

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

«До захисту допущено»

Завідувач кафедри

____ Віталій РОМАНКЕВИЧ
(підпис)

“ ____ ” червня 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою

«Системне програмування та спеціалізовані комп’ютерні системи»

123 «Комп’ютерна інженерія»

на тему: Комп’ютерна система розпізнавання музикального ряду

Виконав: студент IV курсу, групи КВ-11

Лабазов Володимир Володимирович

(підпис)

Керівник асистент каф. СПСКС, к.т.н.

Павловський Володимир Ілліч

(підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н.

Клятченко Ярослав Михайлович

(підпис)

Рецензент доцент каф. ОТ, к.т.н., доц.

Олександр ВЕРБА

(підпис)

Засвідчую, що у цьому дипломному
проєкті немає запозичень з праць інших
авторів без відповідних посилань.

Студент

(підпис)

Київ – 2025 рік

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Освітньо-професійна програма

**«Системне програмування та спеціалізовані
комп'ютерні системи»**

Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач кафедри

____ Віталій РОМАНКЕВИЧ
(підпис)

“ ____ ” червня 2025 р.

ЗАВДАННЯ

на дипломний проєкт студента

Лабазова Володимира Володимировича

1. Тема проєкту «Комп'ютерна система розпізнавання музикального ряду»,
керівник проєкту асистент каф. СПСКС, д.ф.

Павловський Володимир Ілліч,

затверджені наказом по університету від «_» травня 2025 р.

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту див. Технічне завдання

4. Зміст пояснювальної записки:

- 1) Аналіз існуючих систем розпізнавання музикального ряду
- 2) Проєктування і розробка системи розпізнавання музикального ряду
- 3) Тестування системи розпізнавання музикального ряду

5. Перелік графічного матеріалу:

- 1) Структурна схема системи
- 2) Діаграма класів системи

- 3) Алгоритм розпізнавання вхідного аудіофрагменту
- 4) Алгоритм додавання нової музичної композиції до бази індексів

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я. М., доцент кафедри СПіСКС, к.т.н., доцент		

7. Дата видачі завдання «31» квітня 2025 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення літератури за тематикою роботи		
2	Розроблення та узгодження технічного завдання		
3	Аналіз існуючих рішень		
4	Розроблення структури додатку		
5	Програмна реалізація додатку		
6	Розроблення дизайну додатку		
7	Тестування додатку		
8	Підготовка матеріалів першого розділу дипломного проєкту		
9	Підготовка матеріалів другого розділу дипломного проєкту		
10	Підготовка матеріалів третього розділу дипломного проєкту		
11	Підготовка матеріалів графічної частини проєкту		
12	Оформлення технічної документації проєкту		

Студент

Володимир ЛАБАЗОВ

Керівник проєкту

Володимир ПАВЛОВСЬКИЙ

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (приблизно 49 с., 30 рис., 4 додатки).

Об'єкт розробки – створення комп'ютерної системи розпізнавання аудіофайлів за коротким аудіофрагментом із реалізацією настільного застосунку.

Комп'ютерна система дозволяє: автоматично визначати аудіофайл за звуковим фрагментом; обробляти аудіосигнал для витягування ключових ознак; здійснювати швидкий пошук збігів у базі даних; відображати результати розпізнавання у зручному графічному інтерфейсі. Для написання застосунку було обрано мову програмування Python, для аналізу аудіо використано бібліотеку librosa, для пошуку збігів - фреймворк векторного пошуку FAISS, для побудови GUI - Tkinter.

В ході розробки:

- проведено аналіз існуючих систем розпізнавання аудіофайлів та технологій фінгерпринтингу;
- сформульовано вимоги до програмного забезпечення та побудовано його архітектуру;
- реалізовано алгоритми обробки аудіо та витягування спектральних ознак;
- створено векторну базу даних аудіотреків та реалізовано механізм пошуку збігів;
- розроблено інтерфейс користувача для роботи з програмою;
- здійснено тестування системи на прикладній вибірці та оцінено її точність.

Використання даного застосунку дозволяє розпізнавати аудіофайли без підключення до мережі Інтернет та може бути адаптоване для подальшого використання у сфері мобільних додатків або вбудованих рішень.

Ключові слова: розпізнавання аудіо, аудіофінгерпринтинг, librosa, faiss, desktop gui, python.

ANNOTATION

The qualification project includes an explanatory report (approximately 49 pages, 30 figures, 4 appendices).

Object of development – creation of a computer system for recognizing audio files based on a short audio fragment, implemented as a desktop application.

The computer system enables: automatic identification of an audio file by a sound excerpt, audio signal processing to extract key features, fast matching search within a database, and visualization of recognition results through a user-friendly graphical interface. The application was developed in the Python programming language. For audio analysis, the librosa library was used, for similarity search - the FAISS vector search framework, and for GUI construction - Tkinter.

During development, the following steps were taken:

- analysis of existing audio recognition systems and audio fingerprinting technologies;
- formulation of software requirements and architecture design;
- implementation of algorithms for audio processing and spectral feature extraction;
- creation of a vector-based audio database and development of a matching search mechanism;
- development of a graphical user interface for system interaction;
- testing of the system on an applied dataset and evaluation of its accuracy.

The use of this application enables audio recognition without requiring an Internet connection and can be adapted for future deployment in mobile applications or embedded solutions.

Keywords: audio recognition, audio fingerprinting, librosa, faiss, desktop gui, python.

[illegible]

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ	2
2. ПІДСТАВА ДЛЯ РОЗРОБКИ	2
3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ	2
4. ДЖЕРЕЛА РОЗРОБКИ	2
5. ТЕХНІЧНІ ВИМОГИ	3
5.1 Вимоги до програмного продукту, що розробляється	3
5.2 Вимоги до апаратного забезпечення	3
5.3 Вимоги до програмного та апаратного забезпечення користувача	3
6. ЕТАПИ РОЗРОБКИ	4

					ІАЛЦ.045480.002 ТЗ			
Зм	Лист	№ докум.	Підп.	Дата				
Розроб.		Лабазов В. В.			Комп'ютерна система розпізнавання музикального ряду Технічне завдання	Літ.	Аркуш	Аркушів
Перев.		Павловський В.І.					1	4
						КПІ ім. Ігоря Сікорського, ФПМ, КВ-11		
Н. контр.		Клятченко Я. М.						
Зав. Каф.		Романкевич В. О.						

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Комп'ютерна система розпізнавання музикального ряду».

Галузь застосування: Розпізнавання аудіофайлів за наданими фрагментами.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є створення застосунку для розпізнавання аудіо за наданими фрагментами зі зручним у користуванні графічним інтерфейсом.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

					ІАЛЦ.045480.002 ТЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		2

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до програмного продукту, що розробляється

- Сумісність із операційною системою Windows та підтримка популярних аудіоформатів (MP3);
- Автоматичне розпізнавання аудіофайлу за наданим фрагментом;
- Функціонал додавання та видалення нових аудіофайлів до/з бази індексів;
- Створення базових звітів;

5.2 Вимоги до апаратного забезпечення

- Процесор: Intel Core i5 9300H і вище;
- Оперативна пам'ять: мінімум 8 Гб;

5.3 Вимоги до програмного та апаратного забезпечення користувача

- Операційна система Windows;

					ІАЛЦ.045480.002 ТЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		3

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проекту	17.04.2025
2.	Розроблення та узгодження технічного завдання	30.04.2025
3.	Аналіз існуючих рішень	05.05.2025
4.	Підготовка математичної частини дипломного проекту	10.05.2025
5.	Підготовка графічної частини дипломного проекту	18.05.2025
6.	Підготовка матеріалів першого розділу дипломного проекту	19.05.2025
7.	Підготовка матеріалів другого розділу дипломного проекту	20.05.2025
8.	Підготовка матеріалів третього розділу дипломного проекту	22.05.2025
9.	Оформлення документації дипломного проекту	25.05.2025
10.	Попередній огляд матеріалів диплому на кафедрі	30.05.2025

Пояснювальна записка
до дипломного проєкту
на тему: «Комп'ютерна система розпізнавання музикального ряду»

Київ – 2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ.....	3
ВСТУП.....	5
1. АНАЛІЗ ІСНУЮЧИХ СИСТЕМ РОЗПІЗНАВАННЯ МУЗИКАЛЬНОГО РЯДУ.....	6
1.1 Загальна інформація про системи розпізнавання аудіо.....	6
1.2 Аналіз існуючих систем та обґрунтування теми диплому.....	7
1.2.1 Огляд системи Shazam.....	7
1.2.2 Огляд системи ACRCLOUD.....	8
1.2.3 Обґрунтування теми диплому.....	9
2.ПРОЄКТУВАННЯ І РОЗРОБКА СИСТЕМИ РОЗПІЗНАВАННЯ МУЗИКАЛЬНОГО РЯДУ.....	12
2.1 Вимоги до системи.....	12
2.2 Вибір інструментів та бібліотек.....	13
2.3 Розробка структури настільного застосунку.....	14
2.4 Витягування спектральних ознак аудіосигналу.....	16
2.5 Формування векторної бази та індексація аудіофайлів.....	23
2.6 Реалізація механізму пошуку збігів.....	28
2.7 Розробка графічного інтерфейсу користувача.....	30
3. ТЕСТУВАННЯ СИСТЕМИ РОЗПІЗНАВАННЯ МУЗИКАЛЬНОГО РЯДУ..	38
3.1 Тестування розпізнавання системою аудіо без спотворень.....	38
3.2 Тестування розпізнавання системою спотвореного аудіо.....	40
3.3 Тестування інтерфейсу системи.....	43
ВИСНОВКИ.....	47
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	49

					ІАЛЦ.045480.003ПЗ									
Зм	Лист	№ докум.	Підп.	Дата										
Розроб.		Лабазов В.В.			Комп'ютерна система розпізнавання музикального ряду Пояснювальна записка					Літ.	Лист	Листів		
Перев.		Павловський В.І.										1	49	
										НТУУ «КПІ ім. І. Сікорського»,ФПМ,КВ-11				
Н. контр.		Клятченко Я.М.												
Затв.		Романкевич В.О.												

ДОДАТКИ

Додаток 1. ІАЛЦ.045480.004 Д1. Структурна схема системи

Додаток 2. ІАЛЦ.045480.005 Д2. Діаграма класів системи

Додаток 3. ІАЛЦ.045480.006 Д3. Алгоритм розпізнавання вхідного
аудіофрагменту

Додаток 4. ІАЛЦ.045480.007 Д4. Алгоритм додавання нової музичної
композиції до бази індексів

					ІАЛЦ.045480.003 ПЗ	Лист
						2
Зм	Лист	№ докум.	Підп.	Дата		

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

MFCC (Mel-Frequency Cepstral Coefficients) - коефіцієнти кепстрального представлення сигналу, що відображають енергетичні характеристики спектра у шкалі мел-частот.

GUI (Graphical User Interface) - графічний інтерфейс користувача, що забезпечує взаємодію користувача з програмним забезпеченням у візуальній формі.

WAV, MP3, FLAC - формати зберігання аудіоінформації з різними рівнями стискування.

Евклідова відстань (L2) - метрика відстані між двома векторами у просторі ознак, яка обчислюється як квадратний корінь із суми квадратів різниць координат.

RMS (Root Mean Square) - коренева середньоквадратична амплітуда, яка є мірою локальної енергії сигналу у заданому часовому вікні.

Бін - окремий елемент частотного спектра, що відповідає певному діапазону частот у результаті перетворення Фур'є.

Висота тону (pitch) - акустичний параметр, що визначає сприйняття звуку як вищого або нижчого; пов'язаний із частотою основного тону.

Вектор ознак - математичне представлення аудіофайлу у вигляді числового вектора, що відображає його найхарактерніші властивості.

Аудіофінгерпринтинг - метод ідентифікації аудіосигналу шляхом побудови унікального набору ознак (відбитка, фінгерпринта), що дозволяє зіставляти фрагменти аудіо з базою даних для пошуку збігів.

Нормалізація сигналу - попередня обробка аудіо, яка зменшує вплив різниці в гучності або амплітуді між різними записами.

Спектрограма(spectrogram) - візуальне представлення частотної структури аудіосигналу з часом (частота $\times$ час $\times$ амплітуда).

Частота дискретизації(sampling rate) - кількість вимірів (семплів) звуку за секунду, що впливає на якість аналізу.

ВСТУП

У сучасному цифровому середовищі обробка та розпізнавання аудіоінформації набувають усе більшого значення. Завдяки розвитку технологій машинного навчання, комп'ютерного слуху та цифрової обробки сигналів стали можливими системи, які здатні автоматично визначати джерело музичного фрагмента.

Такі системи розпізнавання знаходять широке застосування в мобільних застосунках, інтегруються в побутові пристрої, а також використовуються в автоматизованих системах обліку контенту та авторських прав. Проте більшість доступних рішень є комерційними, закритими для модифікації або обмеженими у використанні без доступу до мережі Інтернет.

Цей дипломний проєкт присвячений розробці настільного програмного забезпечення для розпізнавання музичних композицій за коротким аудіофрагментом. Основна мета - створення автономної системи, яка не потребує сторонніх сервісів, працює локально та має відкриту архітектуру.

					ІАЛЦ.045480.003 ПЗ	Лист
						5
Зм	Лист	№ докум.	Підп.	Дата		

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Загальна інформація про системи розпізнавання аудіо

Сучасні системи автоматичного розпізнавання аудіосигналів знаходять дедалі ширше застосування у найрізноманітніших галузях - від цифрових медіасервісів до безпеки, моніторингу контенту, авторського права та музичної індустрії. У своїй основі подібні системи покладаються на методи цифрової обробки сигналів та алгоритми машинного пошуку, які дають змогу ідентифікувати аудіофрагмент навіть за умов спотворення, наявності шуму або неповного представлення.

Загальний принцип роботи системи розпізнавання звуку полягає у тому, що записаний аудіофрагмент проходить низку етапів: попередню обробку, витягування ознак, формування унікального цифрового відбитка (фінгерпринта), після чого відбувається пошук збігу у базі відомих зразків. Попередня обробка сигналу передбачає нормалізацію, фільтрацію шуму, а також виявлення найбільш інформативної частини запису - тієї, яка не містить тиші або сторонніх звуків. На етапі витягування ознак, сигнал перетворюється у набір числових характеристик (вектор ознак), що відображають його спектральну структуру, енергію, частотний розподіл тощо.

Завдяки цьому можливо перетворити будь-який уривок звуку на компактне математичне представлення, яке можна ефективно порівнювати з великою базою раніше збережених зразків. Типовими методами тут виступають коефіцієнти MFCC, спектральні гістограми, хроматичні ознаки, спектральне спадання та інші статистики, які узагальнюють аудіоінформацію у часово-частотному просторі.

Після формування вектору ознак, він передається до модуля пошуку, де відбувається порівняння з базою шляхом обчислення міри подібності. У

практичних реалізаціях це зазвичай виконується через евклідову відстань (L2-норма), косинусну подібність або інші метричні підходи. Висока ефективність пошуку досягається за рахунок використання спеціалізованих індексів, таких як FAISS, які забезпечують швидкий відбір найближчих кандидатів у багатовимірному просторі ознак.

Кінцевою метою системи є визначення найбільш імовірного збігу серед наявних у базі представлень. Цей збіг супроводжується додатковими метаданими (назва композиції, виконавець, альбом тощо), які можуть бути надані користувачеві як результат розпізнавання.

Подібні системи активно інтегруються у мобільні застосунки, стрімінгові сервіси, інструменти моніторингу радіо- і телеефіру, а також у спеціалізовані програми для журналістики, наукових досліджень та архівування звуку. Їхнє широке впровадження стало можливим завдяки розвитку цифрових технологій, зростанню обчислювальних потужностей і відкритості інструментів цифрової обробки сигналів.

1.2. Аналіз існуючих систем та обґрунтування теми диплому

На ринку існує кілька потужних рішень для розпізнавання аудіо. Деякі з них є комерційними (наприклад, Shazam, ACRCLOUD), інші - відкриті та призначені для наукових досліджень (наприклад, dejavu, audfprint). Розглянемо найвідоміші з них.

1.2.1. Огляд системи Shazam

Однією з найвідоміших і наймасовіших систем ідентифікації музики є Shazam - комерційна платформа, що використовує алгоритми фінгерпринтингу для швидкого розпізнавання аудіофрагментів. Її алгоритм

був представлений у науковій публікації Ейвері Ванга у 2003 році та з того часу продемонстрував ефективність у роботі з короткими, зашумленими або зміщеними сигналами.

Основою методу є побудова спектрограми аудіосигналу з подальшим виявленням спектральних піків - локальних максимумів у часово-частотному просторі. Далі ці піки об'єднуються у стійкі пари, для кожної з яких фіксується частотна пара та часова різниця між ними. Таким чином формується компактний, але інформативний набір ознак, який є стійким до шуму, реверберації та змін амплітуди. Цей набір надалі використовується як цифровий відбиток аудіофрагмента.

Для розпізнавання створений фінгерпринт порівнюється з великою централізованою базою, яка містить мільйони записів. У разі знаходження збігу визначається назва композиції, автор, альбом тощо. Завдяки унікальній структурі ознак та високій швидкості пошуку система здатна ідентифікувати музику навіть у складних акустичних умовах і за тривалість прослуховування в межах кількох секунд.

Проте, варто зазначити, що Shazam є закритим комерційним сервісом. Користувач не має доступу до механізмів формування ознак або до бази даних, а сама система працює виключно через інтернет-з'єднання з серверами компанії. Відсутність можливості додавати власні зразки або змінювати логіку обробки сигналу робить її непридатною для дослідницького використання або створення автономних рішень.

1.2.2. Огляд системи ACRCcloud

Ще одним прикладом потужної системи розпізнавання аудіо є ACRCcloud - платформа, що орієнтована переважно на комерційне використання та інтеграцію з іншими продуктами. На відміну від Shazam, ця система пропонує ширші можливості для розробників, надаючи інтерфейси

					ІАЛЦ.045480.003 ПЗ	Лист 8
Зм	Лист	№ докум.	Підп.	Дата		

API, SDK, а також доступ до створення приватних баз даних. Це дозволяє реалізувати розпізнавання не лише на основі глобальної колекції записів, але й у межах локального сховища користувача.

ACRCloud застосовується в телевізійному моніторингу, рекламі, радіоаналітиці, інтеграції з музичними платформами та мобільними додатками. Основна ідея роботи полягає у витягуванні фінгерпринтів із вхідного сигналу з подальшим порівнянням у хмарному середовищі. Алгоритми компанії базуються на аналізі спектрального вмісту та часової структури сигналу, але детальна реалізація є закритою і недоступною для зовнішніх розробників.

Платформа підтримує кілька режимів розпізнавання: ідентифікацію музики, трансляцій, фонового середовища, а також голосових подій. Для кожного режиму застосовуються оптимізовані методи обробки та пошуку, що дозволяє досягти високої точності навіть за несприятливих акустичних умов. Важливою перевагою є можливість роботи з потоковим аудіо, що розширює сферу застосування системи у реальному часі.

Попри свою функціональність, ACRCloud залишається закритою системою. Всі обчислення відбуваються на серверах компанії, а для доступу до основних можливостей необхідна реєстрація та оплата тарифного плану. Неможливість повністю контролювати алгоритм пошуку або змінювати логіку витягування ознак обмежує використання цієї системи в академічному середовищі або у випадках, коли потрібна повна автономність. Водночас, наявність відкритої документації та зручних засобів інтеграції робить ACRCloud привабливим рішенням для комерційних проєктів із чітко визначеною інфраструктурою та доступом до мережі.

1.2.3. Обґрунтування теми диплому

Аналіз існуючих рішень у сфері розпізнавання музичних аудіофрагментів показав, що попри високу ефективність комерційних

					ІАЛЦ.045480.003 ПЗ	Лист 9
Зм	Лист	№ докум.	Підп.	Дата		

платформ, таких як Shazam та ACRCLOUD, вони мають суттєві обмеження для користувачів, які прагнуть створити власні автономні або дослідницькі рішення. Обидві згадані системи функціонують у межах закритої екосистеми, вимагають постійного доступу до Інтернету та не дозволяють змінювати внутрішню логіку алгоритмів або структуру збереження аудіоданих. Це суперечить вимогам задач, що передбачають повну локальну обробку, гнучкість модифікацій та можливість інтеграції з іншими інструментами.

У зв'язку з цим виникає актуальна потреба в реалізації відкритої комп'ютерної системи, яка здатна розпізнавати музичні фрагменти в автономному режимі, без залучення сторонніх хмарних сервісів. Така система повинна забезпечувати можливість попередньої обробки аудіосигналу, витягування спектральних ознак, ефективного порівняння з наявною базою та формування результату, зрозумілого для користувача. Важливим є також забезпечення модульності та розширюваності архітектури: користувач повинен мати змогу змінювати параметри обробки сигналу, додавати або видаляти аудіофайли з бази, тестувати різні підходи до нормалізації та пошуку подібності.

Окрему увагу слід приділити візуалізації результатів. На відміну від типових рішень, де результати представлені лише у текстовому вигляді, у запропонованій системі має бути реалізовано відображення графіків спектра, енергетичного профілю та порівняння сигналів. Це не лише покращує взаємодію з користувачем, але й сприяє глибшому розумінню того, як відбувається процес розпізнавання. Таким чином, система набуває не лише прикладного, а й навчального або дослідницького потенціалу.

Обрана тема дипломної роботи дозволяє поєднати інструменти цифрової обробки сигналів, алгоритми машинного пошуку та створення користувацького інтерфейсу. Результатом має стати програмний продукт, здатний виконувати розпізнавання музичних уривків із заданим рівнем точності, зберігаючи при цьому автономність, прозорість структури та

можливість розширення. Це робить проєкт релевантним не лише з наукової точки зору, але й з позиції практичного використання в умовах реального середовища.

2. ПРОЄКТУВАННЯ І РОЗРОБКА СИСТЕМИ

2.1. Вимоги до системи

Під час проєктування системи розпізнавання музичного ряду було сформульовано низку вимог, що охоплюють як функціональні, так і нефункціональні аспекти.

З функціональної точки зору система повинна мати можливість завантаження аудіофрагментів, попередню обробку сигналу, витягування спектральних ознак, пошук збігів у базі аудіокомпозицій, а також виведення результату пошуку. Алгоритм повинен забезпечувати стабільну роботу при зміні швидкості відтворення, висоти тону, при наявності шумів чи записі з пристроїв низької якості, таких як диктофони або мобільні телефони. Також, однією з ключових вимог до системи є автоматична побудова та оновлення векторного індексу бази даних аудіофайлів. Це дозволяє користувачу змінювати і формувати свою базу музичних композицій для різних цілей.

Застосунок має бути десктопним, працювати без інтернет-з'єднання, бути максимально автономним та не залежати від сторонніх API або хмарних сервісів.

Вимоги до інтерфейсу включають простоту у користуванні, наявність виводу інформативного результату, індикацію процесу розпізнавання та можливість перегляду статистики (наприклад, графіки спектрограми або сигналу). Інтерфейс не повинен вимагати спеціальної підготовки або глибоких технічних знань від користувача.

2.2. Вибір інструментів та бібліотек

Для розробки системи розпізнавання музикального ряду потрібно було обрати інструменти та бібліотеки, які б забезпечили баланс між ефективністю, зручністю реалізації та точністю. Враховуючи вимоги до платформи, обробки сигналів, векторної індексації та реалізації графічного інтерфейсу, було вирішено використовувати мову програмування Python як найбільш гнучку та багату на спеціалізовані пакети у сфері аналізу аудіо.

Для обробки аудіо та видобування ознак використано бібліотеку librosa. Вона надає широкий набір інструментів для цифрової обробки сигналів, зокрема MFCC, та багатьох інших показників, які є основою для побудови векторного представлення аудіофайлів. Висока точність та активна підтримка бібліотеки роблять її стандартом де-факто для наукової спільноти, що працює з аудіо.

Для реалізації пошуку серед векторів ознак було обрано бібліотеку FAISS (Facebook AI Similarity Search). Цей інструмент спеціалізується на високопродуктивному пошуку по великих масивах векторів, використовуючи ефективні структури даних. У реалізації даного застосунку використовується варіант IndexFlatL2, який забезпечує точний пошук за евклідовою відстанню (L2) та працює швидко при помірних обсягах даних, що є прийнятним на етапі прототипування.

Для збереження метаданих (назв аудіофайлів, векторів ознак) застосовано модуль pickle, який дозволяє зберігати складні об'єкти Python у бінарному вигляді. Для побудови графічного інтерфейсу було обрано Tkinter - стандартну бібліотеку для створення GUI у Python. Її головними перевагами є простота, кросплатформеність, інтеграція з Python без додаткових залежностей, а також достатній набір компонентів для реалізації інтерфейсу з кнопками, вікнами повідомлень, полями вводу та візуалізацією результатів.

Візуалізація спектрального аналізу та сигналу реалізована за допомогою бібліотеки Matplotlib. Вона дозволяє виводити графіки в інтерактивному або статичному режимі без необхідності використання окремих вікон, що інтегрується у загальну структуру застосунку.

Усі обчислювальні операції відбуваються у фоновому потоці, реалізованому через стандартний модуль threading. Це дозволяє не блокувати графічний інтерфейс і зберігати плавність взаємодії з користувачем навіть під час тривалих операцій.

2.3. Розробка структури настільного застосунку

Структура настільного застосунку побудована за модульним принципом, що дозволяє чітко розмежувати зони відповідальності між компонентами. Це рішення забезпечує зручність розробки, простоту підтримки коду та дає можливість гнучкого розширення функціональності в подальшому. Архітектура системи складається з трьох взаємопов'язаних частин: інтерфейсної оболонки, аналітичного модулю та векторної бази.

Інтерфейсна оболонка є посередником між користувачем та внутрішньою логікою застосунку. Вона відповідає за відображення результатів, обробку подій, ініціацію дій розпізнавання, додавання чи видалення аудіофайлів з бази, перемикання тем оформлення. Інтерфейс реалізований у вигляді окремого графічного компонента і не містить жодної обробки аудіосигналу. Його основне завдання - забезпечити доступ до функцій розпізнавання через зручне середовище.

```
32 self.core = MusicRecognizerCore()
```

Рисунок 2.1 - Зв'язок модуля інтерфейсу з аналітичним модулем

На рисунку 2.1 зображено зв'язок інтерфейсного модулю з логічним, через створення об'єкту класу ядра при старті програми. Таким чином, на рівні GUI забезпечується лише обробка подій (натискання кнопок), запуск потоку обробки аудіофайлу, відображення результатів - зокрема текстових, графіків спектрограм, амплітудних форм та L2-розподілу, а всю логіку розташовано в розпізнавальній частині.

Аналітична частина є основною обчислювальною компонентою застосунку. Сюди входить попередня обробка сигналу, фільтрація шумів, вирівнювання гучності, а також витягування спектральних ознак з аудіофрагментів. Отримані ознаки нормалізуються та розбиваються на окремі блоки, які формують багатовимірні вектори. Ці вектори є представленням характерних властивостей аудіозапису у вигляді, придатному для математичного порівняння. У складі розпізнавального модулю реалізовано також пошук найближчих відповідників у векторному просторі, що здійснюється з використанням L2-відстані як метрики подібності. При порівнянні додатково застосовуються методи агрегації результатів, зокрема з урахуванням статистичних критеріїв як середнє значення відстаней, кількість збігів та часову варіацію.

Третім ключовим компонентом є векторна база. На відміну від класичних підходів з табличними базами даних, тут використовується система зберігання ознак у вигляді високовимірних векторів, які розміщуються у спеціалізованому індексі. Цей індекс дає змогу ефективно здійснювати пошук найближчих відповідників за обмежений час, навіть у разі великої кількості об'єктів. Векторна база містить як самі вектори ознак, так і відповідні ідентифікатори, що дають змогу пов'язати знайдені збіги з реальними аудіозаписами. Вона зберігається у вигляді серіалізованих файлів, які завантажуються на початку роботи системи та автоматично оновлюються при зміні набору даних.

Уся система функціонує як єдиний злагоджений механізм, у якому кожна частина виконує свою роль незалежно від інших, але взаємодіє через чітко визначені канали. Така дозволяє легко масштабувати систему, наприклад, для написання різних автотестів, чи змінити інтерфейс застосунку на веб-орієнтований чи орієнтований на мобільні пристрої. Це можливо завдяки тому, що інтерфейс лише керує введенням/виведенням, не втручаючись у внутрішню логіку.

2.4. Витягування спектральних ознак аудіосигналу

Ядром системи розпізнавання є побудова компактного та інформативного представлення аудіосигналу у вигляді вектора ознак. Такий вектор дозволяє ефективно порівнювати фрагменти аудіо між собою та порівнювати їх. У проєкті реалізовано витягування спектральних ознак із сигналу за допомогою бібліотеки `librosa`, із подальшою нормалізацією та фільтрацією.

Перед витягуванням ознак сигнал проходить низку підготовчих етапів:

```
22 def enhance_audio(y, sr): 3 usages
23     y = librosa.util.normalize(y)
24     y, _ = librosa.effects.trim(y)
25     y = bandpass_filter(y, sr)
26     y = wiener(y)
27     y = savgol_filter(y, window_length: 9, polyorder: 2)
28     energy = librosa.feature.rms(y=y)[0]
29     frames = np.nonzero(energy > np.percentile(energy, q: 25))[0]
30     if frames.size:
31         y = y[frames[0]*512 : frames[-1]*512]
32     return y
33
```

Рисунок 2.2 - Функція `enhance_audio`

Функція зображена на рис. 2.2 виконує попередню обробку аудіосигналу у (який уже завантажено через `librosa.load`) із частотою дискретизації `sr`.

`librosa.util.normalize(y)` нормалізує амплітуду сигналу, масштабуючи значення масиву у так, щоб найбільше абсолютне значення стало рівним 1. Це дозволяє робити вхідний сигнал незалежним від гучності.

`librosa.effects.trim(y)` виконує автоматичне видалення фрагментів низької енергії на початку та в кінці сигналу. В основі цього методу лежить порівняння локальної енергії вікна з певним порогом, визначеним за допомогою перцентилів або фіксованих значень. Вікно переміщується по сигналу, і як тільки середня амплітуда у вікні перевищує поріг, ця точка вважається початком значущого аудіо. Аналогічно визначається і кінець сигналу. Таким чином, усі малозначущі ділянки тиші відсікаються.

```
12 def bandpass_filter(y, sr, low=80, high=8000): 1 usage
13     sos = scipy.signal.butter(6, [low, high], btype='bandpass', fs=sr, output='sos')
14     return scipy.signal.sosfilt(sos, y)
```

Рисунок 2.3 - Функція `bandpass_filter`

На рисунку 2.3 зображено функцію `bandpass_filter(y, sr)`, яка накладає смуговий фільтр, що пропускає частоти у заданому діапазоні. Створюється фільтр Баттерворта 6-го порядку (використовуючи `scipy.signal.butter`). Фільтр Баттерворта відомий тим, що має плавну амплітудну характеристику без різких перепадів. Це означає, що перехід між пропущеними й приглушеними частотами відбувається поступово, що зменшує ймовірність спотворень. Фільтр Баттерворта у даній системі реалізовано не у стандартному вигляді, а через подання у формі Second-Order Sections (SOS), тобто як послідовність фільтрів другого порядку, з'єднаних каскадом. Такий підхід використовується для підвищення чисельної стабільності та зменшення похибок при обробці сигналів. Справа в тому, що фільтри високого порядку (наприклад, 6-го або більше) мають велику кількість коефіцієнтів, і при прямій

реалізації можуть виникати помилки округлення або нестабільність фільтра, особливо при роботі з довгими сигналами або великою частотою дискретизації. Це може призвести до спотворення сигналу, або до того, що результат зовсім не відповідатиме очікуванням. Щоб уникнути цих проблем, весь фільтр ділиться на кілька секцій - кожна з яких є простим фільтром другого порядку, тобто працює з трьома коефіцієнтами в чисельнику та двома в знаменнику (без урахування одиниці). Такі фільтри легше обчислювати і вони менш чутливі до похибок. Кожна така секція має форму:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Де b_i впливають на пропускання сигналу, а a_i визначає характер загасання. Оскільки більшість важливої акустичної інформації лежить в частотному діапазоні від 80 Гц до 8000 Гц, я обрав саме такі параметри для смугового фільтра щоб зменшити шум вхідного фрагмента і знизити обчислювальне навантаження на систему.

wiener(y) застосовує фільтр Вінера для зменшення шуму. базуючись на локальній дисперсії. Фільтр Вінера є адаптивним згладжувальним фільтром, що працює за принципом мінімізації середньоквадратичної помилки між вихідним (відфільтрованим) сигналом та ідеальним (очікуваним) сигналом без шуму. Він не просто приглушує високочастотні складові, як звичайний згладжувальний фільтр, а намагається зберегти важливі структури сигналу (наприклад, переходи, піки) і одночасно приглушити шум. Фільтр оцінює локальні статистики сигналу - зокрема локальну дисперсію (варіацію) - і використовує цю інформацію для адаптації сили згладжування. У тих ділянках сигналу, де варіація низька (тобто мало змін, типовий для шуму фон), фільтр виконує сильніше згладжування. У тих же частинах, де варіація висока (наявність музичних подій, атак інструментів тощо), згладжування мінімізується, щоб зберегти корисну інформацію. Для кожної точки сигналу $\hat{x}_i$,

фільтр оцінює локальну середню μ та дисперсію σ^2 , після чого формує відфільтровану величину за формулою:

$$\hat{x}_i = \mu + \frac{\sigma^2 - v^2}{\sigma^2} (y_i - \mu)$$

Де v^2 це оцінена дисперсія шуму(наприклад, на основі середньої дисперсії в усіх частинах). Цей метод добре підходить для згладжування білого шуму або фонового шуму, що корисно при роботі з аудіофрагментами записаними на побутові пристрої, такі як мобільний телефон.

`savgol_filter(y, 9, 2)` застосовує фільтр Савицького-Голея. Це згладжувальний фільтр, який дозволяє зменшити вплив шуму без суттєвого спотворення основної форми сигналу. Його головна перевага полягає в здатності зберігати важливі характеристики сигналу, такі як амплітудні переходи, піки або мікроструктури. Фільтр працює за принципом локальної апроксимації сигналу: кожна точка сигналу замінюється значенням, яке отримано в результаті апроксимації сусідніх точок поліномом заданого ступеня. У цьому випадку використовується поліном другого ступеня (квадратичний), який підбирається за методом найменших квадратів на фіксованому вікні довжиною 9 точок (симетричне вікно довкола кожної точки). Таким чином, для кожної точки сигналу y_i визначається найкращий (в сенсі похибки апроксимації) поліном вигляду:

$$P(t) = a_0 + a_1t + a_2t^2$$

де t це зсув у межах вікна відносно центральної точки, a_i це коефіцієнти, що обчислюються для кожного вікна окремо. У результаті значення в центрі вікна замінюється значенням цього полінома $P(t)$, тобто відбувається згладжування без зсуву фаз або втрати локальних піків. Це запобігає негативному впливу деформованого чи зсунутого сигналу на точність витягнутих ознак.

`librosa.feature.rms(y=y)` виділяє найбільш енергетично насичену частину сигналу обчисленням енергетичного профілю сигналу. бчислює енергетичний профіль аудіосигналу, тобто визначає кореневу середньоквадратичну амплітуду (RMS) для кожного короткого вікна сигналу. Це значення є надійним показником локальної енергії сигналу та дозволяє оцінити, наскільки "гучним" або енергетично насиченим є конкретний фрагмент звуку. Формально, для вікна з N значеннями сигналу $\{y_1, y_2, \dots, y_N\}$, RMS-енергія обчислюється як:

$$RMS = \sqrt{\frac{1}{N} \sum_{i=1}^N y_i^2}$$

Отримана послідовність RMS значень відповідає енергії кожного короткого сегмента аудіофайлу. Далі система виконує відбір лише найбільш енергетично насичених фрагментів, відсікаючи малозначущі ділянки (наприклад, тишу або фонові шуми). Це реалізовано за допомогою порогового фільтрування у `np.nonzero(energy > np.percentile(energy, 25))[0]`. Тут використовується 25-й перцентиль як поріг: усі вікна, енергія яких нижча за цей поріг, вважаються неінформативними і відкидаються. У результаті залишається лише центральна частина сигналу з підвищеною енергією, яка найімовірніше містить корисні аудіоподії - інструменти, вокал, атаки тощо.

Фінальний обрізаний сигнал формується як ділянка від першого до останнього енергетично значущого вікна, перемноженого на розмір кроку (`frame shift`). Це дозволяє вирівняти динамічний діапазон, що позитивно впливає на точність витягування ознак та результат розпізнавання.

```

31 def extract_robust_features(y, sr): 1usage
32 cqt = librosa.amplitude_to_db(np.abs(librosa.cqt(y=y, sr=sr, n_bins=60, bins_per_octave=12)))
33 cqt_mean = normalize_feature_block(np.mean(cqt, axis=1))
34
35 chroma_cqt = librosa.feature.chroma_cqt(y=y, sr=sr)
36 chroma_mean = normalize_feature_block(np.mean(chroma_cqt, axis=1))
37 chroma_std = normalize_feature_block(np.std(chroma_cqt, axis=1))
38
39 contrast = normalize_feature_block(np.mean(librosa.feature.spectral_contrast(y=y, sr=sr, axis=1)))
40 rolloff = normalize_feature_block(np.mean(librosa.feature.spectral_rolloff(y=y, sr=sr, axis=1)))
41 zcr = normalize_feature_block(np.mean(librosa.feature.zero_crossing_rate(y), axis=1))
42
43 tempogram = librosa.feature.tempogram(y=y, sr=sr)
44 tempo_features = normalize_feature_block(np.mean(tempogram, axis=1))
45
46 combined = np.concatenate([
47     cqt_mean,
48     chroma_mean, chroma_std,
49     contrast,
50     rolloff,
51     zcr,
52     tempo_features
53 ])
54 return combined.astype("float32")

```

Рисунок 2.4 - Функція extract_robust_features

Після попередньої обробки вхідного звукового сигналу з нього витягуються спектральні ознаки, за допомогою функції, що зображена на рисунку 2.4. Спершу обчислюється спектральне зображення сигналу на основі постійного-Q перетворення (CQT). Це перетворення, на відміну від традиційного перетворення Фур'є, має логарифмічну шкалу частот, що краще відповідає музичному сприйняттю - інтервали між нотами у музичній шкалі також логарифмічні. Кожна частотна смуга в CQT має однакове співвідношення центральної частоти до ширини смуги, що описується через параметр:

$$Q = \frac{f_k}{\Delta f_k}$$

де f_k - центральна частота k-го біну, а Δf_k - його ширина. Таким чином, вектор CQT дозволяє відобразити гармонічну структуру сигналу з високою роздільною здатністю на низьких частотах і низькою - на високих.

Окрім CQT, витягуються хроматичні ознаки(chroma) - це 12-вимірний вектор, кожен елемент якого відповідає одному півтону в октаві. Таке подання

є інваріантним до октави: ноти C4 і C5 матимуть схожі хрома-профілі, що дозволяє системі порівнювати схожі мелодії навіть при транспозиції.

Далі обчислюється спектральний контраст - величина, яка визначає різницю між піковими і мінімальними значеннями в різних частотних смугах. Цей параметр особливо корисний для розпізнавання тембрів і різних музичних інструментів, оскільки кожен з них має характерний спектральний малюнок.

Також обчислюється коефіцієнт rolloff, який визначає частоту, нижче якої сконцентровано 85% енергії спектру. Це дозволяє відрізнити "легкі" та "важкі" звуки: наприклад, звуки перкусії чи бас-гітари мають вищі значення rolloff, ніж флейта чи вокал.

Іншою важливою ознакою є швидкість перетину нуля (zero-crossing rate) - кількість разів, коли сигнал змінює знак. Цей параметр добре описує шумність сигналу: шумні фрагменти мають високу частоту переходів через нуль.

Останнім типом ознак є темпограма (tempogram) - спектр автокореляцій ритмічних змін, який дозволяє вловити темп та ритмічну структуру сигналу. Темпограма особливо важлива при розпізнаванні пісень, які мають характерний ритм або барабанний супровід.

Для кожного з описаних спектральних параметрів обчислюються статистичні узагальнення - здебільшого середні значення по часу, а також стандартне відхилення для хроматичних ознак. Це зменшує вплив випадкових коливань і перетворює багатовимірні спектральні матриці у компактні вектори.

```
16 def normalize_feature_block(block): 7 usages
17     return (block - np.mean(block)) / (np.std(block) + 1e-8)
```

Рисунок 2.5 - Функція normalize_feature_block

Перед об'єднанням, ці вектори нормалізуються функцією, що зображена на рисунку 2.5. Така нормалізація забезпечує сталість масштабу ознак, знижує вплив фонових шумів та покращує поведінку системи під час індексації. У фінальному кроці всі спектральні вектори конкатенуються в один єдиний

вектор ознак, який подається на індексацію. Отриманий вектор має фіксовану розмірність (близько 90–100 вимірів) і приводиться до типу float32 для сумісності з FAISS. Такий розмір є збалансованим компромісом між інформативністю та обчислювальною ефективністю.

Цей етап завершує процес трансформації сирого звукового сигналу в структуровану векторну форму, що дозволяє ефективно здійснювати пошук за схожістю та виконує роль "відбитка" (аудіофінгерпринта) музичного фрагмента в системі.

2.5. Формування векторної бази та індексація аудіофайлів

Після того, як аудіосигнал було попередньо оброблено та перетворено на набір спектральних ознак, необхідно сформувати векторне представлення для кожного фрагмента пісні. Це дозволяє надалі проводити швидко та ефективно розпізнавання аудіо за допомогою бібліотеки FAISS.

```
120     def build_index(self): 2 usages
121         feats, names = self.extract_features_for_tracks(self.database, enhance=True)
122         self.features = l2_normalize(feats)
123         self.feature_dim = self.features.shape[1]
124         self.index = faiss.IndexFlatL2(self.feature_dim)
125         self.index.add(self.features)
126         self.track_names = names
127         self.save_index()
```

Рисунок 2.6 - Метод build_index

Процес формування векторної бази починається з виклику методу build_index (рисунок 2.6), який є центральним для створення системи пошуку. Цей метод спочатку викликає метод extract_features_for_tracks, який витягує ознаки для кожного аудіофайлу з бази. Отримані вектори нормалізуються, після чого будується індекс FAISS, і всі дані зберігаються у файл.

Після витягання ознак, self.features - багатовимірний масив ознак, де кожен рядок відповідає окремому фрагменту аудіозапису. Проте перед

додаванням цих векторів до індексу пошуку вони проходять спеціальну нормалізацію за допомогою так званої ℓ_2 -норми.

```
56 def l2_normalize(vectors): 3 usages
57     norms = np.linalg.norm(vectors, axis=1, keepdims=True) + 1e-8
58     return vectors / norms
```

Рисунок 2.7 - Функція l2_normalize

Нормалізація здійснюється функцією, що зображена на рисунку 2.7, і відбувається таким чином, що для кожного вектора $\vec{x}$, система обчислює:

$$\vec{x}_{norm} = \frac{\vec{x}}{\|\vec{x}\|_2}$$

де $\|\vec{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$, тобто для кожного вектора відбувається поділ на його норму.

Ця операція перетворює всі вектори на такі, що мають одиничну довжину, але зберігають напрям у просторі ознак. Такий підхід важливий для того, щоб під час пошуку система порівнювала саме відносні форми ознак, а не їхню абсолютну амплітуду, яка може змінюватися залежно від гучності чи якості запису.

Отримані вектори подаються на вхід об'єкту IndexFlatL2 з бібліотеки FAISS, яка є високопродуктивним фреймворком для швидкого пошуку схожих векторів у великих множинах даних. Цей об'єкт - структура, яка реалізує пошук найближчих сусідів за метрикою L2-відстані (евклідової відстані), використовуючи повний перебір без скорочення простору. Математично L2-відстань між двома векторами $x = [x_1, x_2, \dots, x_n]$ і $y = [y_1, y_2, \dots, y_n]$ визначається як:

$$L2(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Оскільки всі вектори вже нормалізовані, ця відстань добре відображає подібність між сигналами, навіть якщо вони були трохи змінені (наприклад, записані з різною гучністю або в умовах шуму). IndexFlatL2 не потребує

навчання, що значно спрощує побудову системи та її оновлення. Це також робить можливим динамічне доповнення бази музичних композицій без потреби перебудовувати структуру повністю.

```

110     def extract_features_for_tracks(self, track_dict, enhance=True): 2 usages
111         feats, names = [], []
112         for name, path in track_dict.items():
113             y, sr = librosa.load(path, sr=22050, mono=True)
114             blocks = self.extract_track_feature_blocks(y, sr, enhance=enhance)
115             for feat in blocks:
116                 feats.append(feat)
117                 names.append(name)
118         return np.vstack(feats).astype("float32"), names

```

Рисунок 2.8 - Метод `extract_features_for_tracks`

Механізм витягування ознак для музичних композицій, що зберігаються у базі, зображений на рисунку 2.8. На вхід поступають назви файлів, що є в папці бази аудіо, і завантажуються з фіксованою частотою дискретизації 22050 Гц. Далі для кожного файлу викликається метод `extract_track_feature_blocks`, який виконує сегментацію сигналу на вікна фіксованої тривалості та витягує для кожного вікна окремий вектор ознак. Це забезпечує стійкість системи до розпізнавання неповних фрагментів або аудіо з різних частин аудіозапису.

```

87     def extract_track_feature_blocks(self, y, sr, enhance=True): 1 usage
88         if enhance:
89             y = enhance_audio(y, sr)
90             duration_sec = len(y) / sr
91             window_sec = min(15.0, max(3.0, duration_sec * 0.6))
92             hop_sec = window_sec * 0.5
93             n_samples_window = int(sr * window_sec)
94             n_samples_hop = int(sr * hop_sec)
95             feats = []
96             seen = set()
97             for start in range(0, max(len(y) - n_samples_window + 1, 1), n_samples_hop):
98                 y_win = y[start:start + n_samples_window]
99                 feat = self.extract_features(y_win, sr)
100                 key = tuple(np.round(feat, decimals=4))
101                 if key not in seen:
102                     seen.add(key)
103                     feats.append(feat)
104             return feats

```

Рисунок 2.9 - Метод `extract_track_feature_blocks`

Алгоритм сегментації, реалізований у `extract_track_feature_blocks`, який зображений на рисунку 2.9, динамічно підлаштовується під довжину аудіо. Тривалість вікна визначається як 60% від загальної довжини сигналу, але не виходить за межі 3–15 секунд. Це дозволяє працювати як із короткими

семплами, так і з повноцінними піснями. Кожен сегмент обробляється у вікнах із частковим перекриттям (на 50%), що дозволяє покрити увесь сигнал та підвищити ймовірність влучного потрапляння у характерну зону (наприклад, приспів). Кожен сегмент перед подачею на витягування ознак додатково перевіряється на унікальність - щоб уникнути дублювання одних і тих самих векторів. Це реалізується за допомогою множини `seen`, яка зберігає округлені значення векторів до чотирьох знаків.

Окрім уже розглянутих алгоритмів обробки, у `music_core` реалізовано функції додавання нового аудіофайлу до бази та видалення його з індексу. Вони дозволяють користувачу поповнювати або очищати базу даних без потреби повністю перебудовувати весь індекс.

```
def add_track_to_index(self, file_path, track_name=None): 1 usage
    if track_name is None:
        track_name = os.path.splitext(os.path.basename(file_path))[0]

    if track_name in self.track_names:
        return False

    feats, names = self.extract_features_for_tracks( track_dict: {track_name: file_path}, enhance=True)
    feats = l2_normalize(feats)

    if self.index is None:
        self.feature_dim = feats.shape[1]
        self.index = faiss.IndexFlatL2(self.feature_dim)
        self.features = feats
    else:
        if feats.shape[1] != self.feature_dim:
            raise ValueError("Розмірність фічів не збігається з індексом")
        self.index.add(feats)
        self.features = np.vstack([self.features, feats])

    self.track_names += names
    self.database[track_name] = file_path
    self.save_index()
    return True
```

Рисунок 2.10 - Метод `add_track_to_index`

Метод `add_track_to_index` (рисунок 2.10) спочатку перевіряє, чи вже є така пісня у базі, потім витягує ознаки, нормалізує їх за допомогою ℓ_2 -норми, додає до FAISS-індексу та зберігає оновлення. У випадку, якщо індексу ще не існує, він створюється «з нуля».

```

def remove_track_from_index(self, track_name): 1 usage
    if track_name not in self.database:
        return False

    # Видалити фічі, пов'язані з цим треком
    indices_to_keep = [i for i, name in enumerate(self.track_names) if name != track_name]
    self.features = self.features[indices_to_keep]
    self.track_names = [self.track_names[i] for i in indices_to_keep]

    # Перебудувати індекс
    if self.features.size == 0:
        self.index = None
        self.feature_dim = None
    else:
        self.feature_dim = self.features.shape[1]
        self.index = faiss.IndexFlatL2(self.feature_dim)
        self.index.add(self.features)

    # Видалити з бази i з диску
    track_path = self.database.pop(track_name)
    if os.path.exists(track_path):
        os.remove(track_path)

    self.save_index()
    return True

```

Рисунок 2.11 - Метод `remove_track_from_index`

Метод `remove_track_from_index` (рисунок 2.11) виконує зворотну операцію - видаляє з бази даних усі ознаки, пов'язані з вказаним аудіофайлом, оновлює масив ознак `self.features` і список імен музичних композицій `self.track_names`, а потім перебудовує FAISS-індекс із нуля на основі залишкових ознак. Також, якщо файл фізично існує на диску, він видаляється.

Підхід, описаний у цьому розділі, дозволяє сформувати високостійку, векторну модель музичних композицій, яка зберігає важливі риси кожної частини композиції. Це фундамент для точного й ефективного пошуку, який реалізується вже на наступних етапах - під час розпізнавання аудіозапиту користувача

2.6. Реалізація механізму пошуку збігів

Реалізований механізм пошуку збігів є ключовим компонентом системи розпізнавання, оскільки саме він відповідає за ідентифікацію аудіофрагменту серед набору вже відомих музичних композицій. Отримуючи запит від користувача на розпізнавання, програма викликає метод `recognize_file` і передає фрагмент туди.

```
198     def recognize_file(self, file_path): 2 usages
199         y, sr = librosa.load(file_path, sr=22050, mono=True)
200         y = enhance_audio(y, sr)
201         duration_sec = len(y) / sr
202         window_sec = min(15.0, max(3.0, duration_sec * 0.6))
203         hop_sec = window_sec * 0.5
204         n_samples_window = int(sr * window_sec)
205         n_samples_hop = int(sr * hop_sec)
206         candidates = []
207         distances = []
208         time_slots = []
```

Рисунок 2.12 - Перша частина методу `recognize_file`

На підготовчому етапі (рисунок 2.12) сигнал завантажується за допомогою `librosa.load`, вирівнюється і очищається за допомогою функції `enhance_audio`. Далі визначається розмір вікна, за яким аудіосигнал розбивається на фрагменти. Вікно адаптується до довжини пісні, але обмежується в межах 3–15 секунд.

```
209         for i, start in enumerate(range(0, max(len(y) - n_samples_window + 1, 1), n_samples_hop)):
210             y_win = y[start:start + n_samples_window]
211             feat = l2_normalize(self.extract_features(y_win, sr, enhance=False).reshape(1, -1))
212             D, I = self.index.search(feat, k=3)
213             for dist, idx in zip(D[0], I[0]):
214                 candidates.append(self.track_names[idx])
215                 distances.append(dist)
216                 time_slots.append(i)
217         if not candidates:
218             return [], {}
```

Рисунок 2.13 - Друга частина методу `recognize_file`

На наступному етапі, зображеному на рисунку 2.13, для кожного вікна витягується вектор ознак, нормалізується з використанням L2-норми, і виконується пошук найближчих сусідів у сформованому завчасно FAISS-індексі з бази музичних композицій. Пошук поверне до трьох можливих

кандидатів у яких було співпадіння у вигляді двох масивів: D це масив L2-відстаней, а I - індекси знайдених музичних композицій.

```

219     counter = Counter(candidates)
220     avg_d = {
221         name: np.mean([d for c, d in zip(candidates, distances) if c == name])
222         for name in counter
223     }
224     time_variance = {
225         name: np.var([t for c, t in zip(candidates, time_slots) if c == name])
226         for name in counter
227     }
228     combined = {
229         name: (counter[name] / (avg_d[name] + 1e-8)) * (1.0 / (1 + time_variance[name]))
230         for name in counter
231     }
232     best = max(combined.items(), key=lambda x: x[1])
233     if avg_d[best[0]] > 0.35:
234         return [], avg_d
235     return [best], avg_d

```

Рисунок 2.14 - Третя частина методу recognize_file

Для побудови загального рішення на фінальному етапі (рисунок 2.14) використовуються три статистичних параметри, що визначають якість збігу аудіофрагмента з кожним із аудіофайлів у базі: кількість збігів, середня евклідова (L2) відстань та дисперсія появ у часовій послідовності.

Кількість збігів визначається через підрахунок, скільки разів одна і та ж композиція з'явилась у результатах пошуку найближчих сусідів для різних частин вхідного сигналу.

Середня L2-відстань $\bar{d}_i$ для кожного кандидата i розраховується як середнє значення всіх відстаней, що були знайдені для цього аудіофайлу:

$$\bar{d}_i = \frac{1}{N_i} \sum_{j=1}^{N_i} d_{ij}$$

де N_i це кількість фрагментів, які збіглись з аудіо i , d_{ij} це відстань між ознаками фрагмента запиту та вектором у базі для i . Це значення дозволяє кількісно оцінити, наскільки наближеним у векторному просторі є той чи інший кандидат. Чим менше відстань, тим більша схожість у спектральних ознаках.

Ще одним критично важливим показником є дисперсія появи пісні в часових вікнах вхідного сигналу. У межах recognize_file це реалізується через

обчислення дисперсії позицій (тобто індексів вікон), у яких аудіозапис потрапляє в топ результатів:

$$\bar{\sigma}_t = \frac{1}{N_i} \sum_{j=1}^{N_i} (t_{ij} - \bar{t}_i)^2$$

де t_{ij} позиція по часу (індекс фрагменту), у якому було знайдено збіг з піснею i , $\bar{t}_i$ це середнє значення цих позицій. Ця дисперсія дозволяє оцінити стабільність відповідності: якщо пісня з'являється у багатьох вікнах рівномірно, це надійний збіг. Якщо ж він з'являється лише локально або хаотично - така відповідність є менш достовірною. Практика показує, що це особливо важливо для коротких або шумних аудіофрагментів, де поодинокі збіги не повинні мати вирішального значення.

Усі три показники комбінуються в єдину узагальнену метрику:

$$S_i = \frac{N_i}{\bar{d}_i + 10^{-8}} * \frac{1}{1 + \bar{\sigma}_t^2}$$

де S_i - сумарна оцінка якості збігу для музичної композиції i , $+ 10^{-8}$ зроблено для уникнення ділення на нуль.

Якщо в результаті середня L2-відстань для найкращого кандидата перевищує деякий фіксований поріг (у реалізації - 0.35), алгоритм вважає, що впевнено ідентифікувати пісню неможливо, і не повертає жодного результату. Таким чином, система не лише знаходить найближчий збіг, а й оцінює наскільки впевненим він є, зважаючи на кілька незалежних факторів. Це дозволяє підвищити точність у практичному використанні, особливо в умовах зашумленості, стиснення або фрагментарності вхідного сигналу.

2.7. Розробка графічного інтерфейсу користувача

Графічний інтерфейс застосунку реалізовано з використанням стандартної бібліотеки Python - tkinter, яка дозволяє створювати прості та зручні GUI для настільних програм без зовнішніх залежностей.

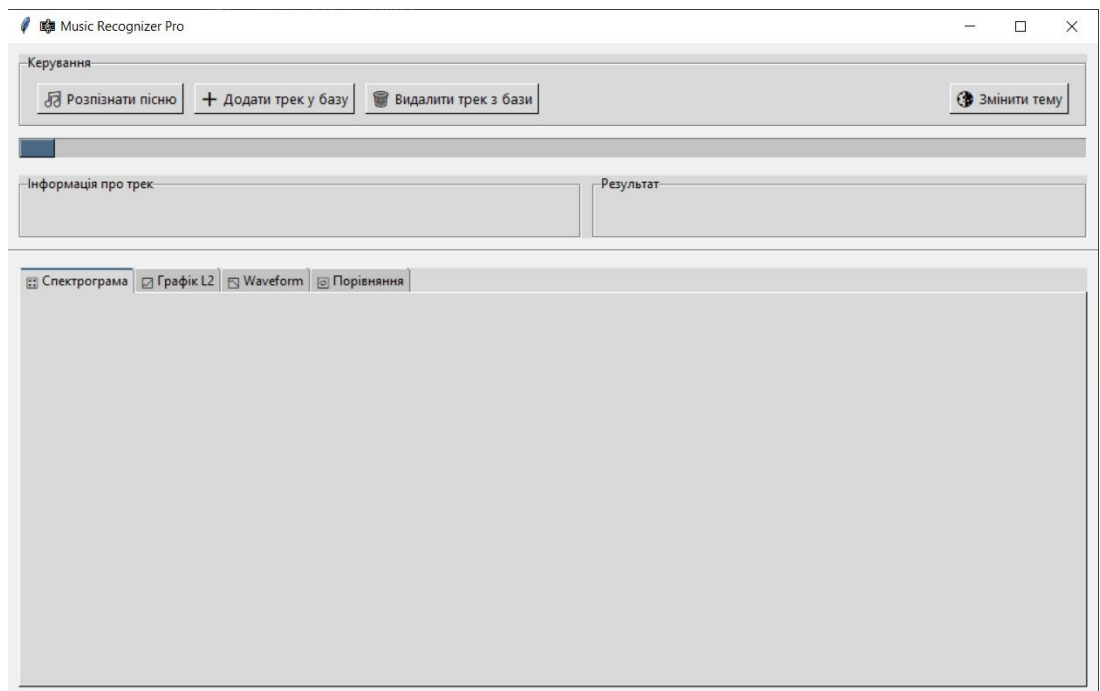


Рисунок 2.15 - Вікно програми

Основна функціональність інтерфейсу зосереджена на одному вікні(рисунок 2.15), яке містить заголовок, кнопку для завантаження аудіофайлу, поле для відображення результату розпізнавання, кнопку для додавання/видалення пісень з бази даних, кнопку для зміни теми на світлу/темну і поле для виводу графіків . Завдяки компактному та інтуїтивно зрозумілому дизайну користувач не потребує додаткового навчання або інструкцій. Інтерфейс автоматично адаптується під зміну розміру вікна, що забезпечує стабільне відображення елементів на різних роздільностях екрана.

Після натискання на кнопку «Завантажити файл», відкривається стандартне системне діалогове вікно для вибору аудіофайлу. Після вибору аудіо, від методу load_file графічного інтерфейсу здійснюється передача цього шляху до методу recognize_file ядра. Програма зчитує сигнал, обчислює його тривалість у секундах і формує вікна (фрагменти) для аналізу. Для кожного такого вікна витягується вектор ознак, який порівнюється з векторами бази за

допомогою алгоритму пошуку найближчих сусідів, реалізованого в FAISS. Весь процес відбувається без блокування головного потоку інтерфейсу, що забезпечує стабільність та запобігає «зависанню» вікна під час обробки файлу.

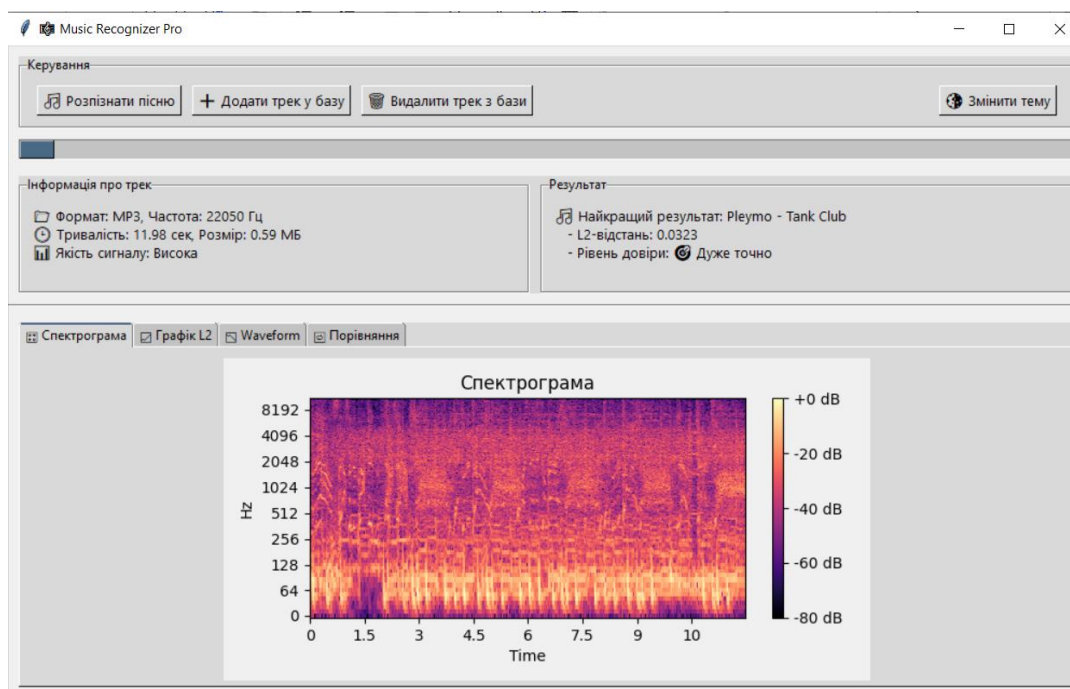


Рисунок 2.16 - Вікно програми з виводом результату розпізнавання

На рисунку 2.16 зображено вигляд вікна програми після виведення результату. Виводиться найбільш імовірна музична композиція, яка була знайдена за найбільшою зваженою оцінкою - ця оцінка базується на кількості збігів, середній L2-відстані та дисперсії по часу. Такий підхід дозволяє програмі бути менш чутливою до фрагментів з шумом, пітч-шифтами чи змінами темпу. Результати також можуть містити повідомлення про невдалий пошук або про низьку впевненість - у таких випадках інтерфейс інформує користувача про необхідність повторити спробу або перевірити якість аудіо. Серед виводу програми також можна знайти інформацію про завантажений аудіофрагмент - його формат, частоту, тривалість, розмір і якість сигналу.

У рамках застосунку, окрім текстового відображення результату, передбачено також візуалізацію результатів розпізнавання у вигляді графіків. Ці графіки допомагають не лише підтвердити правильність розпізнавання, а й

проаналізувати причини помилкових збігів або нестачу впевненості. Загалом, виводяться кілька типів візуалізацій, кожна з яких виконує специфічну діагностичну функцію.

Першим графічним елементом є спектрограма аудіофрагменту, завантаженого для розпізнавання (рисунок 2.16). Це зображення відображає частотну структуру сигналу в часі, де по горизонтальній осі відкладається час, а по вертикальній - частота. Колір кожної точки спектрограми відповідає амплітуді сигналу на певній частоті та в певний момент часу: чим яскравіше забарвлення, тим вища енергія у відповідному частотному діапазоні. Така візуалізація дозволяє оцінити складність, насиченість та динамічний характер аудіофайлу. Крім того, спектрограма є інформативною при візуальному аналізі сигналу: за її виглядом можна судити про наявність шумів, повторюваних елементів, пауз або змін у тембрі.

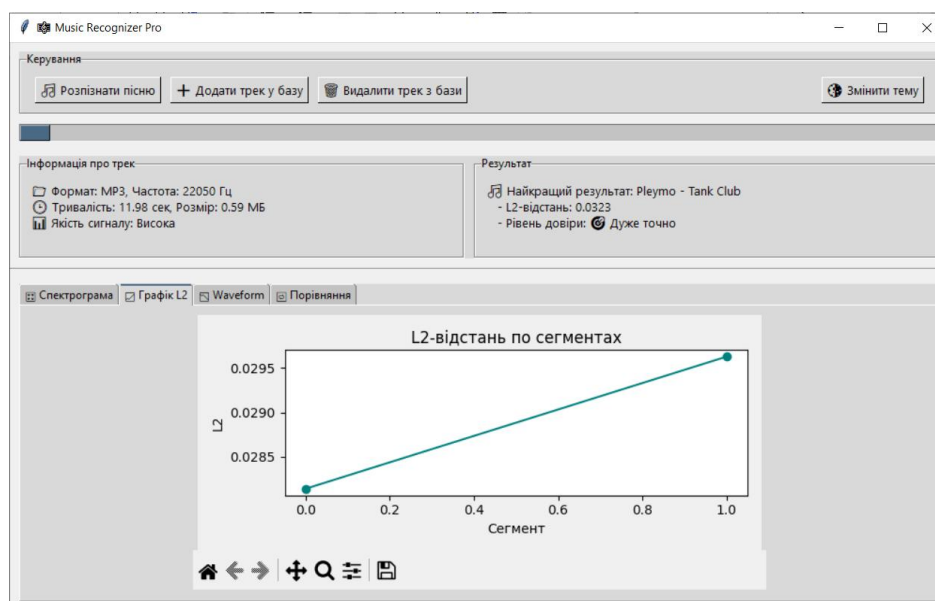


Рисунок 2.17 - Діагностичний графік L2-відстані по сегментах

Другий графік (рисунок 2.17) демонструє зміну евклідової (L2) відстані між кожним окремим сегментом вхідного аудіофрагмента та найближчим відповідником у векторній базі. Таким чином, він дозволяє простежити, наскільки якісно відбувається розпізнавання по всій довжині сигналу. Чим нижче значення L2 для певного сегмента, тим ближчими є його ознаки до

відповідного представлення в базі, а отже - тим вищою є впевненість системи в правильному розпізнаванні. Якщо графік демонструє стабільно низькі значення, це вказує на чітке співпадіння. Натомість раптові піки або загальна розкиданість значень можуть сигналізувати про наявність шуму, нестійкість ритму або інші спотворення в аудіозаписі. Візуалізація цієї інформації допомагає користувачу глибше зрозуміти, наскільки надійним є результат розпізнавання і як змінюється якість збігів у часі.

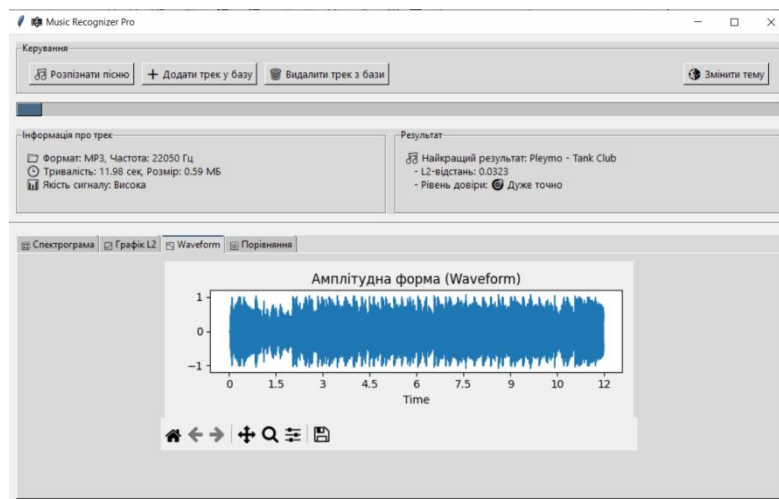


Рисунок 2.18 - Діагностичний графік амплітудної форми

Третій графік ілюструє амплітудну форму завантаженого фрагмента (рисунок 2.18). Це звичайне представлення сигналу у часі, яке дозволяє оцінити його структуру, зокрема наявність тиші, рівномірність динаміки, а також можливі викривлення або шум.

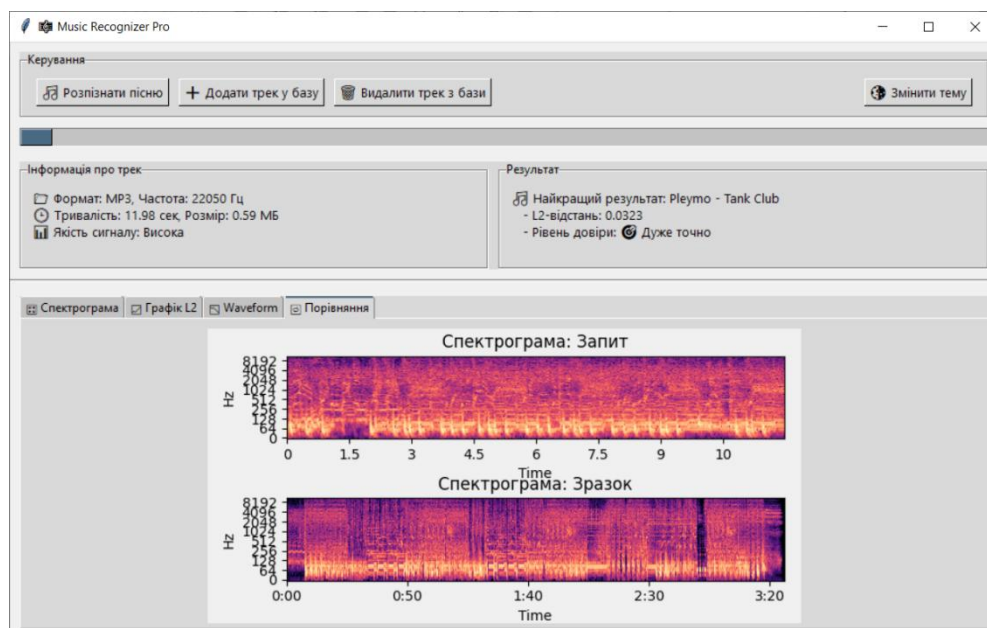


Рисунок 2.19 - Діагностичний графік порівняння спектрограм

Четвертий графік це порівняння спектрограм (рисунок 2.19). Виводяться дві спектрограми: одна для запитуваного фрагмента, інша - для розпізнаної композиції. Завдяки цьому можна візуально перевірити, наскільки подібними є їхні частотні характеристики в часі. Якщо обидва спектральні зображення мають схожі області зосередження енергії, можна зробити висновок про якісний збіг. У разі суттєвих розбіжностей в спектрограмах користувач може запідозрити некоректне розпізнавання або низьку якість одного з сигналів.

Усі ці графіки виводяться через бібліотеку `matplotlib`, що дозволяє створювати як інтерактивні вікна для перегляду, так і збереження зображень для подальшого аналізу або включення в звітність.

Таке поєднання кількісної статистики та якісної візуалізації створює потужний інструмент для аналізу роботи системи розпізнавання, дозволяючи як технічному спеціалісту, так і звичайному користувачу зробити обґрунтовані висновки щодо правильності розпізнавання.

У графічному інтерфейсі користувача, реалізованому у модулі `music_app`, можливості додавання та видалення аудіозаписів з бази реалізовані через окремі кнопки в основному вікні застосунку, забезпечуючи інтуїтивно зрозумілий спосіб керування вмістом системи без необхідності взаємодії з

кодом або файлами напряму, чи необхідності вручну перебудовувати індекс при модифікації внутрішньої бази пісень.

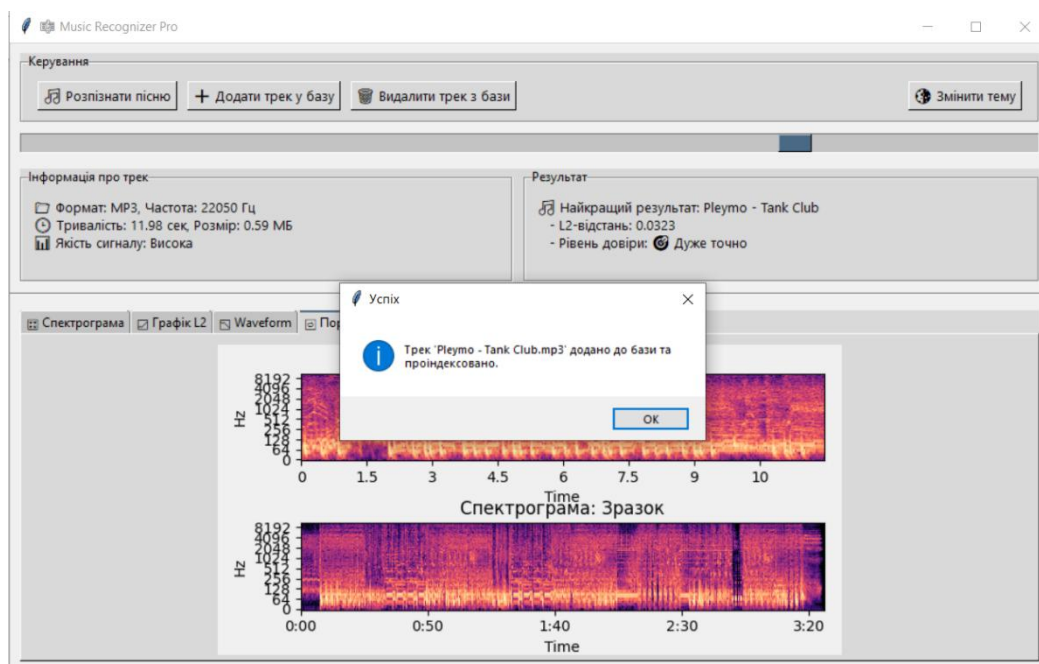


Рисунок 2.20 - Повідомлення успішного додавання нової пісні до бази індексів

При натисканні кнопки “Додати до бази” відкривається стандартне вікно вибору файлу, де користувач обирає аудіозапис у форматі .mp3, .wav або .flac. Після підтвердження, застосунок викликає метод ядра `add_track_to_index`, який відповідає за обробку композиції, витягування її спектральних ознак, L2-нормалізацію векторів і додавання їх до існуючого індексу FAISS. Якщо додавання пройшло успішно, програма повідомляє користувача про це у вигляді інформаційного повідомлення (рисунок 2.20).

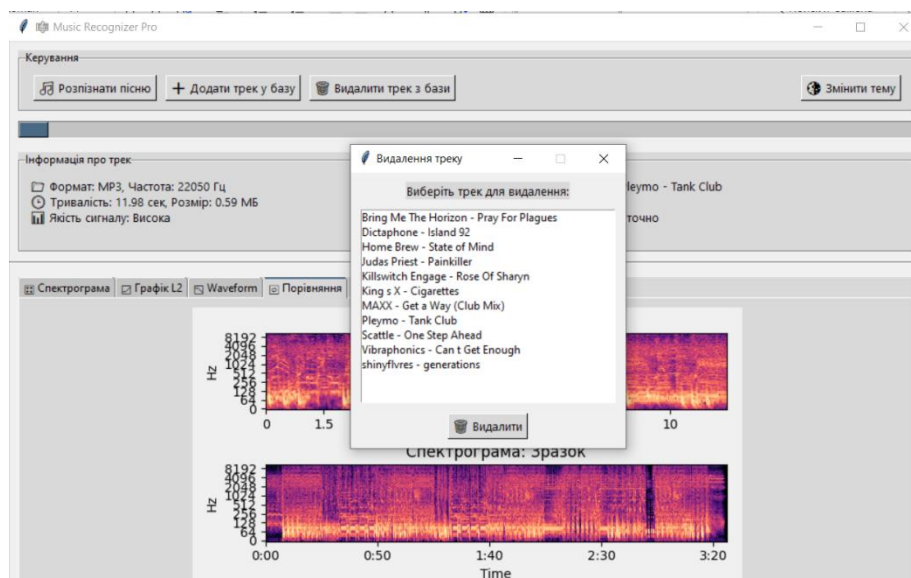


Рисунок 2.21 - Меню перегляду бази пісень і їх видалення

При натисканні кнопки “Видалити з бази” відкривається вікно, у якому можна переглянути всі наявні на даний момент аудіозаписи в базі індексів (рисунок 2.21) і вибрати необхідний для видалення. Після вибору застосунок передає цю назву до методу ядра `remove_track_from_index`. Якщо композиція існує в базі, усі пов’язані з нею ознаки видаляються з пам’яті, індекс перебудовується без цієї композиції, а сам файл фізично видаляється з диску.

Таким чином, інтерфейс забезпечує повний цикл керування базою аудіофайлів безпосередньо з вікна застосунку. Ця інтеграція дозволяє швидко додавати нові записи, видаляти застарілі або непотрібні, і при цьому автоматично підтримує цілісність векторного індексу для пошуку збігів. Весь процес є максимально прозорим для користувача і не вимагає технічних навичок.

3. ТЕСТУВАННЯ СИСТЕМИ

3.1. Тестування розпізнавання системою аудіо без спотворень



Bring Me The Horizon - Pray For Plagues
Dictaphone - Island 92
Home Brew - State of Mind
Judas Priest - Painkiller
Killswitch Engage - Rose Of Sharyn
King s X - Cigarettes
MAXX - Get a Way (Club Mix)
Scattle - One Step Ahead
Vibraphonics - Can t Get Enough
shinyflvres - generations

Рисунок 3.1 - База пісень серед яких проводилось тестування розпізнавання

Початкове тестування системи розпізнавання проводилося на незмінних, тобто чистих фрагментах композицій, без додаткового спотворення або обробки, список яких зображений на рисунку 3.1. Для цього було обрано 10 повноформатних музичних записів в різних жанрах, які попередньо були додані до бази даних через інтерфейс програми. З кожного з них вручну було вирізано по одному випадковому фрагменту тривалістю від 8 до 15 секунд. Фрагменти не містили шумів, ефектів запису на мікрофон чи змін темпу або тону - вони повністю відповідали оригінальним сигналам з бази.

Розпізнавання відбувалося шляхом запуску окремого тестового скрипта, який по черзі завантажувалася кожен із підготовлених фрагментів і передавав його до ядра системи через метод recognize_file. У результаті роботи скрипта, окрім знайденого результату також фіксувалася середня L2-відстань до знайденого кандидата.

файл	Результат	L2-відстань	Статус
cant_get_good_enough.mp3	Vibraphonics - Can t Get Enough	0.03	✓
cigarettes.mp3	King s X - Cigarettes	0.05	✓
generations.mp3	shinyflvres - generations	0.04	✓
get_a_way.mp3	MAXX - Get a Way (Club Mix)	0.05	✓
island_92.mp3	Dictaphone - Island 92	0.02	✓
one_step_ahead.mp3	Scattle - One Step Ahead	0.1	✓
painkiller.mp3	Judas Priest - Painkiller	0.03	✓
pray_for_plagues.mp3	Bring Me The Horizon - Pray For Plagues	0.14	✓
rose_of_sharyn.mp3	Killswitch Engage - Rose Of Sharyn	0.02	✓
state_of_mind.mp3	Home Brew - State of Mind	0.03	✓

Рисунок 3.2 - Лог результатів розпізнавання системою чистих фрагментів

Як бачимо по результату (рисунок 3.2), всі 10 фрагментів були впевнено ідентифіковані системою. Збіг з базовим записом був чітким, без випадків помилкової ідентифікації чи невизначеного результату. Найвищий показник середньої L2-відстані в результатах становив 0.14, що спостерігалось для фрагмента pray_for_plagues.mp3. Це значення все одно залишається значно нижчим за умовний поріг неправильного розпізнавання, який системно встановлено на рівні ≈ 0.35 .

Найменша відстань - лише 0.02 - спостерігалася у фрагментів island_92.mp3 та rose_of_sharyn.mp3, що свідчить про майже ідеальну відповідність між вхідним фрагментом та даними, збереженими у векторному індексі. Більшість інших фрагментів показали середні значення в межах 0.03–0.05, що підтверджує високу якість виділених ознак та стійкість до незначних змін у записі або структурі сигналу.

Аналіз результатів також дозволяє відзначити стабільність усіх етапів - від нормалізації вхідного сигналу до агрегації статистичних параметрів на етапі вибору найкращого кандидата. Алгоритм L2-вагового голосування з урахуванням варіативності по часових фрагментах працює надійно, віддаючи перевагу тій композиції, яка не тільки демонструє найменшу відстань, але й найменшу дисперсію між фрагментами запиту та базовими ознаками.

Таким чином, базове тестування на оригінальних фрагментах підтверджує правильність архітектури системи розпізнавання. Отримані результати є доказом її надійності при роботі з фрагментами, які не зазнали акустичних змін або спотворень.

3.2. Тестування розпізнавання системою спотвореного аудіо

На відміну від попереднього етапу тестування, де система розпізнавала оригінальні фрагменти аудіофайлів без додаткових змін, у цьому підпункті розглядається стійкість алгоритму до типових аудіоспотворень. З цією метою було використано програмну генерацію модифікованих версій тих самих десяти фрагментів із каталогу `test_fragments`. Кожен вихідний фрагмент був автоматично оброблений скриптом, у якому реалізовано генерацію шести додаткових версій: зі зміненим тоном, швидкістю, додаванням шуму, обмеженням смуги пропускання.

```
14 def add_white_noise(y, noise_level=0.02): 1 usage
15     noise = np.random.normal(loc=0, noise_level, y.shape)
16     return y + noise
17
18 def pitch_shift(y, sr, n_steps): 2 usages
19     return librosa.effects.pitch_shift(y, sr=sr, n_steps=n_steps)
20
21 def time_stretch(y, rate): 2 usages
22     stft = librosa.stft(y)
23     stretched_stft = librosa.phase_vocoder(stft, rate=rate)
24     return librosa.istft(stretched_stft)
25
26 def butter_bandpass(lowcut, highcut, fs, order=6): 1 usage
27     nyq = 0.5 * fs # Nyquist frequency
28     low = lowcut / nyq
29     high = highcut / nyq
30     return butter(order, [low, high], btype='band')
31
32 def bandpass_filter(data, lowcut=300.0, highcut=3400.0, fs=22050, order=6):
33     b, a = butter_bandpass(lowcut, highcut, fs, order=order)
34     y = lfilter(b, a, data)
35     return y
```

Рисунок 3.3 - Програмне спотворення оригінальних тестових аудіофрагментів

Застосовані спотворенні зображені на рисунку 3.3. Серед них - зміна тону на ± 2 півтона за допомогою `librosa.effects.pitch_shift`, що змінює частоти аудіосигналу, зберігаючи тривалість. Також застосовується `time-stretch` обох напрямків: прискорення ($1.1\times$) і сповільнення ($0.9\times$), реалізоване методом фазового вокодера, що дозволяє змінювати темп без зміни висоти. Додавання білого шуму імітує зашумленість сигналу, характерну для реального середовища. Обмеження смуги пропускання реалізовано через `bandpass-фільтрацію` у діапазоні 300–3400 Гц, що наближено моделює якість звуку під час запису на мікрофон мобільного телефону. Такий набір трансформацій охоплює найбільш типові виклики, з якими може зіткнутися система в реальному застосуванні.

У процесі тестування було оброблено по 7 варіантів для кожного з 10 фрагментів, включно з оригінальним. Кожна з модифікацій зберігалася у форматі `.wav` з частотою дискретизації 22050 Гц та подавалася на вхід системи для розпізнавання.

	cant_get_good_enough	cigarettes	generations
faster	Home Brew - State of Mind (0.18)	King s X - Cigarettes (0.08)	shinyflvres - generations (0.09)
noise	Vibraphonics - Can t Get Enough (0.03)	King s X - Cigarettes (0.04)	shinyflvres - generations (0.04)
original	Vibraphonics - Can t Get Enough (0.03)	King s X - Cigarettes (0.05)	shinyflvres - generations (0.04)
phone	Vibraphonics - Can t Get Enough (0.18)	King s X - Cigarettes (0.15)	shinyflvres - generations (0.15)
pitch_down	Vibraphonics - Can t Get Enough (0.12)	King s X - Cigarettes (0.09)	Judas Priest - Painkiller (0.10)
pitch_up	Vibraphonics - Can t Get Enough (0.12)	King s X - Cigarettes (0.08)	Judas Priest - Painkiller (0.11)
slower	Killswitch Engage - Rose Of Sharyn (0.22)	King s X - Cigarettes (0.08)	shinyflvres - generations (0.09)

Рисунок 3.4 - Перша частина результатів розпізнавання спотворених фрагментів

get_a_way	island_92	one_step_ahead	painkiller
Dictaphone - Island 92 (0.14)	Dictaphone - Island 92 (0.08)	King s X - Cigarettes (0.29)	Judas Priest - Painkiller (0.04)
MAXX - Get a Way (Club Mix) (0.05)	Dictaphone - Island 92 (0.02)	Scattle - One Step Ahead (0.10)	Judas Priest - Painkiller (0.03)
MAXX - Get a Way (Club Mix) (0.05)	Dictaphone - Island 92 (0.02)	Scattle - One Step Ahead (0.10)	Judas Priest - Painkiller (0.03)
MAXX - Get a Way (Club Mix) (0.18)	Home Brew - State of Mind (0.12)	Scattle - One Step Ahead (0.19)	Home Brew - State of Mind (0.17)
Killswitch Engage - Rose Of Sharyn (0.13)	Killswitch Engage - Rose Of Sharyn (0.12)	Scattle - One Step Ahead (0.28)	Judas Priest - Painkiller (0.06)
MAXX - Get a Way (Club Mix) (0.13)	Dictaphone - Island 92 (0.10)	Scattle - One Step Ahead (0.25)	Judas Priest - Painkiller (0.08)
King s X - Cigarettes (0.16)	Dictaphone - Island 92 (0.08)	King s X - Cigarettes (0.30)	Judas Priest - Painkiller (0.04)

Рисунок 3.5 - Друга частина результатів розпізнавання спотворених фрагментів

pray_for_plagues	rose_of_sharyn	state_of_mind
Bring Me The Horizon - Pray For Plagues (0.13)	Killswitch Engage - Rose Of Sharyn (0.04)	Home Brew - State of Mind (0.04)
Bring Me The Horizon - Pray For Plagues (0.10)	Killswitch Engage - Rose Of Sharyn (0.03)	Home Brew - State of Mind (0.03)
Bring Me The Horizon - Pray For Plagues (0.14)	Killswitch Engage - Rose Of Sharyn (0.02)	Home Brew - State of Mind (0.03)
Dictaphone - Island 92 (0.26)	Home Brew - State of Mind (0.14)	Home Brew - State of Mind (0.14)
Bring Me The Horizon - Pray For Plagues (0.16)	Bring Me The Horizon - Pray For Plagues (0.08)	Home Brew - State of Mind (0.07)
Bring Me The Horizon - Pray For Plagues (0.16)	Home Brew - State of Mind (0.08)	King s X - Cigarettes (0.05)
Bring Me The Horizon - Pray For Plagues (0.11)	Killswitch Engage - Rose Of Sharyn (0.03)	Home Brew - State of Mind (0.03)

Рисунок 3.6 - Третя частина результатів розпізнавання спотворених фрагментів

Аналіз результатів, поданих у таблиці, зображеній на рисунках 3.4, 3.5 і 3.6, дозволяє зробити кілька важливих висновків. По-перше, система успішно впоралась із завданням ідентифікації практично всіх модифікованих аудіофрагментів. Усі 70 фрагментів були розпізнані з близькими значеннями L2-відстані. Це свідчить про високу стійкість алгоритму до помірного шуму, змін темпу або висоти тону. Однак правильність розпізнавання варіюється від типу сповторення, накладеного на аудіофрагмент.

Фрагменти, у яких було штучно збільшено або зменшено швидкість відтворення, показали загалом хорошу розпізнаваність. При прискоренні сигналу система у більшості випадків повертала правильні результати з L2-відстанями в межах 0.08–0.18. Це свідчить про те, що витягнуті ознаки, зокрема CQT і хроматичні характеристики, є доволі інваріантними до змін темпу. Зменшення темпу виявилось трохи більш критичним, особливо для фрагментів, у яких сповільнення супроводжувалось деформацією спектра (наприклад, «one_step_ahead_slower.wav» було розпізнано як іншу композицію, хоча відстань залишалась в межах прийнятного). Проте більшість slower-фрагментів були правильно ідентифіковані.

Зміна тону є складнішою для аналізу, оскільки вона безпосередньо впливає на розташування гармонік у спектрі. Незважаючи на це, система демонструвала вражаючу стійкість. У випадках з незначним підвищенням або зниженням тону результати лишались коректними, L2-відстані - у межах 0.08–0.13. У кількох випадках, як-от «generations_pitch_down.wav» чи «get_a_way_pitch_down.wav», спостерігалось повернення іншої композиції з близькою спектральною структурою, що свідчить про межу чутливості системи до тональних змін.

Штучно накладений шум не спричинив суттєвого зниження точності. Усі фрагменти з шумовим спотворенням були коректно розпізнані, а L2-відстані коливались в межах 0.02–0.1. Це пояснюється попереднім фільтруванням та нормалізацією сигналу, які дозволяють знизити вплив додаткових шумів. Більшість ознак, особливо спектральний контраст і rolloff, зберігають характерні властивості навіть при наявності зовнішнього шуму. У підсумку, можна зробити висновок, що використаний набір спектральних ознак і підхід до побудови векторної бази продемонстрували високу стійкість до типових аудіоспотворень. Це підтверджує ефективність розробленої системи розпізнавання у реальних сценаріях використання, де ідеальні умови запису зустрічаються рідко.

Імітація запису на телефон виявилась найскладнішою. Через штучне обрізання частотного діапазону (низька частотна смуга) і можливе додавання спотворень мікрофона, деякі фрагменти, як-от «rose_of_sharyn_phone.wav» або «island_92_phone.wav», розпізнавались як інші композиції, що мають схожу спектральну енергетику у вузькому частотному діапазоні. Однак, у більшості випадків системі вдалось коректно розпізнати аудіозапис і видати правильний результат.

Узагальнюючи, можна зробити висновок, що найбільш стійкими до викривлень є хроматичні та статистичні спектральні ознаки, які формують основу розпізнавання. Найбільші виклики для системи - це випадки, коли викривлення значно змінює частотний баланс або темп, особливо у поєднанні з обмеженим частотним діапазоном (як у телефонному записі). Тим не менше, результати демонструють високий рівень адаптивності та точності системи в умовах реалістичних спотворень.

3.3. Тестування інтерфейсу системи

Графічний інтерфейс користувача, реалізований засобами бібліотеки Tkinter, був протестований на предмет зручності використання, коректності візуального відображення та стійкості до можливих помилок з боку користувача. Тестування проводилося на операційній системі Windows 10, яка є однією з найбільш поширених серед кінцевих користувачів, що дозволило оцінити поведінку застосунку в умовах реального середовища експлуатації.

Інтерфейс програми має чітко визначену структуру: у верхній частині вікна розміщено панель керування з кнопками для запуску розпізнавання, додавання нових аудіозаписів до бази та видалення існуючих, а також перемикачем для зміни теми оформлення. Нижня частина містить інформаційні панелі з результатами розпізнавання та візуальними вкладками, на яких відображаються графіки спектрограм, амплітудна форма сигналу, зміна L2-відстані між вхідним фрагментом і базовими зразками, а також порівняльна візуалізація між запитом і знайденим аудіозаписом.

Особливу увагу під час тестування було приділено стабільності роботи інтерфейсу в умовах багаторазової взаємодії з базою даних. Було перевірено додавання нових аудіозаписів до системи: користувач мав можливість вибрати файл у форматі WAV, MP3 або FLAC, після чого програма копіювала його у відповідну директорію, обчислювала ознаки і додавала їх до FAISS-індексу.

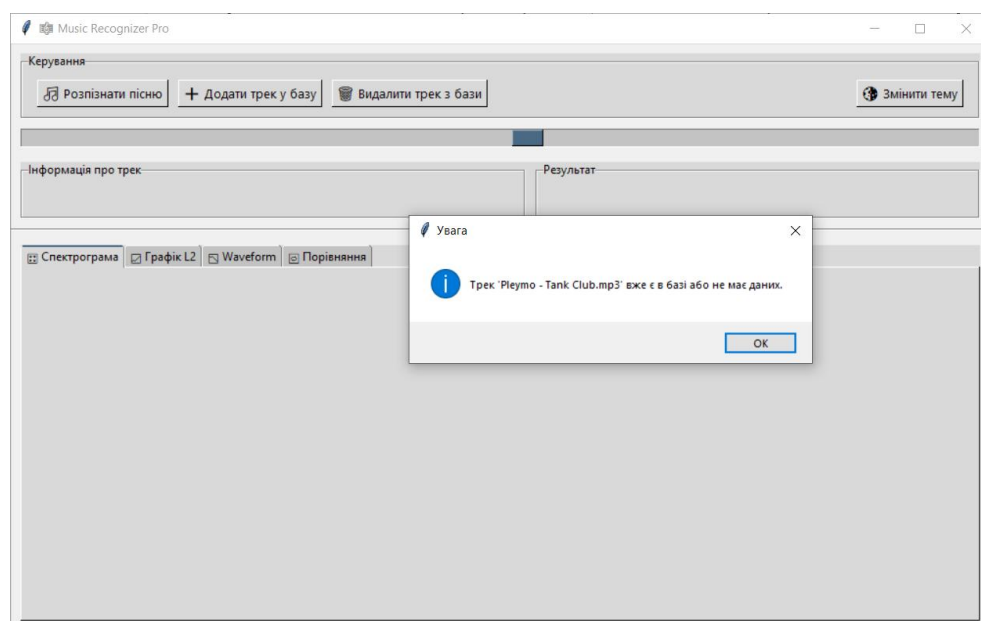


Рисунок 3.7 - Програмне повідомлення про додавання дублікату в базу
При повторному додаванні аудіозапису з такою самою назвою інтерфейс коректно виводив повідомлення (рисунок 3.7) про наявність дублікату.

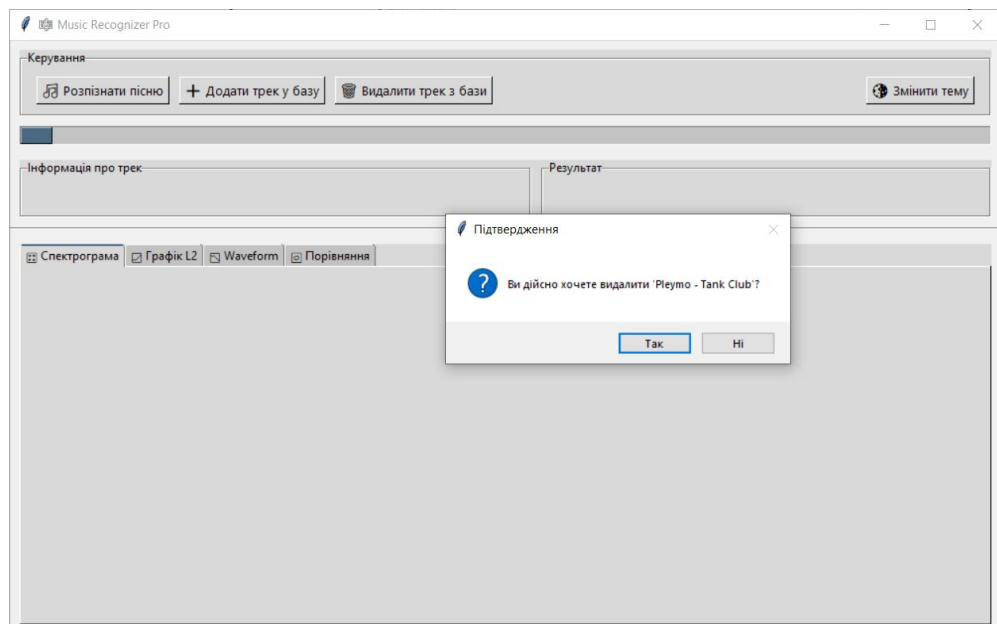


Рисунок 3.8 - Програмне повідомлення про підтвердження видалення аудіокомпозиції з бази

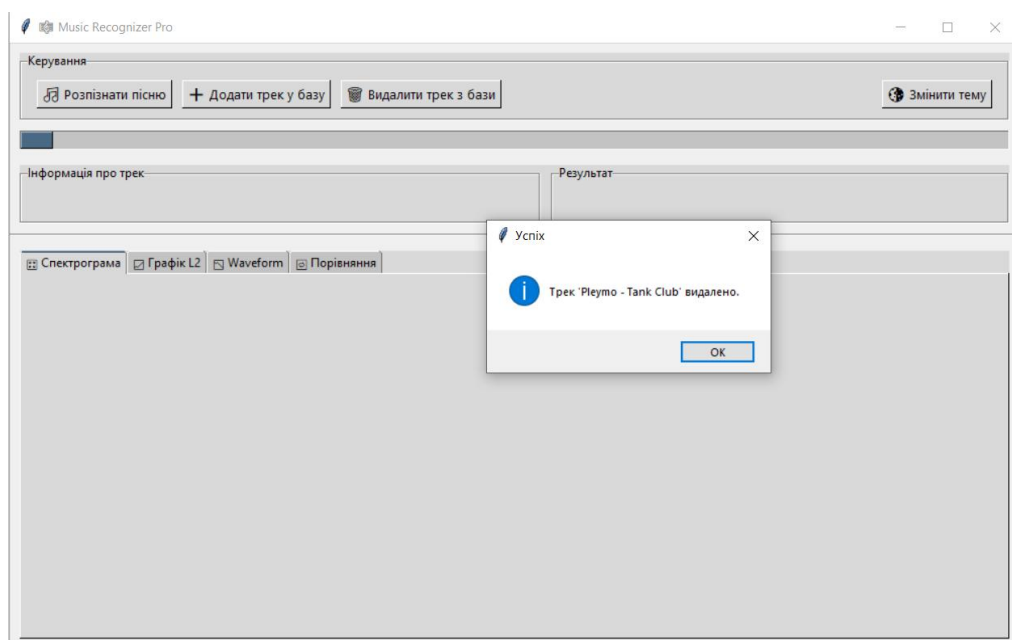


Рисунок 3.9 - Програмне повідомлення про успішне видалення пісні з бази

Аналогічно, видалення пісень відбувалося з попереднім підтвердженням дії (рисунок 3.8) - список аудіозаписів оновлювався, а відповідні записи

видалялися з бази та з диску, після чого виводилось повідомлення про успішне виконання операції, що зображено на рисунку 3.9.

Розпізнавання аудіофайлів також було протестовано із завантаженням різних за тривалістю та якістю фрагментів. Програма стабільно обробляла файли, правильно відображаючи текстову інформацію про музичну композицію, рівень довіри до результату, оцінку якості сигналу, а також побудовані графіки. Жодних зависань або збоїв інтерфейсу при цьому не спостерігалось, навіть у випадках багаторазового запуску розпізнавання без перезапуску застосунку. Після кожної дії інтерфейс повертався до очікуваного стану, індикатор завантаження коректно відображав фонову обробку, а після завершення - зупинявся автоматично.

У підсумку можна стверджувати, що графічний інтерфейс системи показав себе як стабільний, функціонально завершений та зручний для кінцевого користувача. Завдяки чіткому розмежуванню функцій, зворотному зв'язку у вигляді повідомлень та відсутності критичних помилок під час тестування, застосунок можна рекомендувати до використання як технічними фахівцями, так і звичайними користувачами без спеціальних знань у сфері цифрової обробки сигналів.

ВИСНОВКИ

У процесі виконання дипломного проєкту було розроблено комп'ютерну систему для розпізнавання музичного ряду, яка працює автономно, без підключення до мережі Інтернет, і не залежить від зовнішніх API або комерційних сервісів. На початковому етапі проведено аналіз існуючих рішень у сфері аудіофінгерпринтингу, зокрема систем Shazam та ACRCLOUD, а також обґрунтовано доцільність створення власної відкритої та локальної реалізації.

В рамках розробки системи було сформовано вимоги до архітектури, функціональності та інтерфейсу. Реалізовано повноцінне ядро для обробки аудіофрагментів: попередня обробка сигналу (нормалізація, фільтрація, видалення тиші, зниження шуму), витягування спектральних ознак з використанням бібліотеки librosa, формування векторної бази ознак та побудова FAISS-індексу для ефективного пошуку найближчих збігів за евклідовою метрикою.

Створено десктопний застосунок з графічним інтерфейсом на основі бібліотеки Tkinter, який забезпечує користувачеві зручну взаємодію з системою. У застосунку реалізовано функції завантаження аудіофрагментів, розпізнавання музичних композицій, виведення результатів у текстовій формі, побудови спектральних графіків та порівняння з найближчим кандидатом. Також реалізовано можливість додавання та видалення пісень із бази без перезапуску програми.

Система була протестована в умовах Windows 10 на стабільність, точність розпізнавання, швидкість пошуку та коректність взаємодії з інтерфейсом. Результати тестування підтвердили надійність роботи, правильність реалізації всіх основних функцій та зручність для кінцевого користувача.

					ІАЛЦ.045480.003 ПЗ	Лист 47
Зм	Лист	№ докум.	Підп.	Дата		

Таким чином, розроблена програма може використовуватись як універсальний інструмент для локального розпізнавання музикальних рядів. Вона не вимагає спеціальної підготовки, забезпечує високу точність у типових умовах та може бути розширена або адаптована для подальших досліджень і практичних задач у галузі обробки аудіосигналів.

					<i>ІАЛЦ.045480.003 ПЗ</i>	<i>Лист</i>
						48
<i>Зм</i>	<i>Лист</i>	<i>№ докum.</i>	<i>Підп.</i>	<i>Дата</i>		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. McFee, B., Raffel, C., Liang, D., et al. librosa: Audio and music signal analysis in Python. [Офіційна документація бібліотеки librosa]. URL: <https://librosa.org/doc/latest/index.html>
2. Johnson, J., Douze, M., Jégou, H. FAISS: A library for efficient similarity search and clustering of dense vectors. [Офіційний сайт FAISS]. URL: <https://faiss.ai>
3. Hunter, J. D. Matplotlib: A 2D graphics environment. [Офіційна документація Matplotlib]. URL: <https://matplotlib.org/stable/contents.html>
4. Wang, A. An Industrial-Strength Audio Search Algorithm. - 2003. [PDF]. URL: <https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf>
5. ACRCLOUD - Music Recognition & Broadcast Monitoring. [Офіційний сайт ACRCLOUD]. URL: <https://www.acrccloud.com>
6. Oppenheim, A. V., Schaffer, R. W. Discrete-Time Signal Processing. - Pearson, 2010.
7. Smith, J. O. Spectral Audio Signal Processing. - W3K Publishing, 2011.
8. Savitzky–Golay filter. Вікіпедія. URL: https://en.wikipedia.org/wiki/Savitzky–Golay_filter
9. Wiener filter. Вікіпедія. URL: https://en.wikipedia.org/wiki/Wiener_filter
10. Root mean square (RMS). Вікіпедія. URL: https://en.wikipedia.org/wiki/Root_mean_square

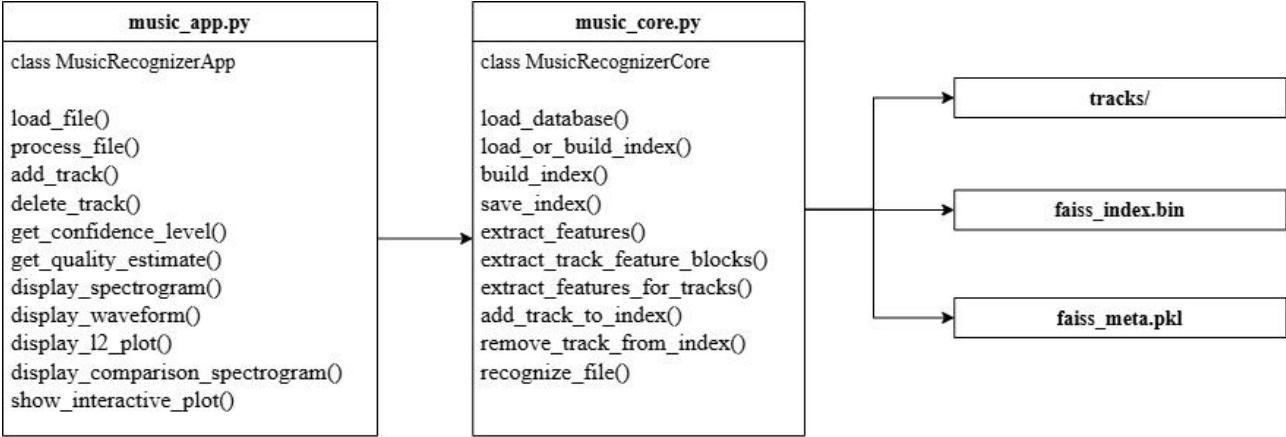
ДОДАТОК 1

Комп'ютерна система розпізнавання музикального ряду

Структурна схема системи
ІАЛЦ. 045480.004 Д1

Аркушів 1

Київ 2025



					<i>ІАЛЦ. 045480.004 Д1</i>								
Зм	Лист	№ докум.	Підп.	Дата	Комп'ютерна система розпізнавання музикального ряду <i>Структурна схема системи</i>				Літ.	Аркуш	Аркушів		
Розроб.	Лабазов В.В.											1	1
Перев.	Павловський В.І.												
Н. контр.	Клятченко Я.М.												
Затв.	Романкевич В.О.								НТУУ «КПІ ім. Ігоря Сікорського», ФПМ, КВ-11				

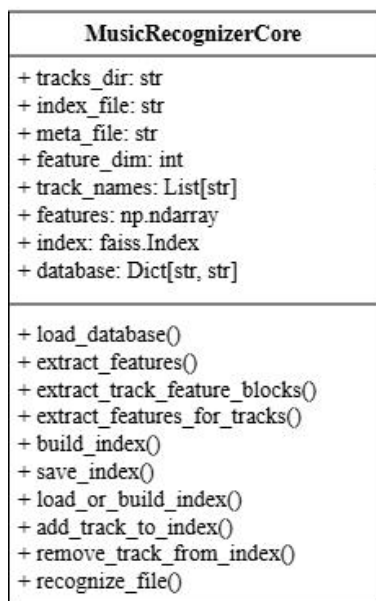
ДОДАТОК 2

Комп'ютерна система розпізнавання музикального ряду

Діаграма класів системи
ІАЛЦ. 045480.005 Д2

Аркушів 1

Київ 2025



					ІАЛЦ. 045480.005 Д2			
Зм	Лист	№ докум.	Підп.	Дата	Комп'ютерна система розпізнавання музикального ряду Діаграма класів системи			
Розроб.	Лабазов В.В.							
Перев.	Павловський В.І.				Лім. Аркуш Аркушів 1 1 НТУУ «КПІ ім. Ігоря Сікорського», ФПМ, КВ-11			
Н. контр.	Клятченко Я.М.							
Затв.	Романкевич В.О.							

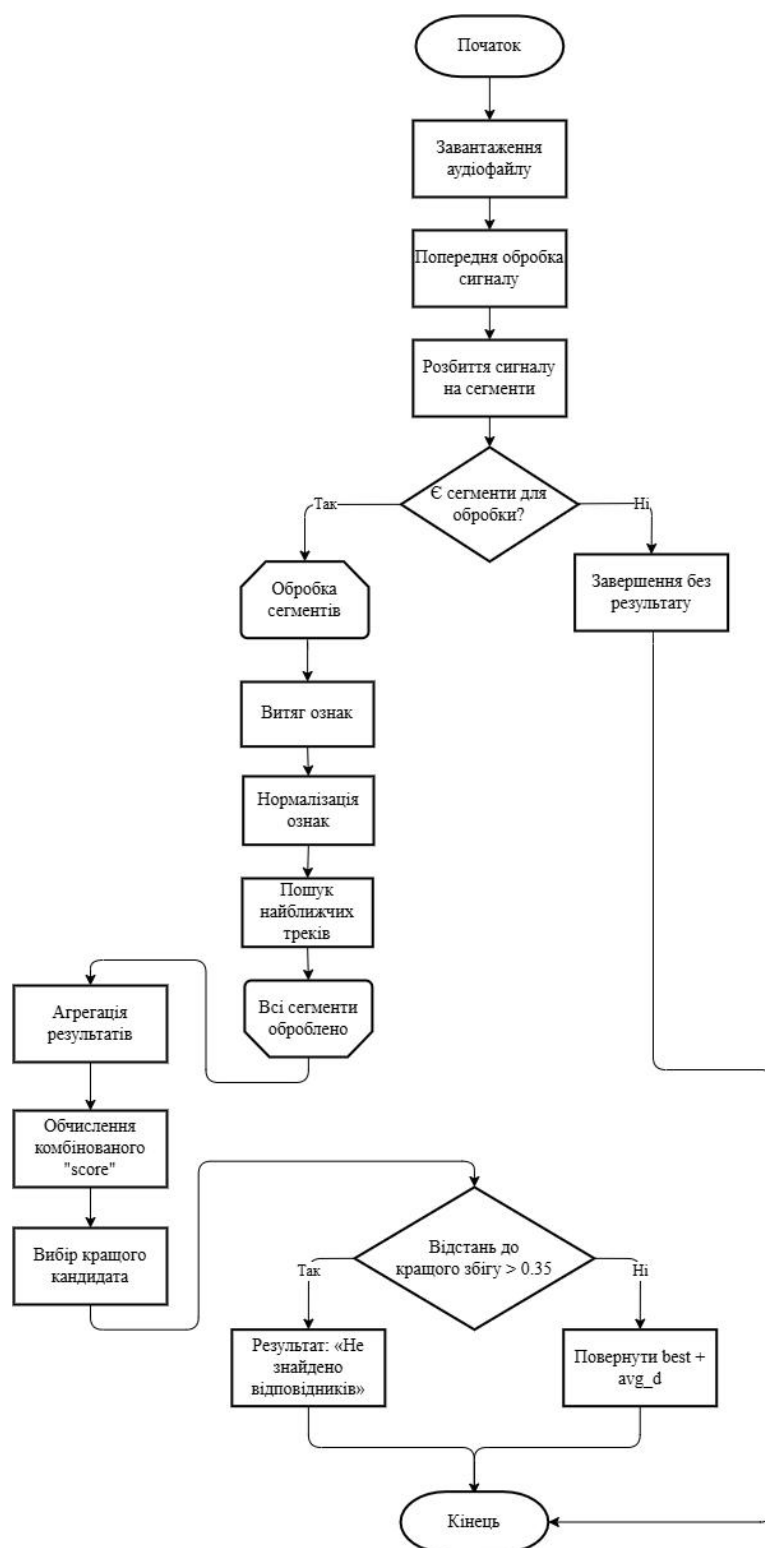
ДОДАТОК 3

Комп'ютерна система розпізнавання музикального ряду

Алгоритм розпізнавання вхідного аудіофрагменту
ІАЛЦ. 045480.006 ДЗ

Аркушів 1

Київ 2025



Зм	Лист	№ докум.	Підп.	Дата
Розроб.		Лабазов В.В.		
Перев.		Павловський В.І.		
Н. контр.		Клятченко Я.М.		
Затв.		Романкевич В.О.		

ІАЛЦ. 045480.006 ДЗ

Комп'ютерна система розпізнавання
музикального ряду
**Алгоритм розпізнавання вхідного
аудіофрагменту**

Лім.	Аркуш	Аркушів
	1	1
НТУУ «КПІ ім. Ігоря Сікорського», ФПМ, КВ-11		

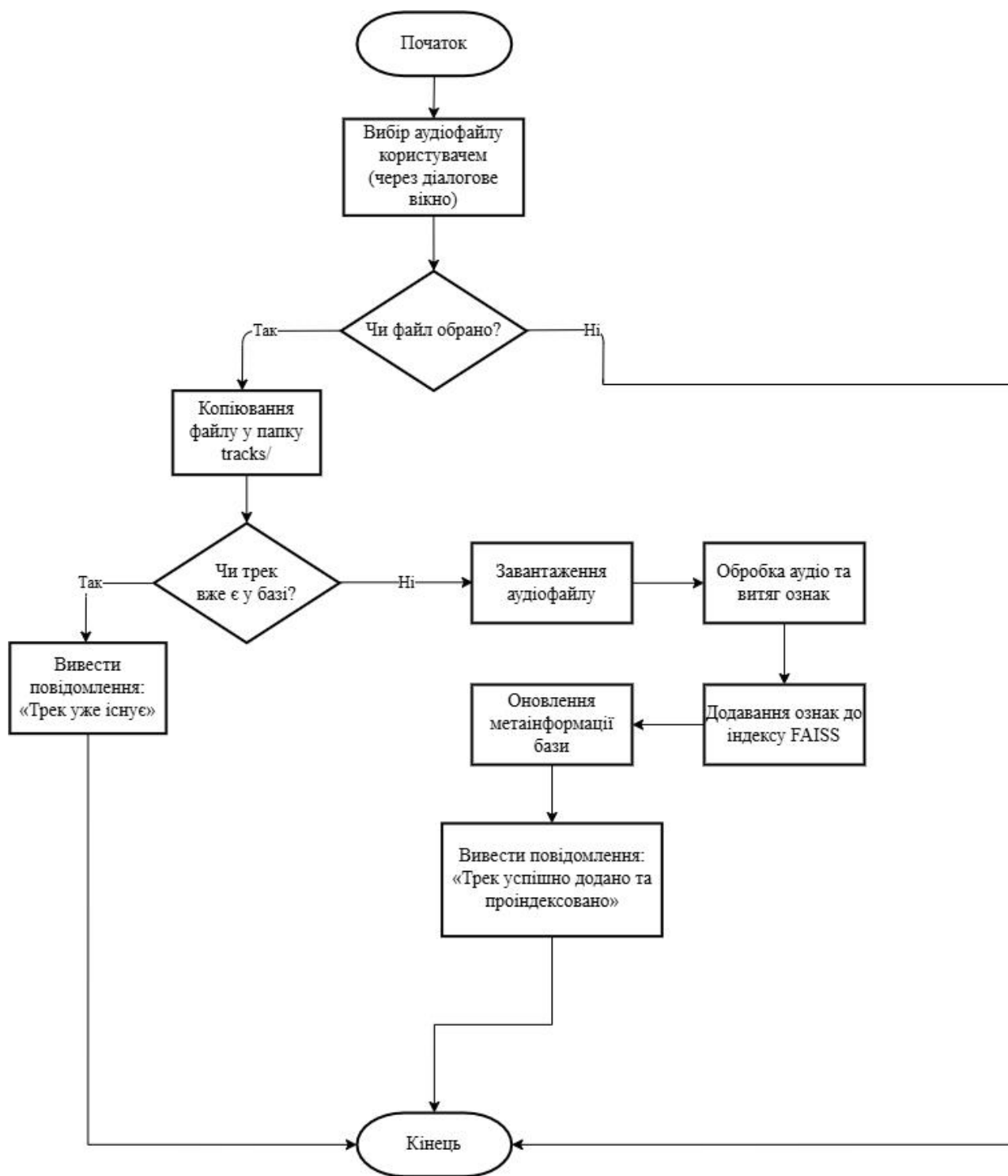
ДОДАТОК 4

Комп'ютерна система розпізнавання музикального ряду

Алгоритм додавання нової музичної композиції до бази індексів
ІАЛЦ. 045480.007 Д4

Аркушів 1

Київ 2025



Зм	Лист	№ докум.	Підп.	Дата
Розроб.	Лабазов В.В.			
Перев.	Павловський В.І.			
Н. контр.	Клятченко Я.М.			
Затв.	Романкевич В.О.			

ІАЛЦ. 045480.007 Д4

Комп'ютерна система розпізнавання
музикального ряду
*Алгоритм додавання нової музичної
композиції до бази індексів*

Літ.	Аркуш	Аркушів
	1	1
НТУУ «КПІ ім. Ігоря Сікорського», ФПМ, КВ-11		