архітектуру із використанням різних типів схем таких як IDEF0 чи діаграма послідовности (Sequence Diagram). Перш ніж перейти до побудов схем та їх детального розгляду, варто зазначити що сучасні алгоритми машиного навчання дозволяють виділяти приховані паттерни та інші характеристики аудіо. Задачі класифікації жанрів, рекомандаціїні системи, дуже важно розв’язати за допомогою традиційних методів. Нижче будуть наведені діаграми, які демонструють поетапну декомпозицію системи аналізу аудіоконтенту яку для зручності було названо «Juketils». Кожна діаграма розкриває певний рівень абстракції, що дозволяє зрозуміти принципи роботи системи та окремі її функцій.

На рисунку 2.1 зображена діаграма із загальним виглядом системи, її входами та виходами, системами керування, та клієнтом який взаємодіє із системою

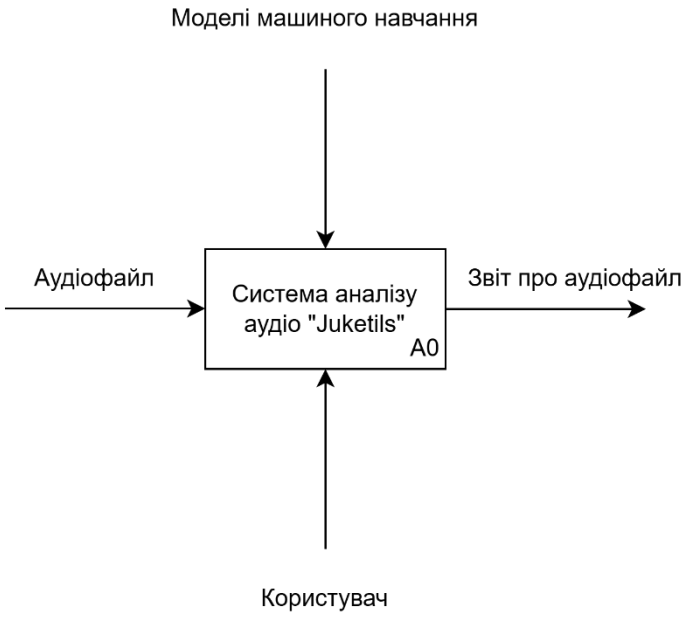

Рисунок 2.1 – Додаток «Juketils» нульовий рівень декомпозиції

Система «Juketils» має один вхід – аудіофайл, який надходить для аналізу аудіо та один вихід – звіт про проаналізований аудіофайл, керування аналізом відбувається за допомогою моделей машинного навчання, за результатом яких і формується звіт. Клієнтом що використовує систему є користувач додатку, він взаємодію з інтерфейсом програми, та ініцює процеси обробки даних.

На рисунку 2.2 зображено діаграма декомпозиції першого рівня системи, яка розкриває внутрішню структуру блоку A0, яка демонструє послідовність обробки даних у системі. Система містить у собі чотири основних функції, які послідовно оброблюють аудіофайл та формують результат.

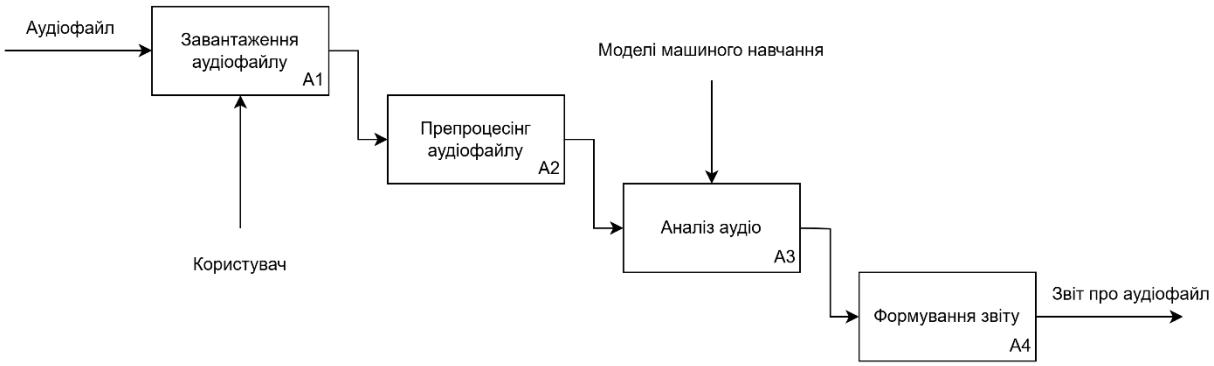

Рисунок 2.2 – Додаток «Juketils» перший рівень декомпозиції

Перший блок відповідає за завантаження аудіофайлу від користувача та його валідацію, та його передачі на препроцесінг. На цьому етапі перевіряється тип файлу, формат файлу, та його розмір, така базова перевірка дозволить уникнути проблем та помилок на наступних етапах обробки. Користувач при завантаженні аудіо буде можливість обрати між завантаженням свого аудіофайлу у форматі .mp3 та ін. та записом на пряму із мікрофона користувача при наявності відповідних дозволів у браузері.

Наступна функція відповідає за препроцесінг аудіо, при цьому виконуються різного типу обробки, такі як нарізання аудіо на стандартизовані за тривалістю відрізки, нормалізація гучності, приведення до однієї частоти дискретизації.

Аналіз аудіо, в цьому блоці виконується глибокий аналіз аудіо відрізків, та виокремлених аудіоознак, за допомогою навчених моделей машиного навчання. Після чого відбувається оформлення результатів за допомогою відповідної функції у архітектурі додатку.

На рисунку 2.3 зображено декомпозицію блоку А2, він містить у собі чотири блока функції.

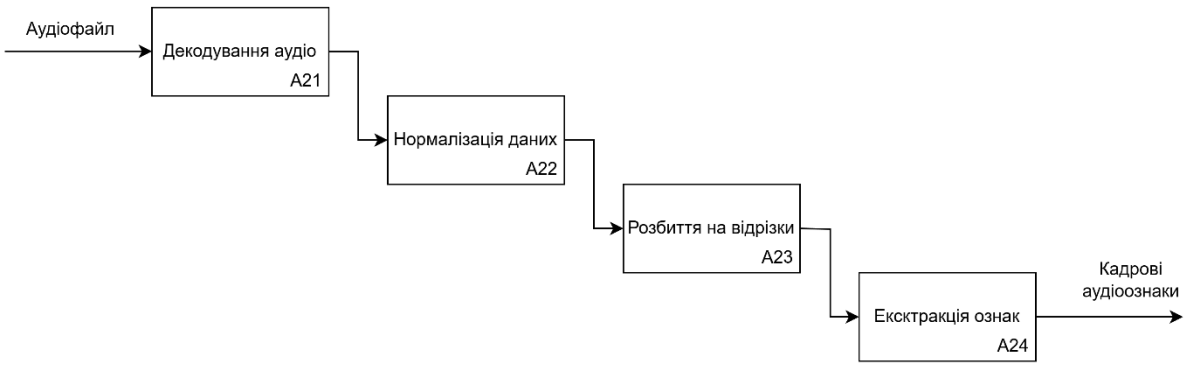

Рисунок 2.3 – Додаток «Juketils» декомпозиція блоку А2

Діаграма показує що препроцесінгу аудіо розбивається на чотири підфункції. Перша функція це декодування аудіо, тобто перетворення аудіосигналу з файлу .mp3 формату у вектор значень амплітуд із роздільною здатністю заданою частотою дискритизації. Цей процес дозволяє привести тип даних до того який можна оброблювати за допомогою спеціальних бібліотек. Наступною функцією є нормалізація даних, це важливий процес який дозволяє виключити вплив розкиду даних на процес навчання моделей, приводячи значення до стандартного діапазону. Наступним кроком є розбиття на відрізки стандартизованої довжини, що дозволить виконувати більш точний пошук за допомогою такого підходу. Ця функція необхідна для того щоб передавати до моделі очікуваний вектор із заданою розмірністю. Завершальною функцією аналізу аудіо є екстракція ознак, таких як, мел-кепстральні характеристики, хроматичні характеристики, спектральний центроїд тощо. За цими ознаками буде виконуватися подальший аналіз аудіо.

Також варто розглянути детально процес аналізу аудіо, у системі, декомпозиція якого зображена на рисунку 2.4.

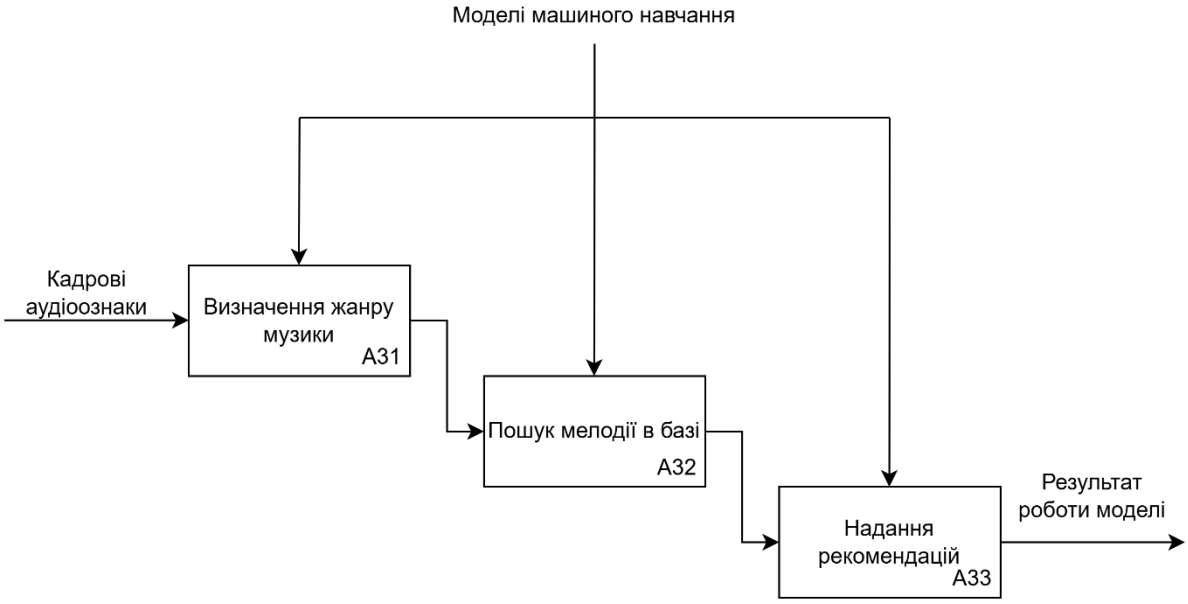

Рисунок 2.4 – Додаток «Juketils» декомпозиція блоку А3

Для того щоб сформувати звіт моделі машинного навчання будуть оброблювати аудіофайл за такими критеріями: визначення жанру музики, пошук мелодії в базі та надання рекомендацій.

Визначення жанру музики відбувається за допомогою раніше виокремлених аудіоознак у файлі, та голосування за варіант жанру, чим триваліше буде аудіо тим вища буде точність такого аналізу.

Пошук мелодії в базі відбувається також за допомогою виокремлених аудіоознак, чи за допомогою спектрограм та їх піків. Та порівнянь відстаней у багатомірному просторі, за допомогою різних метрик таких як косинусна відстань, евклідова та інші метрики.

Надання рекомендацій робота цього блоку схожа на роботу пошуку мелодій оскільки, дуже вірогідно що схожа мелодія сподобається користувачу, проте можна враховувати не тільки це, а й емоційний стан музики, час програвання мелодії та інші не очевидні фактори які можуть надавати високоревалентні рекомендації для користувачів сервісу.

На рисунку 2.5 зображено діаграму послідовності сервісу, яка дозволяє зрозуміти послідовність взаємодій між компонентами системи. Головними компонентами системи є: користувач, інтерфейсна частина, API, та модель машиного навчання.

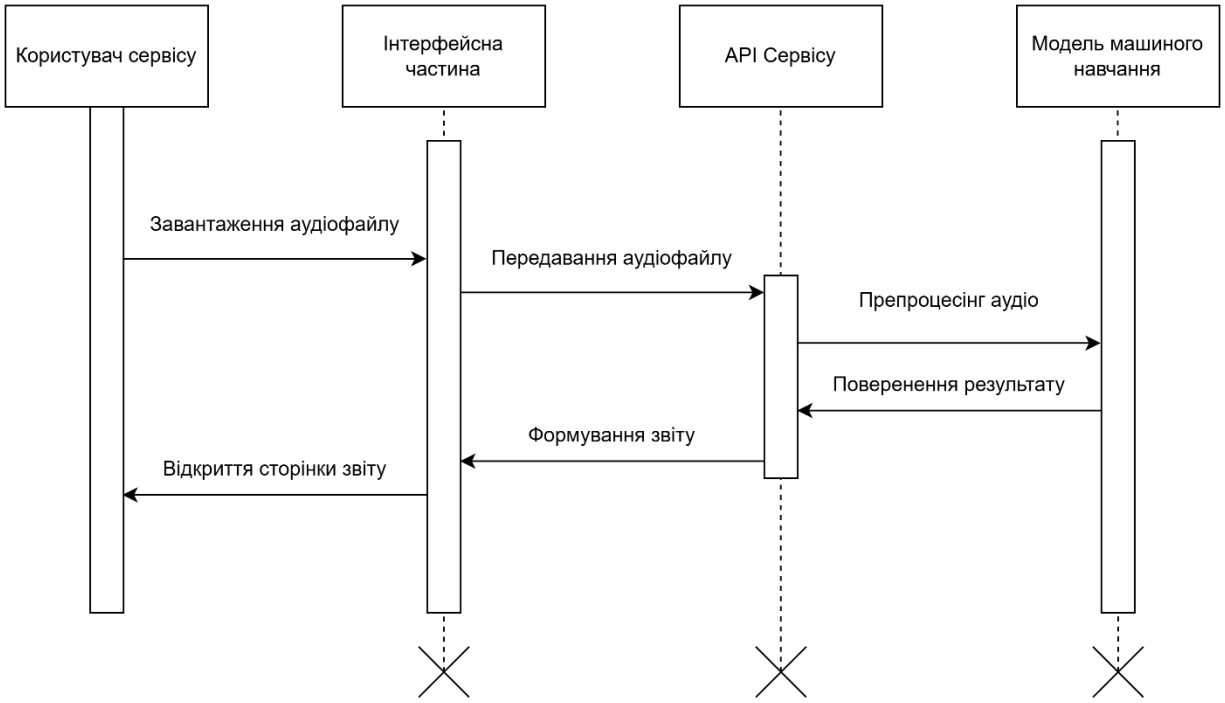

Рисунок 2.5 – Додаток «Juketils» діаграма послідовності

Процес починається із завантаження аудіофайлу до сервісу цей процес ініціює користувач запускаючи ланцюг певних дій між компонентами системи, аудіофайл передається до API, де виконується препроцесінг після чого дані передаються до моделей які сформують результат та повернуть його до API, той сформує звіт, а інтерфейсна частина виведе нову сторінку із детальним звітом.

Також на схемі відмічено життєвий цикл компоненті, що демонструє що з’єднання з API завершується при виконанні завдання, при тому що модель працює завжди.

Наведена архітектура є дуже гнучкою та дозволяя легко її корректувати в процесі розробки оскільки усі програмні компоненти з’єднанні послідовно, та в той час незалежні одне від одного окрім першого рівня деталізації. Тобто

якщо прибрати систему рекомендацій, програма також буде працювати, що наприклад може бути корисно якщо користувачу не потрібен такий тип аналізу його аудіофайлу. Що також значно скоротить час обчислень, також це гарно позначається на можливостях масштабування сервісу для додавання нових функцій.

2.2 Розробка та докладний опис алгоритму роботи програми

Щоб виконати завдання такого типу необхідно дослідити існуючі алгоритими та розробити той, який зможе виконати поставлені задачі. Почнемо із виділення аудіоознак, цю задачу можна розв’язати різними шляхами: екстрактувати конкретні аудіоознаки за допомогою librosa, або ж використати конвуляційну (згорткову) нейронну мережу, яка зможе сама виокремити деякі абстракні ознаки, які можна буде використовувати для задачі класифікації жанрів чи рекомендаційної системи. Будь який з цих підходів є ефективним.

Традиційний метод за допомогою екстракції librosa дозволяє витягувати заздалегідь відомі та дослідженні харктеристики, такі як: мел-кепстральні характеристики, хроматограма,спектральний центроїд, та інші. Це метод має перевагу в інтерпретованості цих ознак, дозволяючи їх аналізувати не тільки за допомогою машиного навчання, а навіть вручну.

Підхід за допомогою нейроних мереж дозволяє виявити приховані патерни в даних, абстрактні ознаки, також при збільшенні даних точність класифікації зростає, проте варто зазначити що такий метод потребує навчання нейромережі на певній кількості даних, що іноді не є можливим.

Для цієї задачі та теми роботи, де відбувається робота із великою кількістю аудіофайлів теоретично краще підходить метод із використанням нейроних мереж, проте необхідно перевірити це практично

На рисунку 2.6 зображено приклад мінімальної конвуляційної мережі для виокремлення ознак, вона складається з таких частин як вхідний

шар(Input), конвуляційний шар(Conv), шар пулінгу (Pooling), та шар вирівнювання (Flatten).

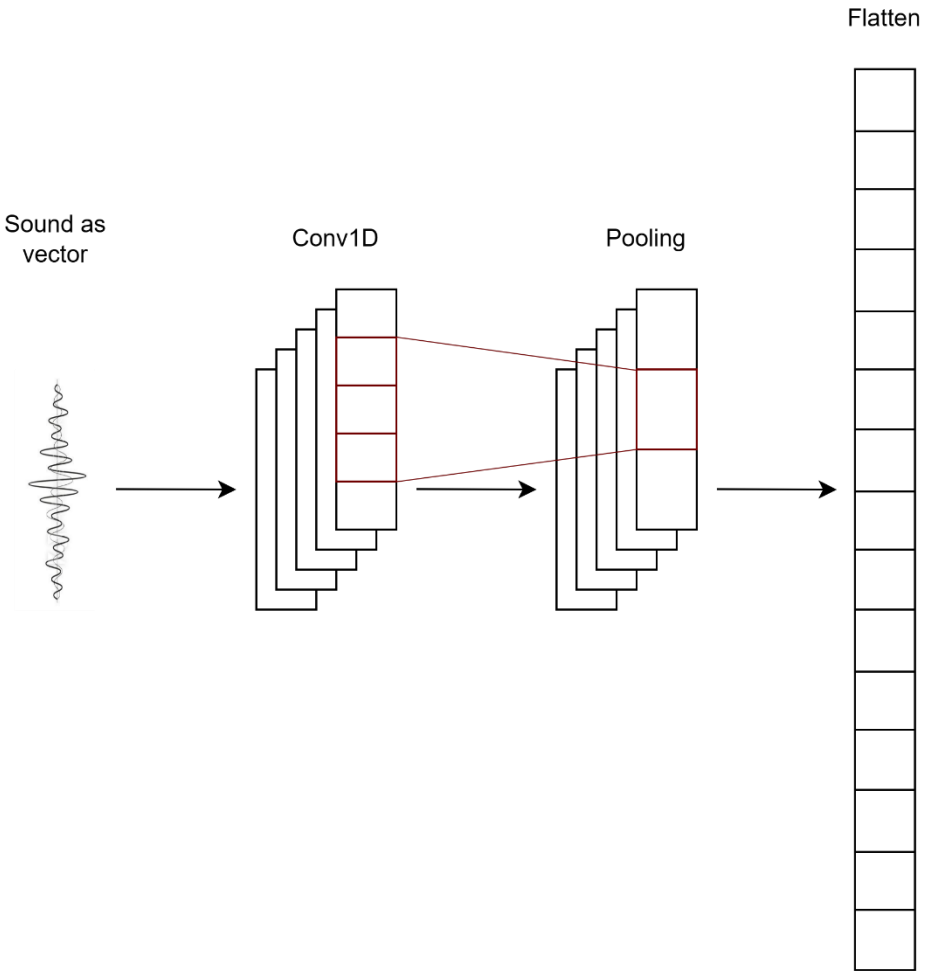

Рисунок 2.6 – Приклад конвуляційної мережі для створення фіч, із аудіовектора

На цій конкретній схемі відображено такі шари, Sound as Vector, цей шар є вхідним до, Conv1D, конвуляційного одновимірного шару, який формує абстрактні фільтри (ознаки), потім відбувається об’єднання у Pooling шарі, в зазначеному прикладі відбувається об’єднання трьох ознак до однієї, це дозволяє зменшувати розмірність даних та виокремлювати найбільш впливові елементи у послідовностях, цей шар може працювати в трьох режимах максимальний, середній та мінімальний, та приймає відповідне значення з цих елементів, після чого відбувається розгортання Pooling шару до звичайного

вектору, який можна під’єднати далі до мережі класифікатора чи регресора в залежності від задачі.

На рисунку 2.7 зображено типову повнозв’язну нейромережу класифікатор якій на вхід подаються або абстрактні ознаки які були отримані за допомогою згорткової мережі, або ж ознаки із бібліотеки librosa, кількість вхідних нейронів дорівнює кількості ознак, тому важливо заздалегідь знати їх кількість, кількість нейронів у прихованому шарі може бути довільною і різнитися від задачі до задачі, як і кількість самих прихованих шарів, а кількість виходів також залежить від задачі, наприклад при класифікації восьми жанрів їх буде вісім, при класифікації тисячі об’єктів, тисяча.

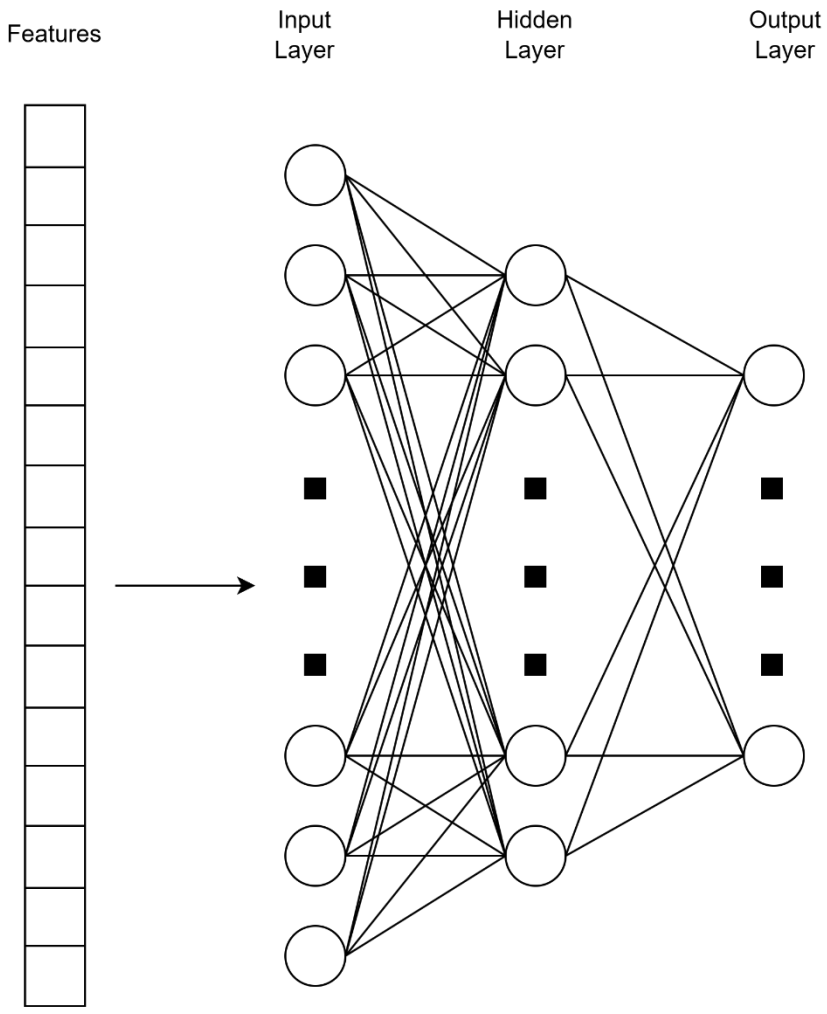

Рисунок 2.7 – Приклад мережі для класифікації за фічами

Такі типові повнозв'язні нейронні мережі для вирішення задач класифікації, досить часто використовують для обробки даних після конвуляційних мереж, що дозволяє поєднати переваги цих двох архітектур, а саме автоматичне виділення абстрактних ознак, з ефективною класифікацією за допомогою багатошарової нейромережі. Такий підхід особливо ефективний при роботі з аудіоданими чи з зображеннями, що підходить для вирішення задачі класифікації жанрів.

Для системи рекомендацій чи пошуку схожих пісень можна використовувати метод k-NN, або k-найближчих сусідів, цей метод полягає в тому що точки даних розміщуються в просторі та відносяться до деяких груп. На рисунку 2.8 зображено принцип роботи алгоритму k-NN. Зліва зображено нову точку поки невідомого класу, позначену сірим та знаком «?», а праве зображення відображає процес віднесення точки до відповідного класу.

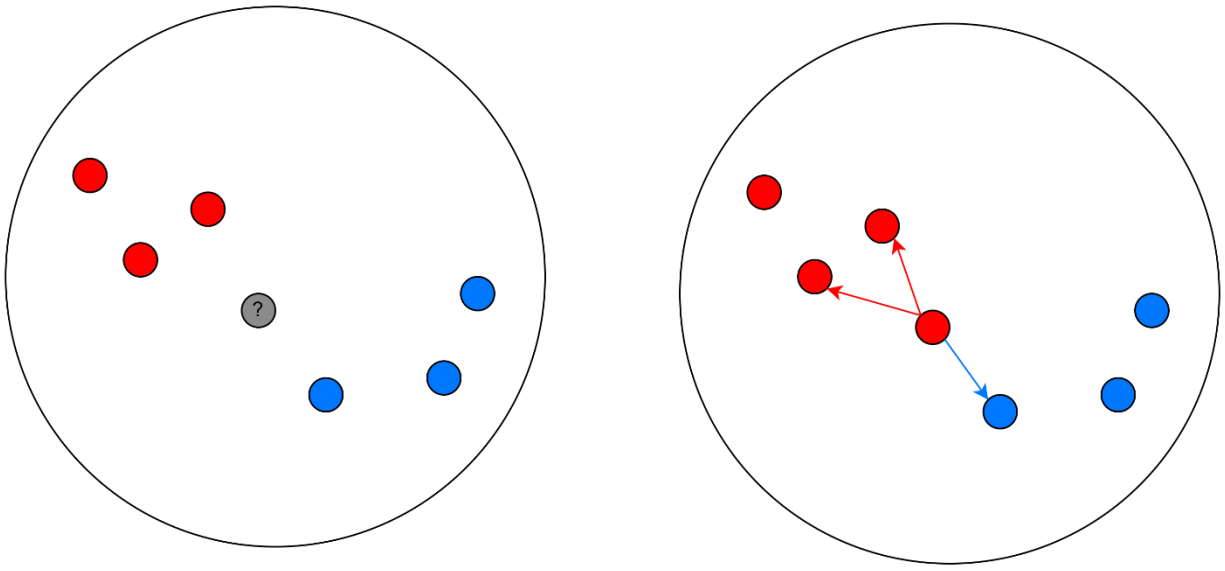

Рисунок 2.8 – Приклад роботи k-NN

Суть методу полягає в тому що клас нової точки у просторі визначається на основі k найближчих точок до неї, та приймає рішення на основі більшості голосів, тому варто використовувати або ж зважений k-NN, або ж непарне k. Також важливо використовувати для k-NN дані з найменш можливою

розмірністю, оскільки цей метод вразливий до «прокляття розмірності», а також із кількістю даних зростає час обчислення класу для нової точки, тому що метод вираховує відстані до усих точок, проте цього можна уникнути за допомогою методів оптимізації цього методу. Із переваг методу є те що він простий та інтуітивний, також він не потребує попереднього навчання моделі, та може адаптуватися до нових даних, просто додаючи їх до існуючого набору. Метриками для цього методу зазвичай використовуються евклідову відстань. У цьому алгоритмі єдиним параметром який необхідно підібрати є значення k. Його оптимальне значення значною мірою впливає на якість алгоритму.

Також для пошуку пісень можна використати інший алгоритм, заснований на методі пошуку за допомогою виокремлення піків на частотах у аудіофрагменті, та пошуку за ними за хешем, цей алгоритм не відноситься до машинного навчання, питання точності в порівнянні із k-NN необхідно дослідити детальніше, проте при великої кількості даних такий алгоритм буде працювати набагато швидше.

2.3 Розробка інтерфейсу програми

При розробці інтефейсу програми необхідно враховувати те що він повинен бути простим ергономічним, та зрозумілим для необізнаного користувача. Головною метою при побудові інтерфейса є забезпечення інтуітивної взаємодії між користувачем та розробленою системою, мінімізуючи при цьому кількість кроків, необхідних для виконання операцій у системі. При такій постановці завдання, резонним є реалізувати інтерфейс, у вигляді односторінкового веб-додатку, який буде відображати візуально стан додатку. Інтерфейс є зручною обгорткою для взаємодії із розробленними моделями. Дивлячись на рисунок 2.9 можна побачити що інтерфейс дуже простий, необхідні дії від користувача зведені до мінімуму. Структура інтерфейсу побудована за принципом вертикального розташування функціональних блоків, в порядку логічної послідовності дій користувача. Програма має виконувати одну дію, формувати цілісний звіт за аудіофайлом,

та видавати його візуальне представлення користувачу. Головна сторінка додатку буде містити у собі логотип, який буде не тільки декоративним рішенням, а й додатковою можливість скинути додаток в початкове положення. Банерна частина буде представляти собою декоративний елемент.

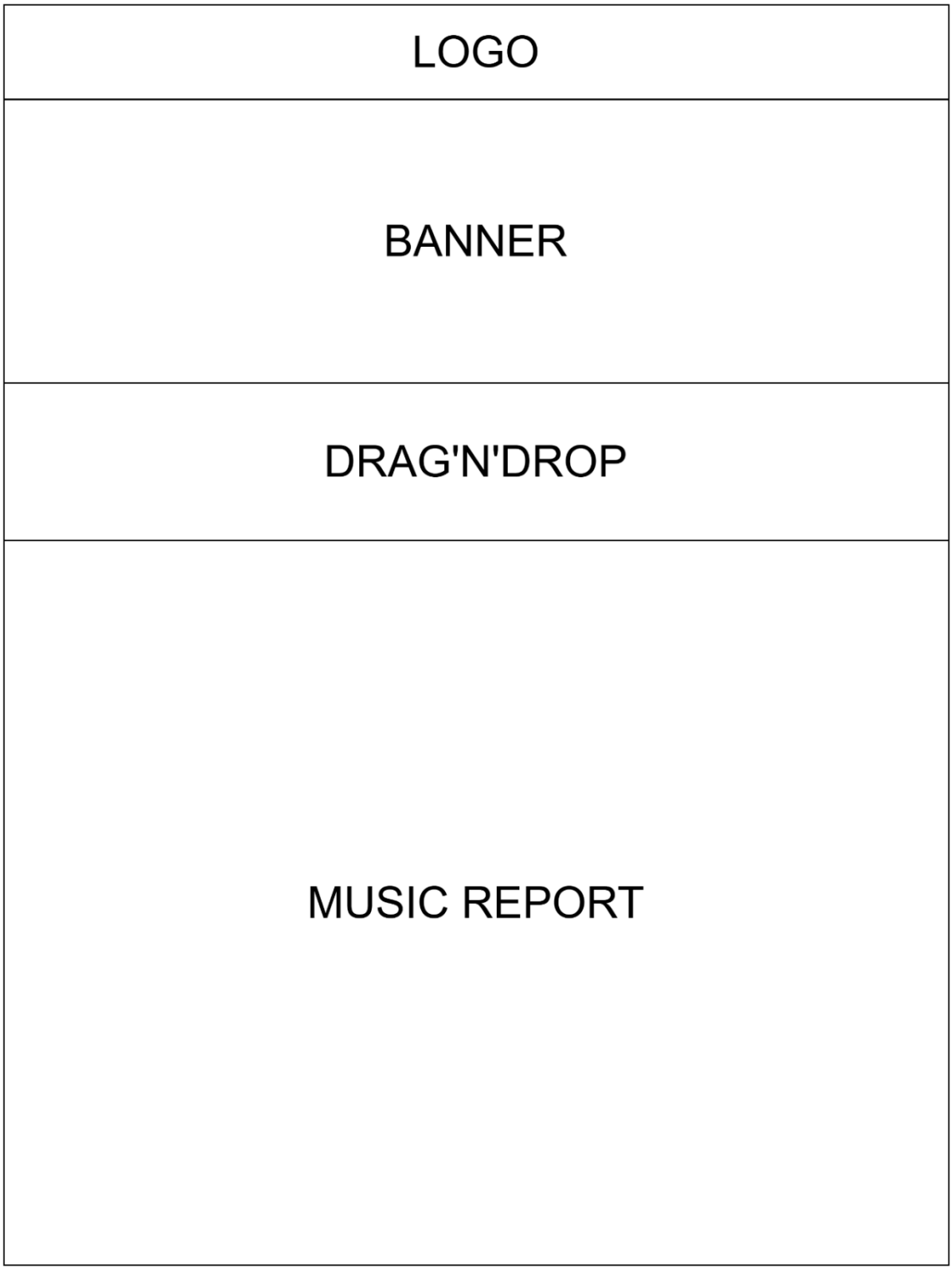

Рисунок 2.9 – Схематичне відображення головної сторінки додатку

Drag’n’Drop цей елемент дає змогу користувачу можливість відправити заздалегідь зроблений аудіофайл на рисунку 2.10 зображено детальне схематичне зображення цього елементу. Як видно він має текст опис як користуватися цим елементом, що також покращує розуміння користувача при користуванні цим додатком. При завантаженні, зона починає відображати назву завантаженного файлу та кнопку скасування завантаження файлу.

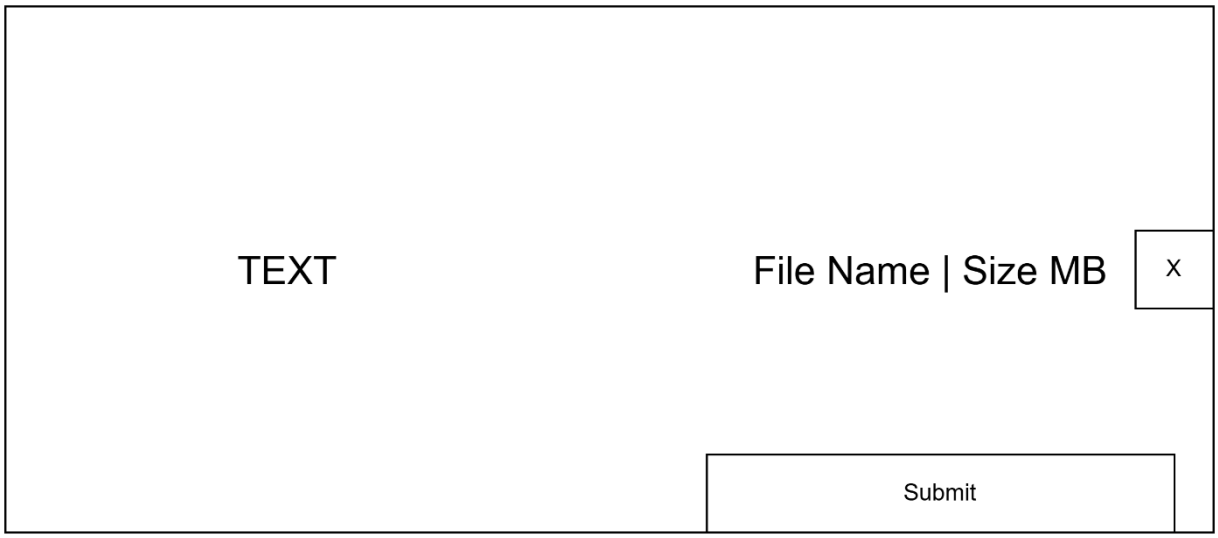

Рисунок 2.10 – Схема елементу «Drag ‘n’ Drop»

Функціональність цього елементу має бути реалізована максимально інтерактивно та зрозуміло для користувача. Коли користувач перетягує файл над цією областю, вона візуально реагує змінює свій колір та текст сигналізуючи що елемент очікує на файл який тримає користувач своїм курсором. Або ж про сигналізувати про помилку відповідним повідомленням, наприклад що користувач завантажив не той формат файлу, завеликий розмір, чи помилка серверу при обробці аудіо. Після завантаження та відправки аудіо на сервер сторінка має надавати зворотній зв’язок, у відображенні статусу виконання аналізу написом «Завантажується» тощо, на місці блоку зі звітом після аналізу.

Нижче блоку «Drag ‘n’ Drop», розташовується блок із звітом за аудіофайлом, поки не виконується аналіз повинен відображатися у якості прикладу блоків які користувач отримає після аналізу, та їх описом. Після виконання аналізу, цей блок змінить свій вигляд на той що схематично зображено на рисунку 2.11

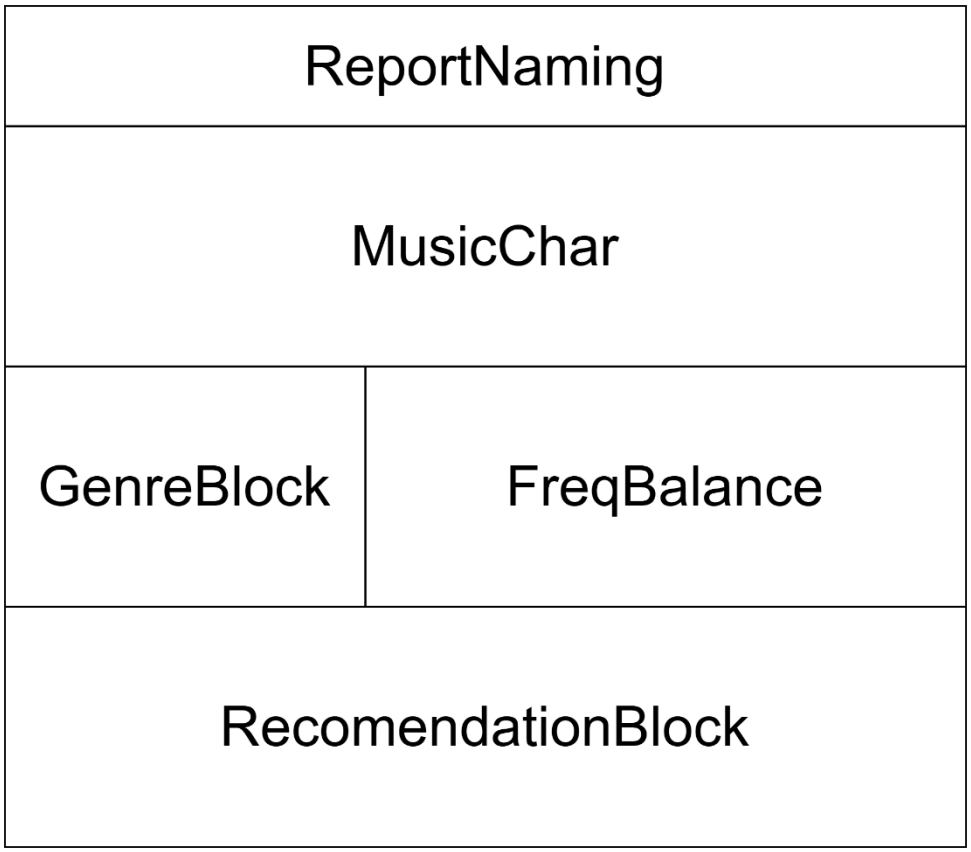

Рисунок 2.11 – Схема елементу звіту

Звіт містить у собі результат аналізу аудіофайлу. Блок із жанровою класифікацією, який відображає настрій треку, його BPM, та тоніку треку. Блок із жанровою класифікацією, що може бути представлений у вигляді списку жанрів, із виділеним основним жанром, і іншими можливими варіантами, також із тим на скільки відсотків впевнена нейромережа, що може бути корисним при прийняті рішень щодо просування треків. Блок із частотним балансом відображає розподіл низьких, середній, високих частот що дає представлення про характеристики цього треку, і полегшує його подальший

мастерінг перед публікацією. Блок рекомендацій буде представлений у вигляді списку із трьох елементів, ці треки можуть бути корисними для користувачів які хочуть знайти треки які сподобаються йому на основі завантаженого треку, або ж для музикантів які займаються пошуком надхнення.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз обраної середи програмування

Перед початком розробки програмного коду, необхідно обрати оптимальний під поставлені задачі інструментарій. Для реалізації програмного забезпечення необхідно: побудувати інтерфейс, побудувати та навчити модель, розробити API.

Для побудови інтерфейсу було обрано фреймворк Next.js[13], як зручний інструмент для побудови single-page application (SPA), який і дозволить забезпечити максимальну інтуітивну взаємодію із юзером. На відміну від HTML, Next.js, використовує багаторазові компоненти React[14] замість дублікації коду що покращує його читабільність та розуміння. Також React-компоненти підтримують систему хуків, що дозволяє динамічно змінювати інтерфейс без перезавантажень сторінки. Використання хуків при написанні коду:

```
const [analysisResult, setAnalysisResult] = useState(null)
const [isAnalysisComplete, setIsAnalysisComplete] =
useState(false)
const [analysisError, setAnalysisError] = useState(null)
const [isLoading, setIsLoading] = useState(false)

const handleAnalysisStart = () => {
  setIsLoading(true)
}

const handleAnalysisComplete = (result) => {
  setAnalysisResult(result)
  setIsAnalysisComplete(true)
  setAnalysisError(null)
  setIsLoading(false)
}

const handleAnalysisError = (error) => {
  setAnalysisError(error)
  setAnalysisResult(null)
  setIsAnalysisComplete(false)
  setIsLoading(false)
}

const resetAnalysis = () => {
  setAnalysisResult(null)
```

```
    setIsAnalysisComplete(false)
    setAnalysisError(null)
    setIsLoading(false)
  }

  return (
    <>
      <Waves/>
      <Separator/>
      <DragDrop
        onAnalysisStart={handleAnalysisStart}
        onAnalysisComplete={handleAnalysisComplete}
        onAnalysisError={handleAnalysisError}
        onReset={resetAnalysis}
        isAnalysisComplete={isAnalysisComplete}
      />
    </>
```

В цьому коді на головній сторінці розроблюванного додатку використовуються хуки які контролюють стан аналізу і відповідно до зміни цього стану змінюється react-компонент DragDrop, який відповідає за завантаження та обробку аудіофайлів. Next.js також надає підтримку серверного рендеру (SSR) та статичної генерації сторінок (SSG), що підвищує швидкість завантаження. Для стилізації інтерфейсу було обрано розширення мови CSS, SCSS, він був обраний через те що надає значно більше можливостей у порівнянні із звичайним CSS. SCSS дозволяє використання змінних, для кольорів, розмірів, тощо. Це дозволяє наприклад швидко змінити тему у всьому додатку наприклад змінивши кольори у файлі colors.scss:

```
$background: #0B0F14;
$foreground: #F0F4F8;
$line_color: #1A2A3A;
$accent_color: #00F0FF;
$second_color: #9B51E0;
$hints: #8A99A5;
$error: #FF4444;
```

Для побудови та навчання машиного навчання було обрано мову програмування Python, із використання бібліотеки keras. Python є стандартною мовою у галузі машиного навчання, завдяки активній спільноті та великої кількості потужних бібліотек для роботи із машиним навчанням. Основними перевагами keras є його інтуїтивний синтаксис. Бібліотека надає можливість використувати різні типи шарів нейронних мереж, згорткові, рекурентні та

повнозв’язні, що дозволяє створювати архітектури, будь-якого рівня складності для вирішення різноманітних задач. Програмна реалізація простої нейроної мережі класифікатора, виглядає так:

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3),
activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3),
activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

Тобто за допомогою keras, можна зручно створювати нейромережі логічно розташовуючи шари через зручний інтерфейс.

Для обробки аудіо доцільно використовувати бібліотеку librosa, вона надає широкий інструментарій для отримання даних для аналізу звуку. Вона надає можливість отримувати різні музичні характеристики, такі як мел-кепстральні коефіцієнти (MFCC), спектральні характеристики, темп, та інші корисні характеристики. Також для обробки даних та їх відображення використовуються бібліотеки numpy, pandas, та matplotlib, іноді seaborn. Ці бібліотеки є надзвичайно потужним інструментом для роботи із великими даними та їх аналізом, оскільки дозволяють швидко маніпулювати із масивами замість повільних списків у python, продивлятися csv-файлы у вигляді зручних таблиць та виконувати над ними препроцесінг, та створювати високоякісні візуалізації.

Комбінація цих бібліотек забезпечують високу сумісність між собою що дозволяє легше та ефективніше досліджувати дані із датасетів, а також створювати ці самі датасети.

Для розробки API, було обрано FastAPI високопродуктивний фреймворк для створення API, на мові Python. FastAPI, був обраний саме тому що він ефективний для побудови високонавантаженних API, із великою кількостю одночасних запитів. Це досягається за допомогою підтримки асинхроних ендпоінтів, та бібліотеки Starlette як основи. Також FastAPI генерує інтерактивну інтерактивну API документацію у форматі OpenAPI. Що значно спрощує процес тестування та інтеграції з фронтенд-частиною. Фреймворк також надає можливість валідувати дані за допомогою датакласів бібліотеки Pydentic, наведемо програмну реалізацію такого класу, та його використання у головному модулі FastAPI:

```
from pydantic import BaseModel, Field
from typing import Dict, List, Optional


class AnalysisResponse(BaseModel):
    genres: List[str]
    freq_balance: Dict[str, float]
    bpm: int
    vocal: str
    emotion: str
    key: str
    recommendations: List[str] = Field(
        default_factory=list,
    )
    matched_track: Optional[str] = Field(None)


@app.post("/api/analyze-audio",
response_model=AnalysisResponse)
async def read_root(audio: UploadFile = File(...)):
    pass
```

Саме такий стек середовища розробки забезпечить найкраще виконання поставленної задачі.

3.2 Програмна реалізація основних функцій

Перед розробкою такої складної системи, доволі логічним кроком буде розділити її на підзавдання. У додатку для аналізу аудіо за допомогою машинного навчання, та інших аналітичних методів, є такі функції:

- аналіз жанру аудіофайлу
- музична характеристика аудіофайлу (BPM, емоційність, тоніка)
- розподіл частот у аудіофайлі
- система рекомендацій (пошук схожих треків) до заданого аудіофайлу

Також під аудіофайлом мається на увазі записаний за допомогою додатку фрагмент аудіо з мікрофона користувача довжиною від 15 до 30 секунд.

Спочатку необхідно реалізувати аналітичні методи, та пошук або створення датасетів для машинного навчання. Почнімо із створення музичної характеристики аудіофайлу (треку). Ця функція включає у себе дві аналітичних підфункції та дві на базі моделей машиного навчання.

Для подальших маніпуляцій із звуком, використовується перетворення за допомогою функції load, із бібліотеки librosa, яка була вмонтована в ініціалізатор класу AudioAnalysis:

```
def __init__(self, filename):
    y, sr = librosa.load(filename, sr=None)
    y_stereo, sr_stereo = librosa.load(filename, sr=None,
mono=False)
    self.y = y
    self.sr = sr
    self.y_stereo = y_stereo
    self.sr_stereo = sr_stereo
```

Для визначення середнього темпу треку або BPM використовується така функція із розробленного класу AudioAnalysis:

```
def get_bpm(self):
    onset_frames = librosa.onset.onset_detect(y=self.y,
sr=self.sr)
    onset_times = librosa.frames_to_time(onset_frames,
sr=self.sr)

    intervals = np.diff(onset_times)
    avg_interval = np.mean(intervals)
    bpm = 60.0 / float(avg_interval)
    return round(bpm)
```

Ця функція працює так витягаються «кадри» із аудіосигналу із початком ударів, нот і т.д. В результаті отримується масив номерів кадрів який перетворюється відповідною функцією на час у аудіофайлі, потім обчислюється відстань між елементами, та за допомогою усереднення ми отримаємо середній час між ударами, і за допомогою ділення на 60 отримуємо удари на хвилину.

При побудові цієї функції враховувалося те що звичайний підрахунок бітів від librosa, не працює із 16-тими долями, які надають пришвидчення, тому замість 220-230 BPM у треці Master of Puppets, результат був 113, що відповідає дійсності якщо опустити 16-ті долі. Проте цей метод дозволяє отримати корректне значення.

Для визначення тоніки використовується такий метод:

```
def get_keynote(self):
    chroma = librosa.feature.chroma_stft(y=self.y,
sr=self.sr, hop_length=512)
    chroma_mean = np.mean(chroma, axis=1)

    major_profile = np.array([6.35, 2.23, 3.48, 2.33, 4.38,
4.09, 2.52, 5.19, 2.39, 3.66, 2.29, 2.88])
    minor_profile = np.array([6.33, 2.68, 3.52, 5.38, 2.60,
3.53, 2.54, 4.75, 3.98, 2.69, 3.34, 3.17])

    major_profile = major_profile / np.sum(major_profile)
    minor_profile = minor_profile / np.sum(minor_profile)

    notes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G',
'G#', 'A', 'A#', 'B']

    max_corr = -1
    best_key = None
    best_mode = None

    for i in range(12):
        major_shifted = np.roll(major_profile, i)
        minor_shifted = np.roll(minor_profile, i)

        major_corr = np.corrcoef(chroma_mean,
major_shifted)[0, 1]
        minor_corr = np.corrcoef(chroma_mean,
minor_shifted)[0, 1]

        if major_corr > max_corr:
            max_corr = major_corr
            best_key = notes[i]
```

```
                best_mode = 'major'

            if minor_corr > max_corr:
                max_corr = minor_corr
                best_key = notes[i]
                best_mode = 'minor'

        return f"{best_key} {best_mode}"
```

Цей метод працює на основі хроматограми яка показує інтенсивність кожної з 12 нот від (C до B), в кожний момент часу, потім виконується усереднення за усім треком. Спочатку метод був заснований на простому аналізі хроматограми та виокремлення ступенів тоніки і співставлення їх із прораховоною таблицею тоніки, при цьому є дві проблеми:

- Суттєвий вплив частоти дискритизації
- Низька точність при визначенні Cmaj, та Amin

Низька точність визначення цих тонік пов’язана із тим що їх I ступінь, Cmaj (C, D, E, F, G, A, B), Amin (A, B, C, D, E, F, G) тобто містить усі ті самі ноти. Тому для точного визначення цих тонік використовується експеременталько визначені психо-акустичні коефіцієнти Krumhansl-Schmuckle[15] для мажора та мінора, нормалізуємо їх для порівняння із хроматограмою, потім алгоритм проходиться по усім можливим тонікам, та перевіряє кореляцію із хроматограмою профілів Krumhansl-Schmuckle. Та робить висновок на основі максимальної кореляції.

Для розподілу частот використовується такий метод:

```
amps = np.abs(librosa.stft(self.y))
energy = amps ** 2
freqs = librosa.fft_frequencies(sr=self.sr, n_fft=2048)

#Freq condition
low_mask = freqs < 250
mid_mask = (freqs >= 250) & (freqs < 4000)
high_mask = freqs >= 4000

low = energy[low_mask].sum()
mid = energy[mid_mask].sum()
high = energy[high_mask].sum()

total = low + mid + high

low = round(float(low / total * 100), 2)
```

```
mid = round(float(mid / total * 100), 2)
high = round(float(high / total * 100), 2)

return {'low': low, 'mid': mid, 'high': high}
```

За допомогою віконного перетворення Фур'є отримуємо амплітуди частот, підносячи цей вектор до модуля, після чого відбувається перетворення амплітуди на потужність шляхом піднесення до квадрату. Отримуємо частотний розподіл за частотою дискретизації, частоти від 0 до sr/2, за теоремою Найквіста. Після чого використовуються частотні маски для аналізу частот:

- 0-250 Гц – низькі частоти (бас, суббас)
- 250-4000 Гц – середні частоти (вокал, більшість інструментів)
- від 4000 Гц – високі частоти (тарілки, високі гармоніки)

Виконується підсумок енергії усіх частот в кожному із діапазонів, після чого обчислюємо суму усіх частот та вираховуємо відсотковий внесок кожної з них.

Для розпізнавання емоції яку викликає трек необхідно знайти або створити датасет, дослідивши сайт kaggle було знайдено датасет «EMOTIFY - Emotion classificaiton in Songs»[16] у якому маються аудіофайли та оцінки від респондентів чи відчувають вони ту чи іншу емоцію в треці, їх вік, гендер та інші характеристики. Тому щоб почати роботу із цим датасетом необхідно спочатку виконати деякий препроцесінг. А саме просумувати всі відповіді респондентів та для кожного треку позначити одиницею три найбільших за кількістю емоції, а інші позначити нулем, таким чином на кожен трек припадає позначення трьох емоцій яку відчуває більшість при прослуховуванні цих треків, однак при прогляданні розподілу класів на рисунку 3.1, можна побачити що розподіл класів є не збалансованим. При такому підході ми отримуємо багатокласову багатоміткову класифікацію, для якої така не збалансованість класів, може виявитися фатальною при питаннях точності виокримлення рідких класів. Тому щоб вирішити цю проблему необхідно прийняти рішення, що робити із цим, чи зменшувати кількість класів,

відкидувати малі класи, робити зважені класи чи використати метод SMOTE, для синтетичного збільшення малих класів. Проте в питаннях музики генерування SMOTE, може негативно вплинути, тому краще використовувати видалення або зваженість. Опис класів датасету може дати розуміння чи пов’язані класи семантично між собою.

- amazement => відчуття подиву і щастя
- solemnity => відчуття трансцендентності, натхнення. Трепет
- tenderness => чуттєвість, афект, почуття любові
- nostalgia => мрійливі, меланхолійні, сентиментальні почуття
- calmness => розслаблення, безтурботність, медитативність
- power => відчуття сили, героїчності, тріумфу, енергійності
- joyful activation => відчуття, що хочеться танцювати, підбадьорений, оживлений, веселий
- tension => нервовий, нетерплячий, роздратований
- sadness => пригнічений, сумний

Так і є Sadness та Nostalgia пов’язані тим що ці стани представляють меланхолію.

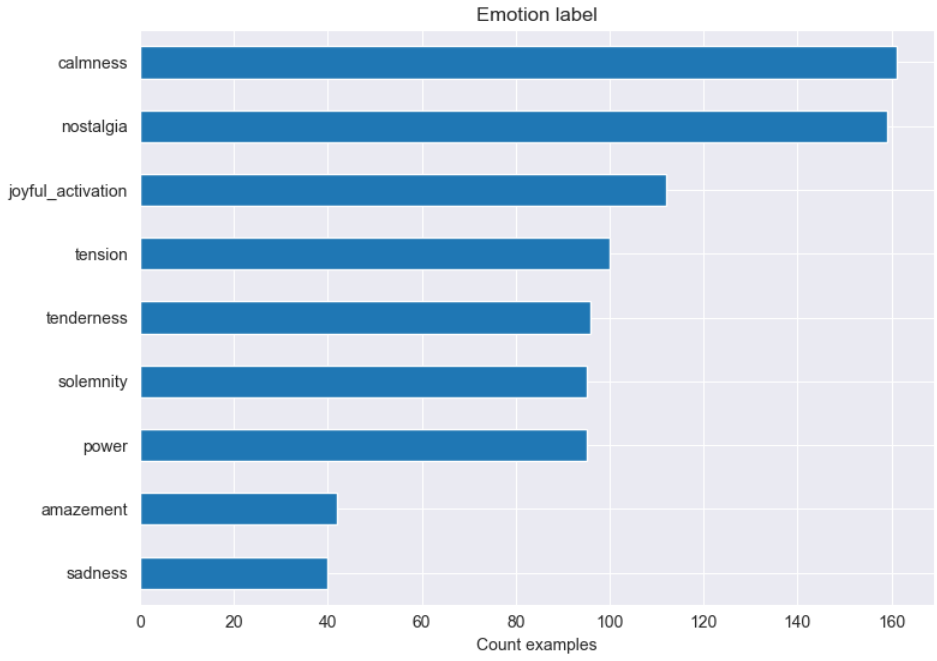

Рисунок 3.1 – Розподіл класів емоцій у датасеті.

Для прийняття рішення щодо видалення об’єктів або класів, необхідно поглянути на матрицю кореляцій між класами зображеного на рисунку 3.2.

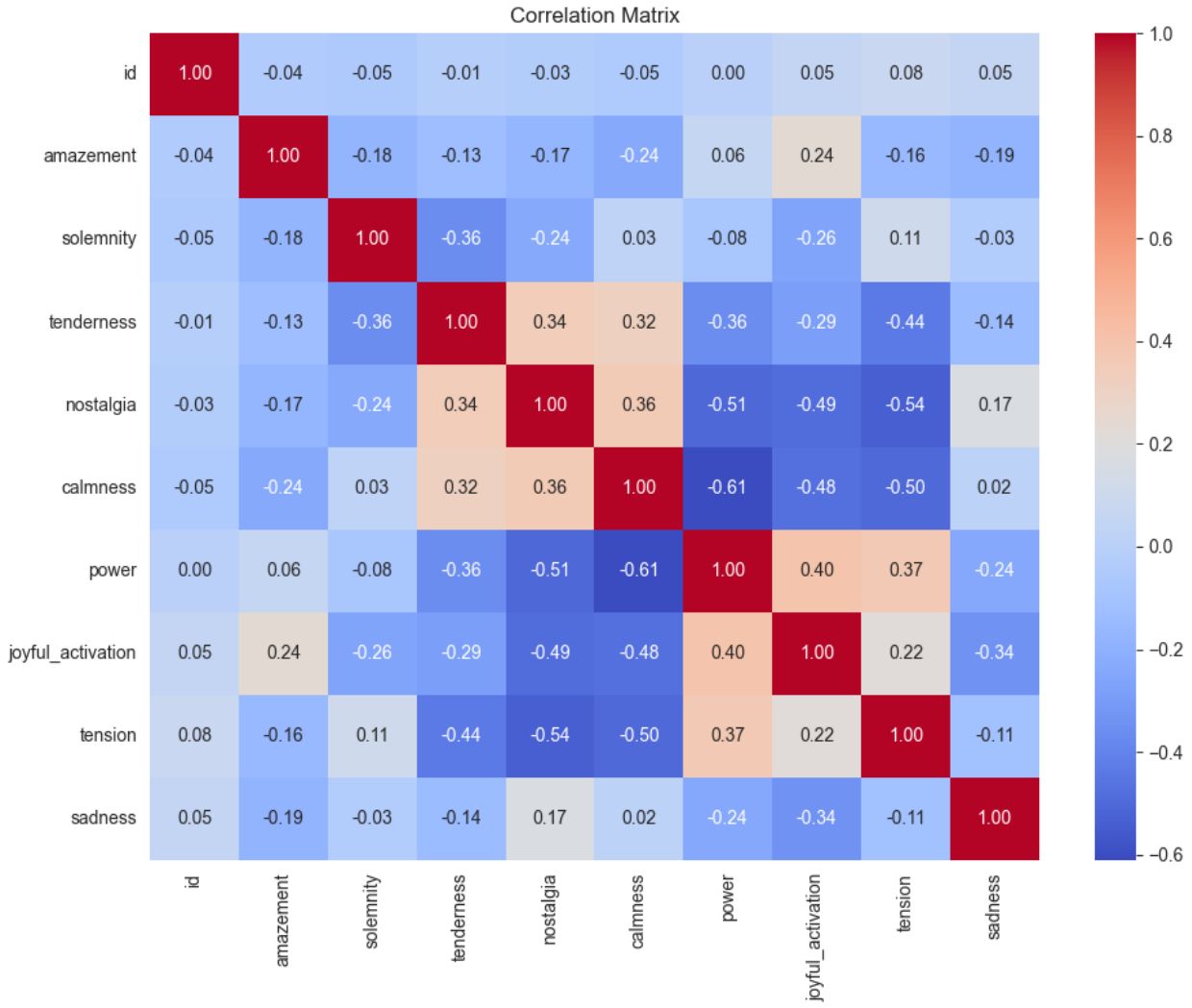

Рисунок 3.2 – Матриця кореляцій

Можна зробити висновок що кореляції між цими класами немає оскільки немає жодного значення вище ніж 0.65, проте можна побачити велике значення від’ємних кореляцій, що вказує на наявність протележних станів.

Тому враховуючи увесь цей аналіз, зроблено висновок, що мультиколліанарність даних відсутня, і прибирати який небудь з класів не має сенсу, має сенс навчити нейромережу розрізняти ці класи, а для цього ми використаємо метод compute_class_weight із бібліотеки sklearn, який надасть словник із значеннями вагів для функціоналу втрат по кожному із класів.

Проте метод class_weight застосовується лише для мультикласової класифікації, а не мультиміточної, застосування методу sample_weight, не дало ніяких покращень.

На рисунку 3.3 зображено процес навчання мультиміточної нейронної мережі яка повинна розрізняти емоції, точність поміток на тестовій виборці складає 70%, тобто 70% міток поставлено правильно, що є досить не поганим показником враховуючи те що ці мітки для деяких людей можуть мати досить схожий сенс.

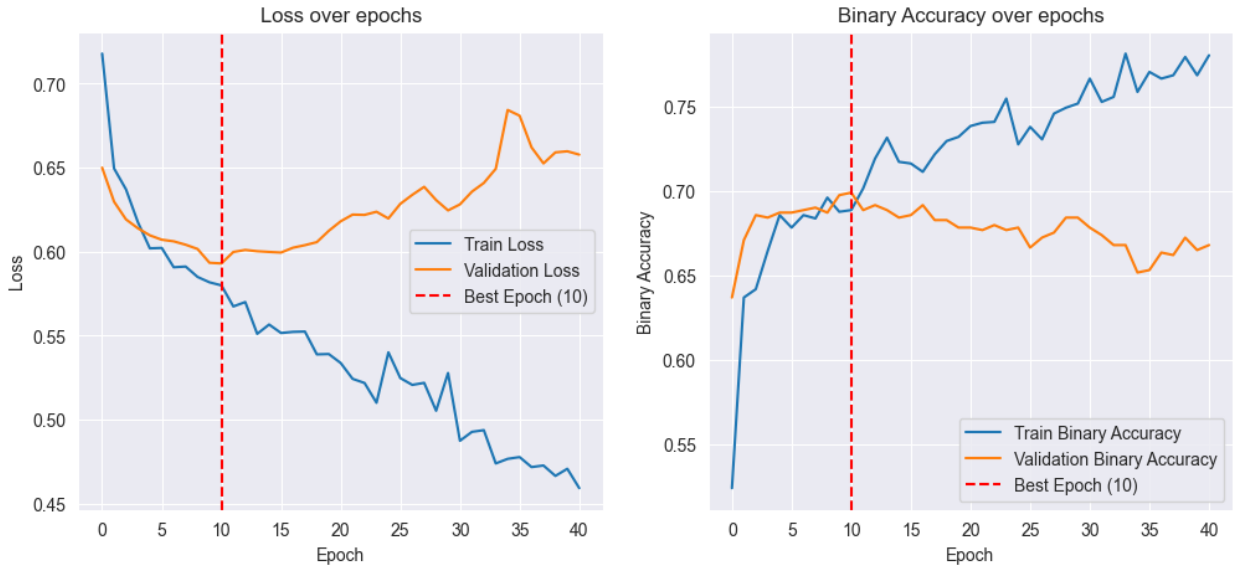

Рисунок 3.3 – Графіки навчання моделі емоційних позначок

На рисунку 3.4 зображено топологію нейронної мережі, на вхід подавалися 42 параметра витягнутих із аудіофайлу за допомогою бібліотеки librosa, після чого проводився аналіз, у прихованих шарах нейромережі. У якості функції активації замість ReLU, використовувався leakyReLU, який значно менше піддається проблемі мертвих нейронів із цією функцією активації. Загалом модель складається із трьох повнозв'язних шарів нейрони в яких зменшуються поступово в два рази, що сприяло поступовому узагальненню ознак. Для запобігання перенавчання застосовувався Dropout, із параметром 0.3. На вихідному шарі використовувалася сигмоїд, для кожного з

9 классів емоцій, що й дозволяє робити мітки на даних які подаються до цієї мережі.

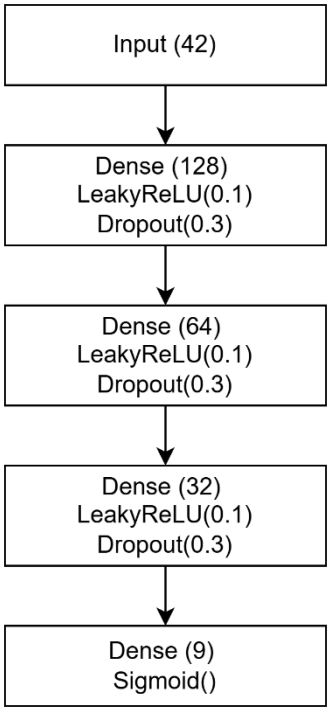

Рисунок 3.4 – Топологія нейронної мережі емоційних позначок

Модель навчалася 41 епоху, але найкращій результат був досягнутий за 10 епох, після чого почалося поступове перенавчання. Це вказує на доцільність використання EarlyStopping код якого наведено нижче:

```
keras.callbacks.EarlyStopping(
    monitor='val_binary_accuracy',
    patience=30,
    restore_best_weights=True,
    verbose=1,
),
```

На рисунку 3.5 зображено результат відображення усіх оброблених характеристик за допомогою алгоритмів та нейромереж.

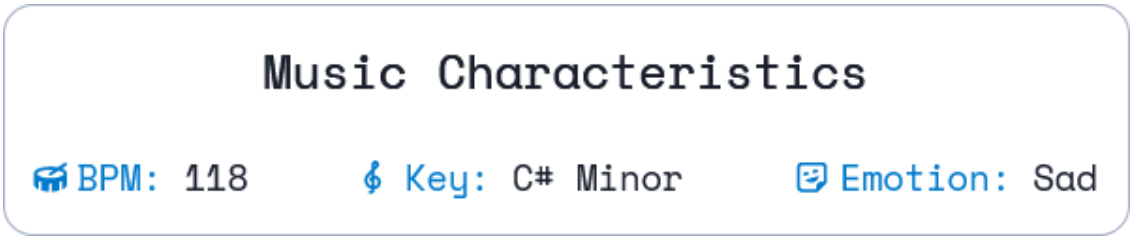

Рисунок 3.5 – Відображення у інтерфейсі результату визначення музичних характеристик

Для реалізації жанрової класифікації, було обрано датасет GTZAN, який містить у собі вже дістані за допомогою librosa аудіоознаки, а також треки у форматі .wav, такий датасет значно спрощує завдання оскільки можна пропустити етап із препроцесінгом і обробкою сирих аудіо файлів. Для вирішення цієї задачі можна використати самі різні методи машиного навчання. А саме повнозв'язні мережі, градієнтний бустінг, рандомний ліс. Порівнявши результати та оптимальність моделей, оберемо одну з них, почнімо із звичайної повнозв'язної мережі. На рисунку 3.5 зображено графіки звичайної повнозв'язної мережі, як видно після 79 эпохи починається перенавчання, тому callback ранньої зупинки, зупинив навчання побачивши що втрати на валідаційній виборці починають зростати

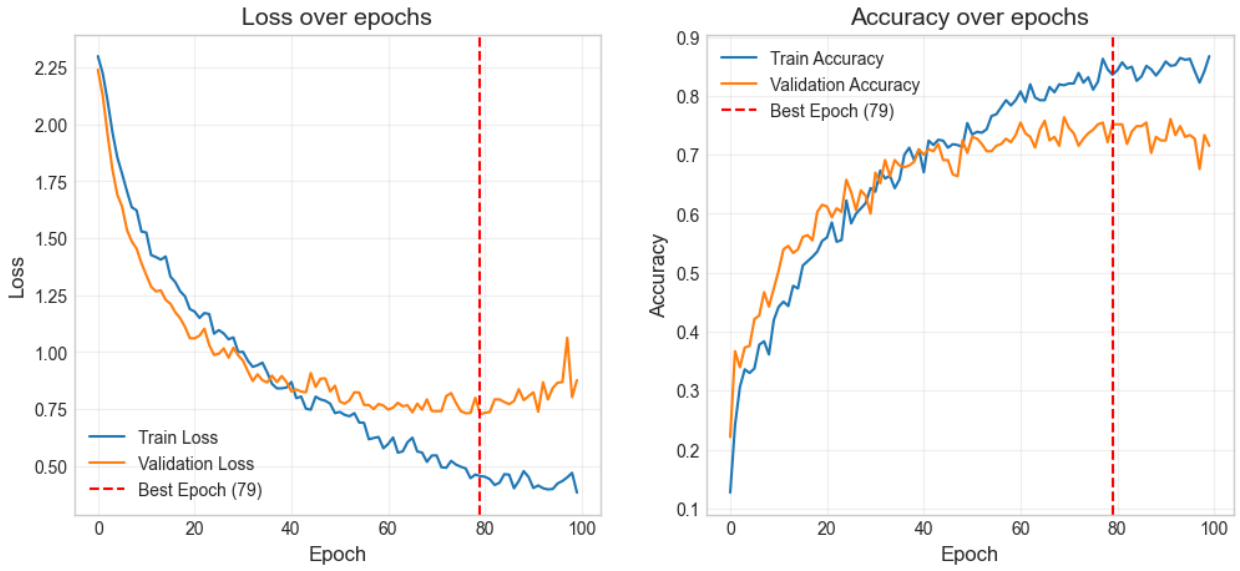

Рисунок 3.6 – Графіки навчання моделі визначення жанрів

Архітектура нейромережі зображена на рисунку 3.6 є досить складною, містить у собі один вхідний шар із 57 нейронами, чотири прихованих шара починаючи із 512 нейронів їх кількість зменшується вдвоє кожен раз. Вихід із нейронної мережі складається із 10 нейронів, які описують 10 жанрів. У якості функції активації використовується Softmax, оскільки задача мультикласової класифікації, а не мультиміточна, як при визначенні емоційного окрасу треку. Для запобігання перенавчання було додано Dropout із коефіцієнтом 0.2.

А також використовується рання зупинка навчання, якщо на валідаційній виборці втрати не зменшуються більше 20 епох.

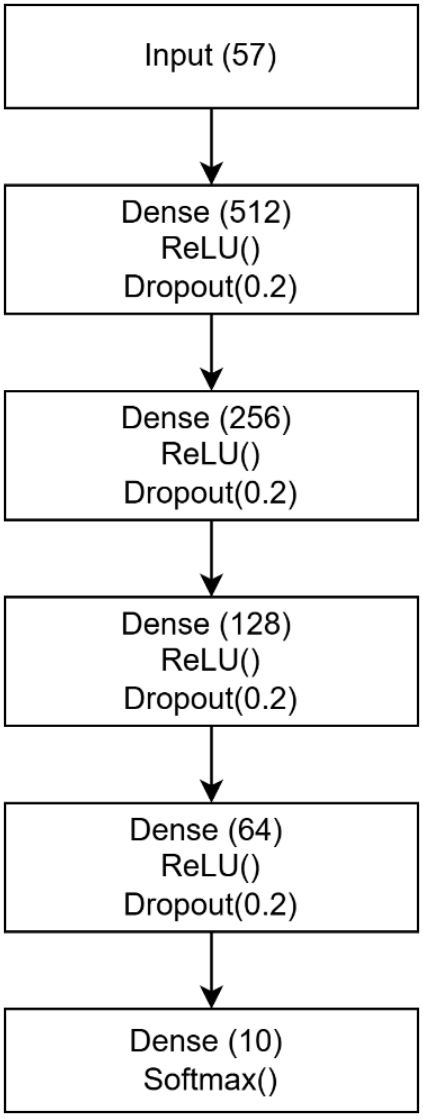

Рисунок 3.7 – Топологія нейромережі для визначення жанрів

Необхідно перевірити інші методи для вирішення цієї задачі. Результати показали, що алгоритм Random Forest продемонстрував точність на рівні 0.6920, тоді як XGBoost показав вищу ефективність з точністю 0.7240. Отримані дані свідчать про перевагу градієнтного бустингу XGBoost над методом випадкового лісу в контексті даної задачі. Проте нейромережа виявилася на 3% точнішою, тому було обрано саме її.

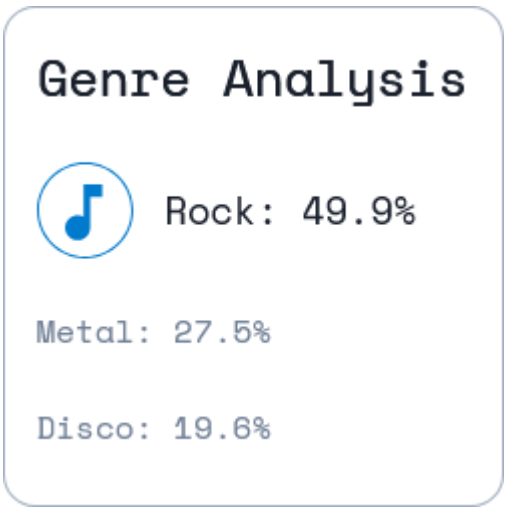

Рисунок 3.8 – Сформований блок інтерфейсу після аналізу жанрів

Для рекомендаційної системи (пошуку схожих мелодій) доцільно використовувати системи на основі k-найближчих сусідів, знаходячи за допомогою цього методу музику із найближчими музичними ознаками до завантаженної, для цього за допомогою librosa витягуємо вже знайомі характеристики: chroma, rms, centroid, bandwith, zcr та ін.

Для створення датасету було використано відео хостінг YouTube, та бібліотека для екстракту аудіо yt-dlp, код створення датасету:

```
from youtubesearchpython import VideosSearch
import yt_dlp
import time

def search_youtube(query):
    videos_search = VideosSearch(query, limit=1)
    result = videos_search.result()
    if result['result']:
        return result['result'][0]['link']
    return None

def download_mp3(url,
output_path="downloads/%(title)s.%(ext)s"):
    ydl_opts = {
        'format': 'bestaudio/best',
        'outtmpl': output_path,
        'postprocessors': [{
            'key': 'FFmpegExtractAudio',
            'preferredcodec': 'mp3',
            'preferredquality': '192',
        }],
        'quiet': False,
    }
```

```
        with yt_dlp.YoutubeDL(ydl_opts) as ydl:
            ydl.download([url])

    def download_songs(song_list):
        for song in song_list:
            query = song
            print(f" Шукаю: {query}")
            url = search_youtube(query)
            if url:
                print(f" Завантажую: {url}")
                download_mp3(url)
                time.sleep(1)
            else:
                print(f"Не знайдено: {query}")
```

За допомогою цієї функції та назв найпопулярніших треків Spotify був створений датасет із популярних пісень у форматі .mp3, після чого з них були витянуті ознаки

Для розв’язання цієї задачі використовується метод kNN, із метрикою косинусної відстань, де ознаками є аудіоознаки здобутих за допомогою бібліотеки librosa, на рисунку 3.9 зображено результат роботи цього модуля.

Рисунок 3.9 – Сформований блок інтерфейсу із схожою музикою до заданного аудіофайлу

В якості результатів модель k-NN, видає 3 результати, найбільш схожих треків. Тут кількість сусідів відповідає лише на кількість рекомендацій, а навчання полягає просто в розміщенні точок у просторі, тому ніякого підбору гіперпараметрів тут немає.

3.3 Методика роботи користувача

Взаємодія із програмою дуже проста і розібратися із нею зможе навіть недосвідченний користувач, спершу необхідно завантажити файл через «Drag ‘n’ Drop» зону, відповідний елемент відмічено на рисунку 3.10.

Рисунок 3.10 – Області для завантаження файлу у додаток

Передати файл можна двома способами, перетягнути файл на цю зону, або ж, натиснути на неї та обрати із своїх файлів. На рисунку 3.11 зображений процес вибору файлу із провідника, та процес підтвердження обраного файлу.

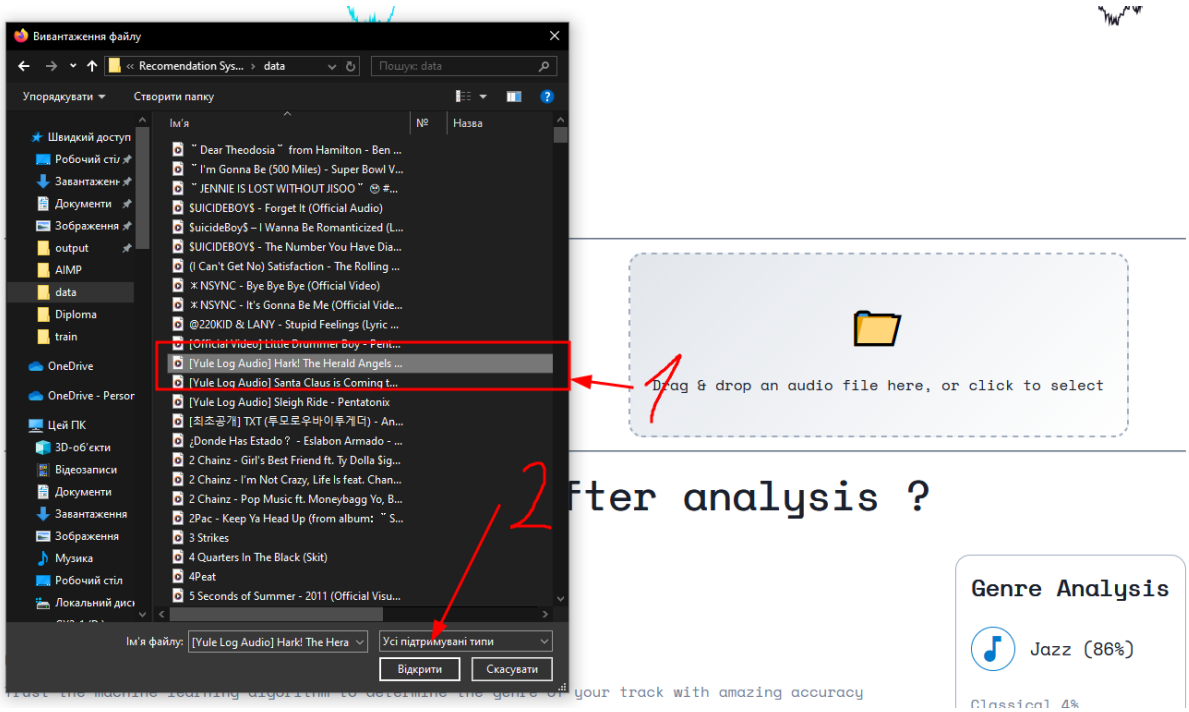

Рисунок 3.11 – Процес обирання файлу

Після того як користувач обрав файл, зона змінить свій вигляд сигналізуючи що зараз утримується саме цей файл, користувач може відмінити свій вибір натиснувши на хрестик з правого боку. Або ж натиснути на кнопку яка відправить трек на аналіз.

Рисунок 3.12 – Завантаженний файл у зоні «Drag ‘n’ Drop»

Коли користувач натисне на кнопку аналізу файлу, йому необхідно буде зачекати приблизно 10-15 секунд (при умові .flac аудіо та тривалості 3 хвилини). Додаток також надає користувачу зворотній відгук що почався аналіз, заблюривши приклад блоків аналізу, що можна побачити на рисунку 3.13

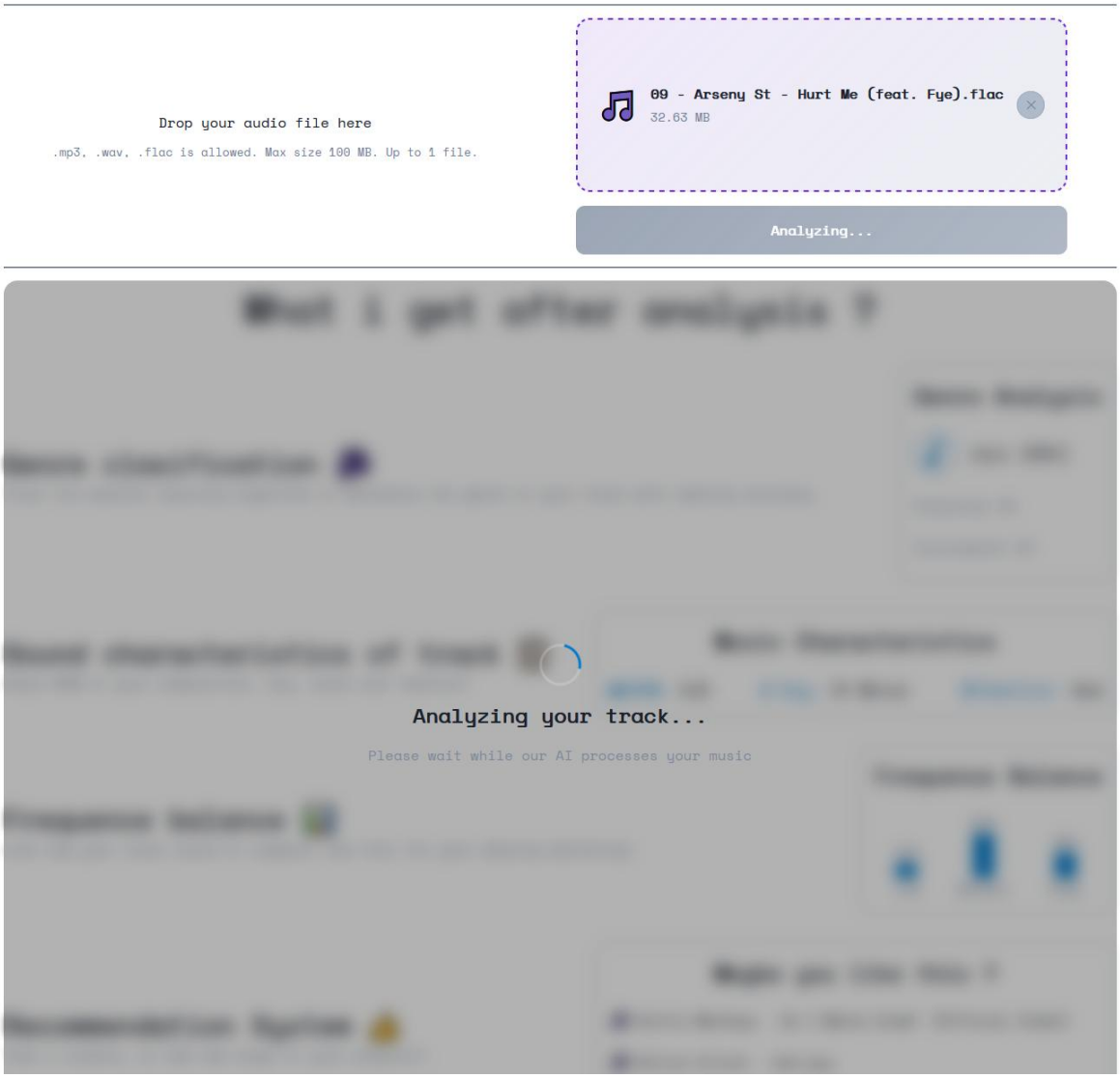

Рисунок 3.13 – Процес аналізу треку

Після очікування користувачем, він отримає інформативний цільний зміст який містить чотири блоки, які репрезентують результати аналізу, результат який побачить користувач зображено на рисунку 3.14.

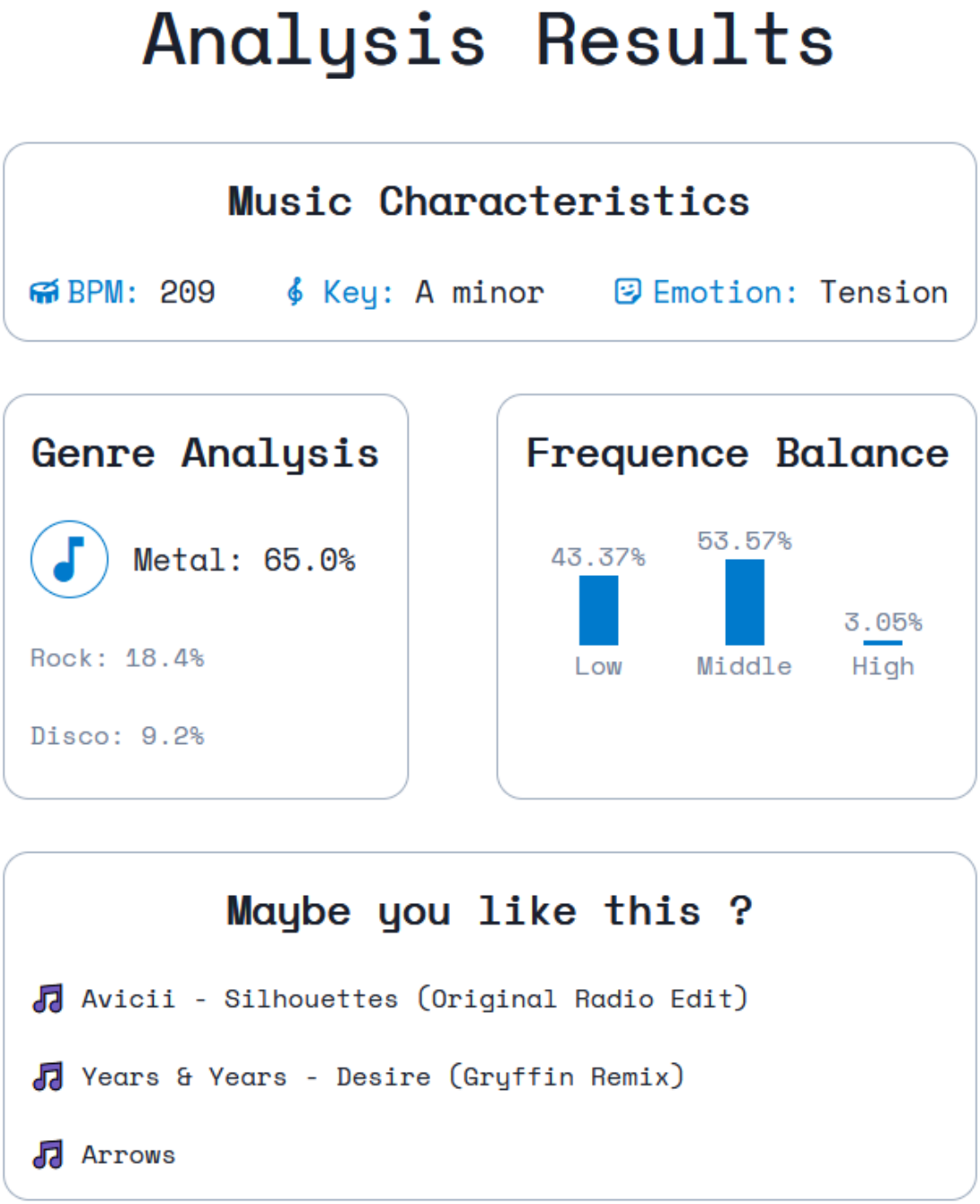

Рисунок 3.14 – Області для завантаження файлу у додаток

Мінімум дій, максимум результату, саме такий підхід дозволяє користувачеві почувати себе максимально комфортно при використанні додатку. Пізніше ці дані користувач може використати для просування свого треку, подальшого розвитку свого треку, чи фінального мастерінгу.

ВИСНОВКИ

В результаті виконанні квалифікаційної роботи було досліджено питання аналізу аудіо та використанні інструментів штучного інтелекту. Розглянуто популярні музичні платформи у якості існуючих аналогів, розроблено просту проте ефективну архітектуру ПЗ обгортки для розробленних моделей.

Реалізовано систему автоматичного аналізу музичних композицій з використанням алгоритмів машинного навчання, методів машинного навчання та аналітичних методів, що дозволяє ефективно класифікувати аудіоконтент за жанрами, настроєм та досліджувати інші важливі характеристики такі як тоніка чи розподіл частот.

Проведено порівняльний аналіз ефективності різних підходів до обробки аудіосигналів, який показав переваги застосування нейромереж для задач розпізнавання жанрів. Експериментальні дослідження підтвердили працездатність розробленої архітектури.

Розроблені моделі були збалансовані між простотою, точністю та часом обробки що дозволяє використовувати їх у великих проектах із можливим маштабуванням у майбутньому

ПЕРЕЛІК ПОСИЛАНЬ

1. Musical genre classification of audio signals / G. Tzanetakis, P. Cook // IEEE Transactions on Speech and Audio Processing. – 2002. – Т. 10, № 5. – С. 293–302.

2. librosa – librosa 0.11.0 documentation [Електронний ресурс] // librosa. – Режим доступу: https://librosa.org/doc/latest/index.html.

3. librosa: Audio and Music Signal Analysis in Python / Brian McFee [та ін.] // Python in Science Conference, Austin, Texas., 2015.

4. Allamy S., Koerich A. L. 1D CNN architectures for music genre classification. Матеріали 2021 IEEE symposium series on computational intelligence (SSCI). 2021. С 1-7.

5. Sridhar A. Attention-guided Spectrogram Sequence Modeling with CNNs for Music Genre Classification / Aditya Sridhar // arXiv. – 2024.

6. Shi G. Data Compression for Audio-based Smart Beekeeping / Guangyu Shi, Lei Song, Iman Ardekani // Atlantis Highlights in Sustainable Development. – Dordrecht, 2024. – С. 117–130.

7. Brandenburg K. MP3 and AAC explained. High Quality Audio Coding: тези доповідей. Матеріали 17th International Audio Engineering Society Conference. 2024.

8. Sunray E. Sounds of science: Copyright infringement in AI music generator outputs. / E. Sunray // Cath. UJL & Tech. – 2021. – Т. 29. – С. 185–218.

9. Spotify - Web Player: Music for everyone [Електронний ресурс] // Spotify. – Режим доступу: https://spotify.com.

10. Youtube Music [Електронний ресурс] // Youtube Music. – Режим доступу: https://music.youtube.com.

11. Shazam - Music Discovery, Charts & Song Lyrics [Електронний ресурс] // Shazam. – Режим доступу: https://www.shazam.com.

12. Stream and listen to music online for free with SoundCloud [Електронний ресурс] // SoundCloud. – Режим доступу: https://soundcloud.com.

13. Next.js by Vercel - The React Framework [Електронний ресурс] // Next.js. – Режим доступу: https://nextjs.org/.

14. React Reference Overview [Електронний ресурс] // React. – Режим доступу: https://react.dev/reference/react.

15. Temperley D. What's Key for Key? The Krumhansl-Schmuckler Key-Finding Algorithm Reconsidered / David Temperley // Music Perception. – 1999. – Т. 17, № 1. – С. 65–100.

16. Aljanaki A. Studying emotion induced by music through a crowdsourcing game / Anna Aljanaki, Frans Wiering, Remco C. Veltkamp // Information Processing & Management. – 2016. – Т. 52, № 1. – С. 115–128.

Додаток А

Код програми

Файл «main.py»

```
import io
from fastapi import FastAPI, Request, UploadFile, File
from fastapi.middleware.cors import CORSMiddleware
import analysis
from my_classes import *
import tempfile
import os
import time


app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.post("/api/analyze-audio",
response_model=AnalysisResponse)
async def read_root(audio: UploadFile = File(...)):
    start = time.time()
    audio_bytes = await audio.read()
    file_extension = os.path.splitext(audio.filename)[1] if
audio.filename else '.wav'

    with tempfile.NamedTemporaryFile(suffix=file_extension,
delete=False) as tmp_file:
        tmp_file.write(audio_bytes)
        tmp_file_path = tmp_file.name

    aa = analysis.AudioAnalysis(tmp_file_path)

    response = AnalysisResponse(
        genres=aa.get_genre(),
        freq_balance=aa.get_freq_balance(),
        bpm=aa.get_bpm(),
        emotion=aa.get_emotion(),
        key=aa.get_keynote(),
        recommendations=aa.get_similarities(),
    )

    print(f"Аналіз зайняв {time.time() - start:.4f} сек")

    return response
```

Файл «my_classes.py»

```
from pydantic import BaseModel, Field
from typing import Dict, List, Optional


class AnalysisResponse(BaseModel):
    genres: List[str]
    freq_balance: Dict[str, float]
    bpm: int
    emotion: str
    key: str
    recommendations: List[str] = Field(
        default_factory=list,
    )
```

Файл «analysis.py»

```
import librosa
import numpy as np
from keras import models
import joblib
import pandas as pd
import time
from librosa.feature import zero_crossing_rate


class EmotionalRecognizer:
    def __init__(self):
        self.model =
models.load_model('./models/emotional_recog/emotion_model.ke
ras')
        self.scaler =
joblib.load('./models/emotional_recog/scaler.joblib')
        self.emotions = ['Amazement', 'Solemnity',
'Tenderness', 'Nostalgia', 'Calmness', 'Power', 'Joyful
Activation', 'Tension', 'Sadness']

    def analyze(self, features):
        df = pd.DataFrame([features])
        x = self.scaler.transform(df)
        predict = self.model.predict(x)
        predict = self.emotions[np.argmax(predict)]
        return predict

class GenreRecognizer:
    def __init__(self):
        self.model =
models.load_model('./models/genre_recog/genre_model.keras')
        self.scaler =
```

```
joblib.load('./models/genre_recog/scaler.joblib')
        self.genres = ['Blues', 'Classical', 'Country',
'Disco', 'Hip-Hop', 'Jazz', 'Metal', 'Pop', 'Reggae',
'Rock']

    def analyze(self, features):
        df = pd.DataFrame([features])
        x = self.scaler.transform(df)
        predict = self.model.predict(x)

        top_3_indices = np.argsort(predict[0])[-3:][::-1]
        top_3_results = []

        for idx in top_3_indices:
            genre = self.genres[idx]
            probability = predict[0][idx] * 100
            top_3_results.append(f"{genre}:
{probability:.1f}%")

        return top_3_results

class MusicSimilarityRecognizer:
    def __init__(self):
        self.model =
joblib.load('./models/music_similarity/knn_model.joblib')
        self.scaler =
joblib.load('./models/music_similarity/scaler.joblib')
        self.y =
joblib.load('./models/music_similarity/labels.joblib')

    def analyze(self, features):
        df = pd.DataFrame([features])
        x = self.scaler.transform(df)

        distances, indices = self.model.kneighbors(x,
n_neighbors=3)

        return [self.y.iloc[i] for i in indices[0]]


class AudioAnalysis:
    def __init__(self, filename):
        y, sr = librosa.load(filename, sr=None)

        self.y = y
        self.sr = sr

        self.emotional_recognizer = EmotionalRecognizer()
        self.genre_recognizer = GenreRecognizer()
        self.music_similarity_recognizer =
MusicSimilarityRecognizer()

        self._get_features()
```

```
    def _get_features(self):
        stft = librosa.stft(self.y)
        self.stft = stft
        mel_spec =
librosa.feature.melspectrogram(S=np.abs(stft)**2,
sr=self.sr)

        def _get_mfcc_both():
            mfcc_20 =
librosa.feature.mfcc(S=librosa.power_to_db(mel_spec),
n_mfcc=20)
            mfcc_13 = mfcc_20[:13]
            return mfcc_13, mfcc_20

        def _get_chroma():
            return
librosa.feature.chroma_stft(S=np.abs(stft), sr=self.sr,
hop_length=512)

        def _get_spectral_contrast():
            return
librosa.feature.spectral_contrast(S=np.abs(stft),
sr=self.sr)

        def _get_tonnetz():
            return librosa.feature.tonnetz(y=self.y,
sr=self.sr)

        def _get_zcr():
            return
librosa.feature.zero_crossing_rate(self.y)

        def _get_rmse():
            return librosa.feature.rms(S=stft)

        def _get_spectral_centroid():
            return
librosa.feature.spectral_centroid(S=np.abs(stft),
sr=self.sr)

        def _get_spectral_bandwidth():
            return
librosa.feature.spectral_bandwidth(S=np.abs(stft),
sr=self.sr)

        def _get_rolloff():
            return
librosa.feature.spectral_rolloff(S=np.abs(stft), sr=self.sr)

        def _get_tempo():
```

```
            tempo, _ = librosa.beat.beat_track(y=self.y,
sr=self.sr)
            return tempo[0]



        mfcc = _get_mfcc_both()
        self.mfcc_20 = mfcc[1]
        self.mfcc_13 = mfcc[0]
        self.chroma = _get_chroma()
        self.spectral_contrast = _get_spectral_contrast()
        self.tonnetz = _get_tonnetz()
        self.zcr = _get_zcr()
        self.rmse = _get_rmse()
        self.spectral_centroid = _get_spectral_centroid()
        self.spectral_bandwidth = _get_spectral_bandwidth()
        self.rolloff = _get_rolloff()
        self.tempo = _get_tempo()

        self.mfcc_20_mean = np.mean(self.mfcc_20, axis=1)
        self.mfcc_13_mean = np.mean(self.mfcc_13, axis=1)
        self.chroma_mean = np.mean(self.chroma, axis=1)
        self.spectral_contrast_mean =
np.mean(self.spectral_contrast, axis=1)
        self.tonnetz_mean = np.mean(self.tonnetz, axis=1)
        self.zcr_mean = np.mean(self.zcr)
        self.rmse_mean = np.mean(self.rmse)
        self.spectral_centroid_mean =
np.mean(self.spectral_centroid)
        self.spectral_bandwidth_mean =
np.mean(self.spectral_bandwidth)
        self.rolloff_mean = np.mean(self.rolloff)

        self.mfcc_20_var = np.var(self.mfcc_20, axis=1)
        self.mfcc_13_var = np.var(self.mfcc_13, axis=1)
        self.chroma_var = np.var(self.chroma, axis=1)
        self.spectral_contrast_var =
np.var(self.spectral_contrast, axis=1)
        self.tonnetz_var = np.var(self.tonnetz, axis=1)
        self.zcr_var = np.var(self.zcr)
        self.rmse_var = np.var(self.rmse)
        self.spectral_centroid_var =
np.var(self.spectral_centroid)
        self.spectral_bandwidth_var =
np.var(self.spectral_bandwidth)
        self.rolloff_var = np.var(self.rolloff)

    def get_bpm(self):
        onset_frames = librosa.onset.onset_detect(y=self.y,
sr=self.sr)
        onset_times = librosa.frames_to_time(onset_frames,
sr=self.sr)
```

```
        intervals = np.diff(onset_times)
        avg_interval = np.mean(intervals)
        bpm = 60.0 / float(avg_interval)
        return round(bpm)

    def get_keynote(self):
        chroma = librosa.feature.chroma_stft(y=self.y,
sr=self.sr, hop_length=512)
        chroma_mean = np.mean(chroma, axis=1)

        major_profile = np.array([6.35, 2.23, 3.48, 2.33,
4.38, 4.09, 2.52, 5.19, 2.39, 3.66, 2.29, 2.88])
        minor_profile = np.array([6.33, 2.68, 3.52, 5.38,
2.60, 3.53, 2.54, 4.75, 3.98, 2.69, 3.34, 3.17])

        major_profile = major_profile /
np.sum(major_profile)
        minor_profile = minor_profile /
np.sum(minor_profile)

        notes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G',
'G#', 'A', 'A#', 'B']

        max_corr = -1
        best_key = None
        best_mode = None

        for i in range(12):
            major_shifted = np.roll(major_profile, i)
            minor_shifted = np.roll(minor_profile, i)

            major_corr = np.corrcoef(chroma_mean,
major_shifted)[0, 1]
            minor_corr = np.corrcoef(chroma_mean,
minor_shifted)[0, 1]

            if major_corr > max_corr:
                max_corr = major_corr
                best_key = notes[i]
                best_mode = 'major'

            if minor_corr > max_corr:
                max_corr = minor_corr
                best_key = notes[i]
                best_mode = 'minor'

        return f"{best_key} {best_mode}"

    def get_emotion(self):
        features = {}

        for i, coef in enumerate(self.mfcc_13_mean, 1):
            features[f'mfcc_{i}'] = coef
```

```
        for i, val in enumerate(self.chroma_mean, 1):
            features[f'chroma_{i}'] = val

        for i, val in enumerate(self.spectral_contrast_mean,
1):
            features[f'contrast_{i}'] = val

        for i, val in enumerate(self.tonnetz_mean, 1):
            features[f'tonnetz_{i}'] = val

        features['zcr'] = self.zcr_mean

        features['rmse'] = self.rmse_mean

        features['spectral_centroid'] =
self.spectral_centroid_mean

        features['spectral_bandwidth'] =
self.spectral_bandwidth_mean

        result = self.emotional_recognizer.analyze(features)

        return result

    def get_genre(self):
        features = {}

        features['chroma_stft_mean'] =
np.mean(self.chroma_mean)
        features['chroma_stft_var'] =
np.var(self.chroma_var)

        features['rms_mean'] = self.rmse_mean
        features['rms_var'] = self.rmse_var

        features['spectral_centroid_mean'] =
np.mean(self.spectral_centroid_mean)
        features['spectral_centroid_var'] =
np.var(self.spectral_centroid_var)

        features['spectral_bandwidth_mean'] =
self.spectral_bandwidth_mean
        features['spectral_bandwidth_var'] =
self.spectral_bandwidth_var

        features['rolloff_mean'] = self.rolloff_mean
        features['rolloff_var'] = self.rolloff_var

        features['zero_crossing_rate_mean'] = self.zcr_mean
        features['zero_crossing_rate_var'] = self.zcr_var

        features['tempo'] = self.tempo
```

```
        for i in range(20):
            features[f'mfcc{i+1}_mean'] =
self.mfcc_20_mean[i]
            features[f'mfcc{i+1}_var'] = self.mfcc_20_var[i]

        return self.genre_recognizer.analyze(features)


    def get_freq_balance(self):
        amps = np.abs(self.stft)
        energy = amps ** 2
        freqs = librosa.fft_frequencies(sr=self.sr,
n_fft=2048)

        #Freq condition
        low_mask = freqs < 250
        mid_mask = (freqs >= 250) & (freqs < 4000)
        high_mask = freqs >= 4000

        low = energy[low_mask].sum()
        mid = energy[mid_mask].sum()
        high = energy[high_mask].sum()

        total = low + mid + high

        low = round(float(low / total * 100), 2)
        mid = round(float(mid / total * 100), 2)
        high = round(float(high / total * 100), 2)

        return {'low': low, 'mid': mid, 'high': high}

    def get_similarities(self):
        features = {}

        features['chroma_stft_mean'] =
self.chroma_mean.mean()
        features['chroma_stft_var'] = self.chroma_var.var()

        features['rms_mean'] = self.rmse_mean
        features['rms_var'] = self.rmse_var

        features['spectral_centroid_mean'] =
self.spectral_centroid_mean
        features['spectral_centroid_var'] =
self.spectral_centroid_var

        features['spectral_bandwidth_mean'] =
self.spectral_bandwidth_mean
        features['spectral_bandwidth_var'] =
self.spectral_bandwidth_var

        features['rolloff_mean'] = self.rolloff_mean
```

```
        features['rolloff_var'] = self.rolloff_var

        features['zero_crossing_rate_mean'] = self.zcr_mean
        features['zero_crossing_rate_var'] = self.zcr_var

        features['tempo'] = self.tempo

        return
self.music_similarity_recognizer.analyze(features)
```

Файл « Emotional/train.py »

```
import keras
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt
#%%
X = pd.read_csv('.\\dataset\\train\\X_train.csv')
y = pd.read_csv('.\\dataset\\train\\y_train.csv')

col = 'id'
cols = [col] + [c for c in X.columns if c != col]
X = X[cols]

X.head()
#%%
y.drop(columns=['id']).sum().sort_values().plot(kind='barh',
figsize=(8, 6), title="Emotion label")
plt.xlabel("Count examples")
plt.tight_layout()
plt.show()
#%%
correlation_matrix = y.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f")
plt.title("Correlation Matrix")
plt.tight_layout()
plt.show()
#%%
y
#%%
X.columns
#%%
y.columns
#%%
X = X.drop(columns=['id'])
y = y.drop(columns=['id'])
#%%
```

```
y.head()
#%%
X.head()
#%%
scaler = StandardScaler()
scaler.fit(X)

X = scaler.transform(X)
#%%
import joblib
with open('scaler.joblib', 'wb') as file:
    joblib.dump(scaler, file)
#%%
from tensorflow.keras.layers import LeakyReLU

model = keras.Sequential([
    keras.Input(shape=(42,)),

    keras.layers.Dense(128),
    LeakyReLU(negative_slope=0.1),
    keras.layers.Dropout(0.3),

    keras.layers.Dense(64),
    LeakyReLU(negative_slope=0.1),
    keras.layers.Dropout(0.3),

    keras.layers.Dense(32),
    LeakyReLU(negative_slope=0.1),
    keras.layers.Dropout(0.3),

    keras.layers.Dense(9, activation='sigmoid')
])

model.compile(keras.optimizers.Adam(learning_rate=0.002),
loss='binary_crossentropy', metrics=['binary_accuracy'])
#%%
callbacks = [
    keras.callbacks.EarlyStopping(
        monitor='val_binary_accuracy',
        patience=30,
        restore_best_weights=True,
        verbose=1,
    ),

    keras.callbacks.ModelCheckpoint(
        'best_emotion_model.keras',
        monitor='val_binary_accuracy',
        save_best_only=True,
        verbose=1
    )
]
#%%
X_train, X_val, y_train, y_val = train_test_split(X, y,
```

```
test_size=0.25, random_state=42)

fitted_model = model.fit(X_train, y_train,
validation_data=(X_val, y_val), epochs=500,
callbacks=callbacks)
#%%
import matplotlib.pyplot as plt

history = fitted_model.history
best_epoch = callbacks[0].best_epoch

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history['loss'], label='Train Loss')
plt.plot(history['val_loss'], label='Validation Loss')
plt.axvline(x=best_epoch, color='r', linestyle='--',
label=f'Best Epoch ({best_epoch})')
plt.title('Loss over epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history['binary_accuracy'], label='Train Binary
Accuracy')
plt.plot(history['val_binary_accuracy'], label='Validation
Binary Accuracy')
plt.axvline(x=best_epoch, color='r', linestyle='--',
label=f'Best Epoch ({best_epoch})')
plt.title('Binary Accuracy over epochs')
plt.xlabel('Epoch')
plt.ylabel('Binary Accuracy')
plt.legend()

plt.show()
```

Файл «Emotional/preprocess.py»

```
import pandas as pd
import os
from sklearn.model_selection import train_test_split
import shutil
import librosa
from tqdm import tqdm
#%%
df = pd.read_csv('.\\EmoMusic\\data.csv', sep=',')

df.columns = df.columns.str.strip()

df = df.drop(columns=['gender', 'age', 'liked', 'disliked',
'mother tongue', 'mood'])
```

```
emotion_cols = ['amazement', 'solemnity', 'tenderness',
'nostalgia', 'calmness', 'power', 'joyful_activation',
'tension', 'sadness']

grouped_df = df.groupby(['track id',
'genre'])[emotion_cols].sum().reset_index()

grouped_df
#%%
emotion_cols = ['amazement', 'solemnity', 'tenderness',
'nostalgia', 'calmness', 'power', 'joyful_activation',
'tension', 'sadness']

def top_3_mask(row):
    top3_indices =
row[emotion_cols].nlargest(3).iloc[:3].index
    mask = pd.Series(0, index=emotion_cols)
    mask[top3_indices] = 1
    return mask

grouped_df[emotion_cols] =
grouped_df[emotion_cols].apply(top_3_mask, axis=1)
grouped_df
#%%
preprocessed_df = grouped_df.copy()

def set_id_by_genre_track(data, genre, track_id, new_id):
    data.loc[(data['genre'] == genre) & (data['track id'] ==
track_id), 'id'] = new_id

source_root = 'EmoMusic'

train_dest = 'dataset\\train'
test_dest = 'dataset\\test'

os.makedirs(train_dest, exist_ok=True)
os.makedirs(test_dest, exist_ok=True)

preprocessed_df.insert(0, 'id', range(1,
len(preprocessed_df) + 1))

original_paths = [(os.path.join(source_root, genre,
f"{(track_id % 100) or 100}.mp3"), new_id) for new_id,
track_id, genre in preprocessed_df[['id', 'track id',
'genre']].itertuples(index=False)]

train, test = train_test_split(original_paths,
test_size=0.25, random_state=42, shuffle=True)

df_train = pd.DataFrame()

for path, new_id in train:
```

```
    shutil.copy2(path, os.path.join(train_dest,
f"{new_id}.mp3"))
    selected_row = preprocessed_df[preprocessed_df['id'] ==
new_id].copy()
    selected_row = selected_row.drop(columns=['genre',
'track id'])
    df_train = pd.concat([df_train, selected_row],
ignore_index=True)

df_train.to_csv(os.path.join(train_dest, 'y_train.csv'),
index=False)
df_test = pd.DataFrame()

for path, new_id in test:
    shutil.copy2(path, os.path.join(test_dest,
f"{new_id}.mp3"))
    selected_row = preprocessed_df[preprocessed_df['id'] ==
new_id].copy()
    selected_row = selected_row.drop(columns=['genre',
'track id'])
    df_test = pd.concat([df_test, selected_row],
ignore_index=True)

df_test.to_csv(os.path.join(test_dest, 'y_test.csv'),
index=False)
#%%
def extract_audio_features(file_path):
    y, sr = librosa.load(file_path, sr=22500)

    features = {}

    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
    for i, coef in enumerate(mfcc.mean(axis=1), 1):
        features[f'mfcc_{i}'] = coef

    chroma = librosa.feature.chroma_stft(y=y, sr=sr)
    for i, val in enumerate(chroma.mean(axis=1), 1):
        features[f'chroma_{i}'] = val

    contrast = librosa.feature.spectral_contrast(y=y, sr=sr)
    for i, val in enumerate(contrast.mean(axis=1), 1):
        features[f'contrast_{i}'] = val

    tonnetz =
librosa.feature.tonnetz(y=librosa.effects.harmonic(y),
sr=sr)
    for i, val in enumerate(tonnetz.mean(axis=1), 1):
        features[f'tonnetz_{i}'] = val

    zcr = librosa.feature.zero_crossing_rate(y)
    features['zcr'] = zcr.mean()

    rmse = librosa.feature.rms(y=y)
```

```
    features['rmse'] = rmse.mean()

    centroid = librosa.feature.spectral_centroid(y=y, sr=sr)
    features['spectral_centroid'] = centroid.mean()

    bandwidth = librosa.feature.spectral_bandwidth(y=y,
sr=sr)
    features['spectral_bandwidth'] = bandwidth.mean()

    return features
#%%
from concurrent.futures import ThreadPoolExecutor,
as_completed

features_list = []

def process_audio(train_audio_id):
    audio_path = os.path.join(train_dest,
f'{train_audio_id}.mp3')
    feats = extract_audio_features(audio_path)
    return feats

with ThreadPoolExecutor(max_workers=8) as executor:
    futures = {executor.submit(process_audio,
train_audio_id): train_audio_id for train_audio_id in
df_train['id'].tolist()}
    for future in tqdm(as_completed(futures),
total=len(futures), desc='Extracting features'):
        features_list.append(future.result())

df_train_X = pd.DataFrame(features_list)
df_train_X.to_csv(os.path.join(train_dest, 'X_train.csv'),
index=False)
#%%
from concurrent.futures import ThreadPoolExecutor,
as_completed

features_list = []

def process_audio(test_audio_id):
    audio_path = os.path.join(test_dest,
f'{test_audio_id}.mp3')
    feats = extract_audio_features(audio_path)
    feats['id'] = test_audio_id
    return feats

with ThreadPoolExecutor(max_workers=8) as executor:
    futures = {executor.submit(process_audio,
test_audio_id): test_audio_id for test_audio_id in
df_test['id'].tolist()}
    for future in tqdm(as_completed(futures),
total=len(futures), desc='Extracting features'):
        features_list.append(future.result())
```

```
df_test_X = pd.DataFrame(features_list)
df_test_X.to_csv(os.path.join(test_dest, 'X_test.csv'),
index=False)
```

Файл «Genre/Dense.ipynb»

```
import joblib
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
import numpy as np
import keras
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping
from sklearn.metrics import confusion_matrix
import librosa
from keras.optimizers import Adam
#%%

dataset_path = './Data/genres_original'
genres = ['blues', 'classical', 'country', 'disco',
'hiphop', 'jazz', 'metal', 'reggae', 'pop', 'rock']
def preprocess_dataset():
    data = []

    for genre in genres:
        genre_path = os.path.join(dataset_path, genre)
        filelist = os.listdir(genre_path)

        for file in filelist:
            file_path = os.path.join(genre_path, file)
            features = extract_feature(file_path, genre)
            if features:
                data.append(features)

    df = pd.DataFrame(data)
    df.to_csv("genre_features.csv", index=False)
    print("Збережено у genre_features.csv")



def extract_feature(file_path, genre):
    try:
        y, sr = librosa.load(file_path, sr=None)
        stft = librosa.stft(y)

        chroma = librosa.feature.chroma_stft(S=np.abs(stft),
sr=sr, hop_length=512)
        rms = librosa.feature.rms(S=stft)
        centroid =
librosa.feature.spectral_centroid(S=np.abs(stft), sr=sr)
        bandwidth =
```

```
librosa.feature.spectral_bandwidth(S=np.abs(stft), sr=sr)
        rolloff =
librosa.feature.spectral_rolloff(S=np.abs(stft), sr=sr)
        zcr = librosa.feature.zero_crossing_rate(y)
        tempo, _ = librosa.beat.beat_track(y=y, sr=sr)
        mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=20)

        features = {
            'genre': genre,
            'filename': os.path.basename(file_path),
            'chroma_stft_mean': chroma.mean(),
            'chroma_stft_var': chroma.var(),
            'rms_mean': rms.mean(),
            'rms_var': rms.var(),
            'spectral_centroid_mean': centroid.mean(),
            'spectral_centroid_var': centroid.var(),
            'spectral_bandwidth_mean': bandwidth.mean(),
            'spectral_bandwidth_var': bandwidth.var(),
            'rolloff_mean': rolloff.mean(),
            'rolloff_var': rolloff.var(),
            'zero_crossing_rate_mean': zcr.mean(),
            'zero_crossing_rate_var': zcr.var(),
            'tempo': tempo
        }

        for i in range(20):
            features[f'mfcc{i+1}_mean'] = mfcc[i].mean()
            features[f'mfcc{i+1}_var'] = mfcc[i].var()

        return features

    except Exception as e:
        print(f"Помилка з {file_path}: {e}")
        return None

#%%
from concurrent.futures import ThreadPoolExecutor,
as_completed
from tqdm import tqdm
import pandas as pd
import os

def preprocess_dataset_parallel():
    tasks = []
    for genre in genres:
        genre_path = os.path.join(dataset_path, genre)
        files = os.listdir(genre_path)
        for file in files:
            file_path = os.path.join(genre_path, file)
            tasks.append((file_path, genre))

    results = []
    with ThreadPoolExecutor() as executor:
```

```
        futures = [executor.submit(extract_feature,
file_path, genre) for file_path, genre in tasks]

        for future in tqdm(as_completed(futures),
total=len(futures), desc="Обробка треків"):
            result = future.result()
            if result:
                results.append(result)

    df = pd.DataFrame(results)
    df.to_csv("genre_features.csv", index=False)
    print("Збережено у genre_features.csv")

preprocess_dataset_parallel()

#%%
df = pd.read_csv('genre_features.csv')

df.head()
#%%
df_cleared = df.copy()
df_cleared = df_cleared.drop(columns=['filename'])

X = df_cleared.drop(columns=['genre'])
y = df_cleared['genre']
#%%
X.columns
#%%
y
#%%
X
#%%
y
#%%
X['tempo'] = X['tempo'].apply(lambda x:
float(str(x).strip("[]")) if isinstance(x, str) else x)

scaler = MinMaxScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

joblib.dump(scaler, 'scaler.joblib')
#%%
X_scaled
#%%
encoded_y = LabelEncoder().fit_transform(y)
#%%
callbacks = [
    EarlyStopping(
        monitor='val_loss',
        patience=20,
        restore_best_weights=True,
        verbose=1
```

```
    )
]
#%%
model = keras.Sequential([
    keras.Input(shape=(53,)),
    keras.layers.Dense(512, activation="relu"),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(256, activation="relu"),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10, activation='softmax'),
])

model.compile(optimizer=Adam(learning_rate=0.0001),
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
#%%
X_train, X_val, y_train, y_val = train_test_split(X_scaled,
encoded_y, test_size=0.25, random_state=42,
stratify=encoded_y)
#%%
history = model.fit(X_train, y_train,
validation_data=(X_val, y_val), epochs=500, batch_size=128,
callbacks=callbacks)
#%%
preds = model.predict(X_val)
preds = np.argmax(preds, axis=1)
confusion_matrix(y_val, preds)
#%%
model.save('genre_model.keras')
#%%
import matplotlib.pyplot as plt

best_epoch = callbacks[0].best_epoch

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation
Loss')
plt.axvline(x=best_epoch, color='r', linestyle='--',
label=f'Best Epoch ({best_epoch})')
plt.title('Loss over epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train
```

```
Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
plt.axvline(x=best_epoch, color='r', linestyle='--',
label=f'Best Epoch ({best_epoch})')
plt.title('Accuracy over epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

Файл «Genre/XGBoosting.ipynb»

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder,
StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
classification_report
import pandas as pd
from xgboost import XGBClassifier
import numpy as np
import joblib
#%%
df = pd.read_csv('features_30_sec.csv')

df.head()
#%%
df_cleared = df.copy()
df_cleared = df_cleared.drop(columns=['filename', 'length'])

X = df_cleared.drop(columns=['label'])
y = df_cleared['label']
#%%
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
#%%
label_encoder = LabelEncoder()

encoded_y = label_encoder.fit_transform(y)

joblib.dump('scaler.joblib', scaler)
#%%
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, encoded_y,
    test_size=0.25,
    random_state=42,
    stratify=encoded_y
)
#%%
```

```
rf_model = RandomForestClassifier(
    n_estimators=250,
    max_depth=25,
    n_jobs=-1
)
#%%
rf_model.fit(X_train, y_train)
#%%
y_pred = rf_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"\nТочність RF: {accuracy:.4f}")

print("\nЗвіт по класифікації:")
print(classification_report(y_test, y_pred,
target_names=label_encoder.classes_))
#%%
xgb_model = XGBClassifier(n_estimators=1000)
#%%
xgb_model.fit(X_train, y_train)
#%%
y_pred = xgb_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"\n Точність XGBoost: {accuracy:.4f}")
print(classification_report(y_test, y_pred,
target_names=label_encoder.classes_))
```

Файл «Recommendation System/knn_train.ipynb»

```
import librosa
import numpy as np
import pandas as pd
#%%
import os
import librosa
import numpy as np
import pandas as pd
from tqdm import tqdm
from concurrent.futures import ThreadPoolExecutor

def extract_audio_features(file_path, file_name):
    y, sr = librosa.load(file_path, sr=None)

    stft = librosa.stft(y)

    chroma = librosa.feature.chroma_stft(S=np.abs(stft),
sr=sr, hop_length=512)
    rms = librosa.feature.rms(S=stft)
    centroid =
librosa.feature.spectral_centroid(S=np.abs(stft), sr=sr)
    bandwith =
```

```
librosa.feature.spectral_bandwidth(S=np.abs(stft), sr=sr)
    rolloff =
librosa.feature.spectral_rolloff(S=np.abs(stft), sr=sr)
    zcr = librosa.feature.zero_crossing_rate(y)
    tempo, _ = librosa.beat.beat_track(y=y, sr=sr)

    features = {
        'filename': file_name,
        'chroma_stft_mean': chroma.mean(),
        'chroma_stft_var': chroma.var(),
        'rms_mean': rms.mean(),
        'rms_var': rms.var(),
        'spectral_centroid_mean': centroid.mean(),
        'spectral_centroid_var': centroid.var(),
        'spectral_bandwidth_mean': bandwith.mean(),
        'spectral_bandwidth_var': bandwith.var(),
        'rolloff_mean': rolloff.mean(),
        'rolloff_var': rolloff.var(),
        'zero_crossing_rate_mean': zcr.mean(),
        'zero_crossing_rate_var': zcr.var(),
        'tempo': tempo,
    }

    return features
#%%
import os
import librosa
import numpy as np
import pandas as pd
from tqdm import tqdm
from concurrent.futures import ThreadPoolExecutor

def extract_audio_features(file_path, file_name):
    y, sr = librosa.load(file_path, sr=None)

    stft = librosa.stft(y)

    chroma = librosa.feature.chroma_stft(S=np.abs(stft),
sr=sr, hop_length=512)
    rms = librosa.feature.rms(S=stft)
    centroid =
librosa.feature.spectral_centroid(S=np.abs(stft), sr=sr)
    bandwith =
librosa.feature.spectral_bandwidth(S=np.abs(stft), sr=sr)
    rolloff =
librosa.feature.spectral_rolloff(S=np.abs(stft), sr=sr)
    zcr = librosa.feature.zero_crossing_rate(y)
    tempo, _ = librosa.beat.beat_track(y=y, sr=sr)

    features = {
        'filename': file_name,
        'chroma_stft_mean': chroma.mean(),
        'chroma_stft_var': chroma.var(),
```

```
        'rms_mean': rms.mean(),
        'rms_var': rms.var(),
        'spectral_centroid_mean': centroid.mean(),
        'spectral_centroid_var': centroid.var(),
        'spectral_bandwidth_mean': bandwith.mean(),
        'spectral_bandwidth_var': bandwith.var(),
        'rolloff_mean': rolloff.mean(),
        'rolloff_var': rolloff.var(),
        'zero_crossing_rate_mean': zcr.mean(),
        'zero_crossing_rate_var': zcr.var(),
        'tempo': tempo,
    }

    return features

def process_file(file_path):
    file_name = os.path.basename(file_path)
    try:
        return extract_audio_features(file_path, file_name)
    except Exception as e:
        print(f"Помилка з файлом {file_name}: {e}")
        return None

def process_all_audio_files_threaded(data_folder='data',
n_workers=6):
    file_paths = []
    for root, _, files in os.walk(data_folder):
        for file in files:
            file_paths.append(os.path.join(root, file))

    print(f"Знайдено {len(file_paths)} аудіофайлів")

    results = []
    with ThreadPoolExecutor(max_workers=n_workers) as
executor:
        for res in tqdm(executor.map(process_file,
file_paths), total=len(file_paths)):
            if res is not None:
                results.append(res)

    df = pd.DataFrame(results)
    df.to_csv("audio_features.csv", index=False)
    print("Готово. Збережено у audio_features.csv")


process_all_audio_files_threaded('data', n_workers=6)
#%%
df = pd.read_csv("audio_features.csv")
df.head()
#%%
df['tempo'] = df['tempo'].apply(lambda x:
float(str(x).strip("[]")) if isinstance(x, str) else x)
df['filename'] = df['filename'].apply(lambda x:
```

```
x.replace('.mp3', ''))

X = df.drop(columns=['filename'])
y = df['filename']
#%%
from sklearn.preprocessing import StandardScaler
import joblib

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
joblib.dump(scaler, 'scaler.joblib')
#%%
from sklearn.neighbors import NearestNeighbors

knn = NearestNeighbors(n_neighbors=5, metric='cosine')
knn.fit(X_scaled)
joblib.dump(knn, 'knn_model.joblib')
#%%
joblib.dump(y, 'labels.joblib')
#%%
def recommend_for_new_track(path_to_audio, top_k=3):
    new_features = extract_audio_features(path_to_audio,
os.path.basename(path_to_audio))
    new_df = pd.DataFrame([new_features])
    new_X = new_df[X.columns]
    new_X_scaled = scaler.transform(new_X)

    distances, indices = knn.kneighbors(new_X_scaled,
n_neighbors=top_k)
    return [(y.iloc[i], round(1 - d, 4)) for i, d in
zip(indices[0], distances[0])]

#%%
recommendations = recommend_for_new_track("17.flac")
for name, score in recommendations:
    print(f"{name} (подібність: {score})")
```