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

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавраза освітньо-професійною програмою «Системи і методи штучного інтелекту» спеціальності 122 «Комп'ютерні науки» на тему: «Система персоналізованих рекомендацій на основі алгоритмів машинного навчання»

Виконав:
студент IV курсу, групи КІ-11
Лобарєв Павло Євгенович _____

Керівник:
доцент кафедри ШІ, к. т. н.
Коваленко Анатолій Єпіфанович _____

Консультант з економічного розділу:
доцент кафедри економічної кібернетики, к.е.н., доцент,
Рощина Надія Василівна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри штучного інтелекту, к.т.н., доцент,
Комариста Богдана Миколаївна _____

Рецензент:
доцент кафедри ММСА, к.т.н.,
Зінченко Артем Юрійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«15» січня 2025 р.

ЗАВДАННЯ
на дипломну роботу студенту
Лобарєву Павлу Євгеновичу

1. Тема роботи «Система персоналізованих рекомендацій на основі алгоритмів машинного навчання», керівник роботи Коваленко Анатолій Єпіфанович, доцент кафедри ШІ, к. т. н. затверджені наказом по університету від «30» травня 2023 р. № 2065-с

2. Термін подання студентом роботи «09» червня 2025 року.

3. Вихідні дані – інтегрований набір Million Song Dataset із піднабором Last.fm Taste Profile та аудіо-характеристиками треків, отриманими через Spotify API.

4. Зміст роботи: дослідження предметної області, теоретичні основи та методи побудови рекомендаційних систем, практичні результати: реалізація, порівняння моделей, оцінка якості, функціонально-вартісний аналіз розробленої системи.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Надія Василівна, доцент, к. е. н.	03.06.2025	03.06.2025

7. Дата видачі завдання «03» лютого 2025 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури за темою	21.04.2025	Виконано
2	Підготовка першого розділу	28.04.2025	Виконано
3	Підготовка другого розділу	05.05.2025	Виконано
4	Розробка програмного продукту	12.05.2025	Виконано
5	Підготовка третього розділу	19.05.2025	Виконано
6	Підготовка економічної частини	26.05.2025	Виконано
7	Оформлення розділів відповідно до нормоконтролю	02.06.2025	Виконано
8	Оформлення дипломної роботи	07.06.2025	Виконано
9	Підготовка презентації доповіді	09.06.2025	Виконано

Студент Павло ЛОБАРЄВ

Керівник Анатолій КОВАЛЕНКО

РЕФЕРАТ

Дипломна робота: 103 с., 11 рис., 5 табл., 20 посилань, додаток.

РЕКОМЕНДАЦІЙНІ СИСТЕМИ, ПЕРСОНАЛІЗАЦІЯ, МАШИННЕ НАВЧАННЯ, КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, ГРАФОВІ НЕЙРОННІ МЕРЕЖІ.

Об’єкт дослідження – автоматизоване формування персоналізованих списків музичних творів на основі аналізу великих обсягів взаємодій «користувач – трек» та контентних характеристик записів.

Сучасні музичні стримінгові сервіси пропонують десятки мільйонів композицій; без інтелектуальної підтримки користувачі стикаються з проблемою інформаційного перевантаження й не можуть оперативно віднаходити музику, що відповідає їх індивідуальним смакам. Ефективна система рекомендацій дає змогу підвищити задоволеність слухачів – показник, критичний для бізнес-метрик онлайн-платформ.

Мета роботи – розробити програмний продукт, що поєднує контентні та колаборативні алгоритми машинного навчання (зокрема матричну факторизацію, графову нейронну модель LightGCN та бібліотеку LightFM) для формування персоналізованих рекомендацій музичних треків із оптимальним використанням обчислювальних ресурсів.

У рамках дослідження інтегровано дані із Million Song Dataset, Spotify API та Last.fm Taste Profile, виконано попередню обробку та збагачення метаданих, реалізовано ансамбль моделей, здатний генерувати рекомендаційні списки топ-N. Програмний продукт створено мовою Python з використанням бібліотек pandas, scikit-learn, LightFM, PyTorch та PyG; передбачено можливість подальшого розширення функціоналу та деплойменту у хмарному середовищі без потреби у виділених GPU-ресурсах.

Практичним результатом є система, що демонструє Recall@20 ≈ 0,18 та MAP@20 ≈ 0, 09 на тестовій підвибірці, суттєво перевершуючи базову випадкову стратегію та традиційні k-NN рекомендації.

ABSTRACT

Bachelor's thesis: 103 p., 11 figures, 5 tables, 20 references.

RECOMMENDER SYSTEMS, PERSONALIZATION, MACHINE LEARNING, COLLABORATIVE FILTERING, GRAPH NEURAL NETWORKS.

The object of this study is the automatic generation of personalized music playlists based on large-scale user–item interaction logs and content-based audio features.

Modern streaming platforms host tens of millions of tracks, causing serious information overload: listeners struggle to discover music that matches their individual taste. Accurate recommendation engines mitigate this problem, increasing user satisfaction and retention – metrics that are vital for online businesses.

The goal of the thesis is to design and implement a software solution that combines content-based and collaborative machine-learning algorithms (including matrix factorization, the LightGCN graph neural model, and the LightFM hybrid framework) to deliver personalized music recommendations while using computational resources efficiently.

The work integrates data from the Million Song Dataset, Spotify API, and the Last.fm Taste Profile, performs thorough preprocessing and metadata enrichment, and builds an ensemble of models deployed as a prototype web service capable of producing real-time top-N recommendations. The system is developed in Python with pandas, scikit-learn, LightFM, PyTorch, and PyG, and is designed for cloud deployment without dedicated GPU requirements.

The resulting engine achieves a Recall@20 of approximately 0.18 and a MAP@20 of ≈ 0.09 on a held-out test split, significantly outperforming random baselines and classical k-NN approaches.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API – Application Programming Interface

BPR – Bayesian Personalized Ranking

CBF – Content-Based Filtering

CF – Collaborative Filtering

CSR – Compressed Sparse Row (формат розріджених матриць)

GNN – Graph Neural Network

GPU – Graphics Processing Unit

ID – Identifier (ідентифікатор)

K-NN – k-Nearest Neighbours

LightFM – Light Factorization Machine (бібліотека/модель)

LightGCN – Light Graph Convolutional Network

MAP@K – Mean Average Precision at K

MLP – Multi-Layer Perceptron

MSD – Million Song Dataset

NDCG – Normalized Discounted Cumulative Gain

PyG – PyTorch Geometric (бібліотека для GNN)

Recall@K – Recall at K (повнота на рівні K)

TF-IDF – Term Frequency–Inverse Document Frequency

ВСТУП

Сучасний[1] цифровий простір пропонує користувачам майже необмежений доступ до інформації й контенту: десятки мільйонів пісень у музичних сервісах, тисячі фільмів та серіалів у відеострімінгу, мільйони товарів в електронній комерції. Зіткнення з таким масштабом вибору неминуче породжує проблему інформаційного перевантаження: людина фізично не здатна самостійно переглянути величезний каталог і швидко знайти саме те, що відповідає її індивідуальним уподобанням. Саме тому рекомендаційні системи стали невід'ємною складовою сучасних онлайн-платформ, допомагаючи формувати адресні поради та підвищувати задоволеність аудиторії. Проте побудова дійсно точних і водночас ефективних рекомендаційних алгоритмів залишається складним завданням, що потребує поєднання різних підходів машинного навчання, ретельної роботи з даними та збалансованої оцінки якості.

Особливо гостро питання персоналізації постає у музичній індустрії. Музичні стримінгові сервіси на кшталт Spotify, Apple Music чи YouTube Music володіють колекціями, що налічують десятки мільйонів треків. Навіть досвідчений меломан не може осягнути такий репертуар без допоміжних інструментів. Успішність платформи безпосередньо залежить від здатності швидко й точно пропонувати слухачеві нові композиції, що з великою ймовірністю припадуть йому до смаку, відкриваючи при цьому менш відомі роботи з «довгого хвоста» каталогу. У цьому контексті поєднання колаборативних методів (які враховують колективні вподобання схожих користувачів) та контентних підходів (аналіз аудіо-характеристик, жанрів і тегів треків) видається найбільш перспективним шляхом, адже дозволяє

[1]Тут і нижче використані такі інструменти штучного інтелекту як чат-бот з генеративним штучним інтелектом ChatGPT виключно для корегування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КПІ ім. Ігоря Сікорського (протокол №11 Вченої ради КПІ ім. Ігоря Сікорського від 11 грудня 2023 р.).

компенсувати недоліки кожного з алгоритмів окремо.

Предметом дослідження обрано побудову системи персоналізованих рекомендацій музичних треків із використанням відкритих великих датасетів Million Song Dataset, даних Last.fm та аудіоознакових описів Spotify. Працюючи з неявним зворотним зв'язком (фактами прослуховування замість явних оцінок), необхідно сформувати модель, здатну якісно ранжувати більший каталог композицій, враховуючи і поведінкові закономірності, і власне вміст музики. Основна увага зосереджена на тому, щоб досягти прийнятного балансу між точністю рекомендацій (Recall@20, MAP@20) та обчислювальною вартістю моделей, що особливо актуально для систем, які повинні працювати у реальному часі й масштабуватися на мільйони користувачів.

Пояснювальна записка складається з чотирьох розділів. Перший розділ окреслює актуальність персоналізованих рекомендацій, подає стислий огляд існуючих підходів та формулює завдання, яке розв'язується в роботі. Другий розділ детально висвітлює математичні та алгоритмічні засади: від класичної колаборативної фільтрації до сучасних моделей на базі графових нейронних мереж і гібридних факторизацій. У третьому розділі наведено практичні результати: опис підготовки даних, реалізацію кількох варіантів моделей, їх порівняльний аналіз за ключовими метриками, а також візуалізації, що ілюструють поведінку системи. Заключний розділ узагальнює отримані висновки, окреслює обмеження проведеного дослідження та можливі напрями подальшого удосконалення. Таким чином, робота послідовно демонструє шлях від постановки проблеми до експериментальної перевірки та оцінювання рекомендаційної системи, спираючись на сучасні досягнення машинного навчання.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність роботи

У сучасну цифрову епоху користувачі стикаються з надлишком інформації та величезною кількістю вибору в інтернет-сервісах. Онлайн-магазини пропонують сотні тисяч товарів, медіаплатформи містять безліч фільмів, серіалів, музичних треків тощо. У таких умовах виникає проблема інформаційного перевантаження: людині дедалі складніше знайти потрібний і цікавий саме їй контент серед океану доступних даних [1]. Рекомендаційні системи з’явилися як важливий інструмент для подолання цього перевантаження, оскільки вони автоматично відфільтровують і персоналізують контент під уподобання конкретного користувача. По суті, система рекомендацій аналізує великі масиви даних про користувачів і об’єкти та видає індивідуальні поради щодо того, що переглянути чи придбати далі [1]. Це значно зменшує час і зусилля, необхідні людині для пошуку релевантної інформації, підвищуючи зручність користування онлайн-сервісами.

Персоналізовані рекомендації сьогодні впроваджені на більшості популярних платформ і довели свою ефективність. За оцінками компанії McKinsey, близько 35% продажів на Amazon генерується саме завдяки системі рекомендацій товарів, а приблизно 75% контенту, що переглядається на Netflix, визначається алгоритмічними рекомендаціями [2]. Іншими словами, значна частка рішень користувачів щодо покупок чи вибору мультимедійного контенту формується під впливом рекомендаційних алгоритмів. Якісна система рекомендацій здатна не лише заощадити час користувача, але й істотно підвищити його задоволеність сервісом та лояльність. З погляду бізнесу, персоналізація контенту сприяє утриманню аудиторії та зростанню доходів компанії [1]. Недарма провідні компанії (Netflix, Amazon, YouTube, Spotify тощо) інвестують значні ресурси в розвиток власних рекомендаційних платформ, адже вони стали ключовим елементом конкуренції за увагу користувача.

Відтак, тема персоналізованих рекомендацій є надзвичайно актуальною як у промисловості, так і в наукових колах. Рекомендаційні алгоритми безперервно вдосконалюються, залучаючи методи штучного інтелекту та машинного навчання для підвищення точності й корисності порад. Сфера застосування таких систем стрімко розширюється: нині вони використовуються не лише в електронній комерції чи стримінговому відео, але й у сферах освіти, туризму, фінансів, охорони здоров'я тощо [3]. Окремо варто відзначити домен музичних стримінгових сервісів (Spotify, Apple Music, Last.fm та ін.), де кількість доступних пісень обчислюється десятками мільйонів. Без персоналізованих рекомендацій користувачам було б вкрай складно орієнтуватися в такому величезному каталозі музики та відкривати нові вподобані композиції [4]. Саме тому розробка ефективних алгоритмів рекомендацій є на часі, а обрана тема дипломної роботи має як теоретичну, так і прикладну значущість.

1.2 Огляд підходів до рекомендаційних систем

Існує кілька концептуальних підходів до побудови рекомендаційних систем. За способом генерування порад їх можна поділити на методи колаборативної фільтрації, контентні (засновані на вмісті), гібридні, а також ряд інших (демографічні, знання-орієнтовані тощо) [1]. Найбільш поширеними та успішними на практиці є саме колаборативні та контентно-орієнтовані підходи, тому з них почнемо огляд. У сучасних системах також часто використовують поєднання кількох підходів (гібридні системи) і новітні алгоритми машинного навчання для підвищення якості рекомендацій – про це також зазначимо наприкінці розділу.

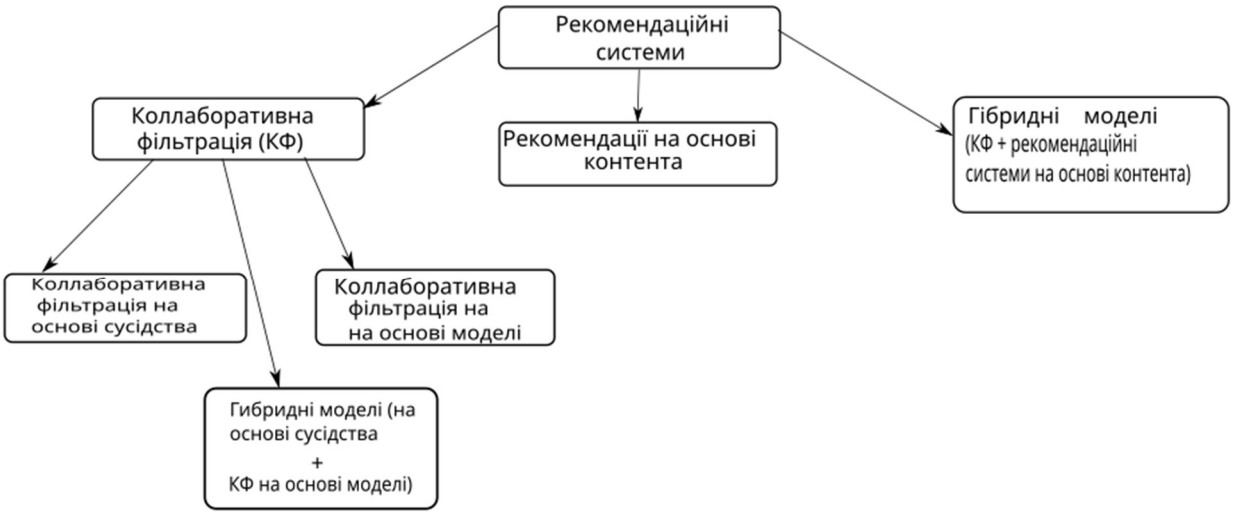

Рисунок 1.1 – Рекомендаційні системи [16]

1.2.1 Колаборативна фільтрація

Колаборативні рекомендаційні системи (від англ. collaborative filtering) базуються на аналізі колективного досвіду і поведінки багатьох користувачів. Ідея полягає в тому, що для активного користувача знаходиться група «сусідів» – інших користувачів зі схожими вподобаннями або оцінками – і на основі того, що подобається цим схожим користувачам, формуються рекомендації [1]. Іншими словами, система шукає закономірності в матриці взаємодій «користувач-об’єкт» і прогнозує, які об’єкти можуть сподобатися певному користувачу, спираючись на вподобання подібних людей. Колаборативна фільтрація може реалізовуватися двома основними способами: на основі користувачів або на основі об’єктів. У першому випадку для цільового користувача знаходять найбільш подібних користувачів (за історією оцінок чи поведінки) і рекомендують йому об’єкти, високо оцінені тими «сусідами». У другому випадку аналізуються подібності між самими об’єктами (наприклад, товарами чи треками), і користувачу радять об’єкти, схожі на ті, що він уже вподобав. Колаборативні методи добре працюють без необхідності явних

характеристик об’єктів – система автоматично «навчається» на даних і виявляє приховані патерни уподобань. Важливо, що такі моделі здатні рекомендувати несподівані, нові для користувача елементи (за рахунок орієнтації на смаки схожих людей), тим самим відкриваючи йому нові інтереси [1]. Водночас колаборативні системи мають і недоліки. Найвідоміша проблема – це «cold start», або проблема холодного старту: система не може дати якісну рекомендацію для нового користувача (у якого ще немає історії) або щодо нового об’єкта (який ще ніхто не оцінив), оскільки бракує даних для аналізу. Також колаборативні методи страждають від розрідженості даних (більшість користувачів оцінюють лише малу частку об’єктів), що ускладнює пошук сусідів, та можуть давати надто узагальнені рекомендації популярних об’єктів, оминаючи вузькоспеціалізовані інтереси.

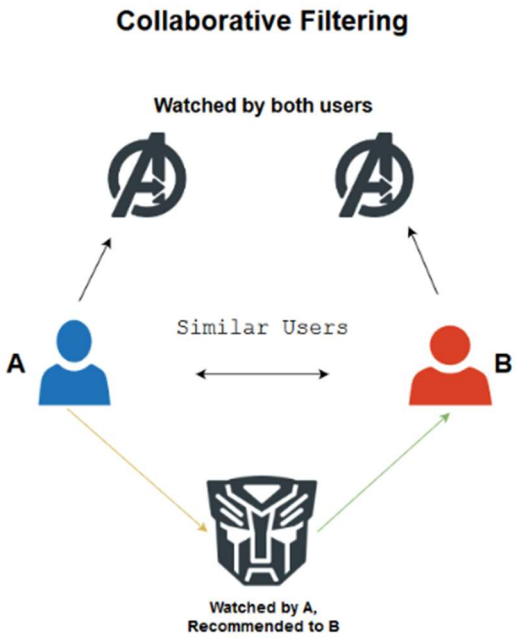

Рисунок 1.2 – Колаборативна фільтрація [17]

1.2.2 Контентні рекомендаційні системи

Контентно-орієнтований підхід (англ. content-based filtering) грунтується на аналізі властивостей та характеристик самих об’єктів, що рекомендуються, і співставленні їх з інтересами користувача. Для кожного користувача будується профіль уподобань – набір характеристик або ключових слів, що описують об’єкти, які він вподобав у минулому. Аналогічно, для кожного об’єкта (фільму, пісні, товару) відомі певні атрибути: жанр, опис, виконавець, технічні параметри тощо. На основі порівняння профілю користувача з характеристиками доступних об’єктів система визначає, які об’єкти найбільше відповідають уподобанням цього користувача, і рекомендує їх [1]. Наприклад, якщо слухач переважно слухає рок-музику 1970-х років, то контентна система рекомендацій радитиме йому інші рок-треки 70-х, спираючись на збіг атрибутів (жанр, рік, виконавець тощо). Перевагою контентного підходу є те, що він не залежить від даних інших користувачів – рекомендації генеруються суто з індивідуальної історії та вподобань. Це означає, що відсутня проблема холодного старту для нових об’єктів: якщо відомі характеристики нового треку, система зможе порекомендувати його відповідним слухачам, навіть якщо ніхто ще не встиг його оцінити [1]. Контентні системи також легко масштабуються на велику кількість користувачів, оскільки для кожного достатньо знаходити схожі об’єкти за його профілем, незалежно від бази інших користувачів. Однак, і цей підхід має обмеження. По-перше, потрібні достатньо інформативні ознаки об’єктів: якщо опис товару або метадані треку мізерні, то складно зробити точну рекомендацію. Часто буває необхідно залучати експертні знання для визначення релевантних характеристик, особливо в специфічних доменах. По-друге, контентні фільтри схильні до проблеми вузької спеціалізації: система радить користувачу лише дуже подібні до вже відомих об’єкти, через що новизна рекомендацій обмежена. Користувач може опинитися в інформаційному «міхурі», отримуючи однотипні

пропозиції і не дізнаючись про щось зовсім відмінне від його минулих вподобань.

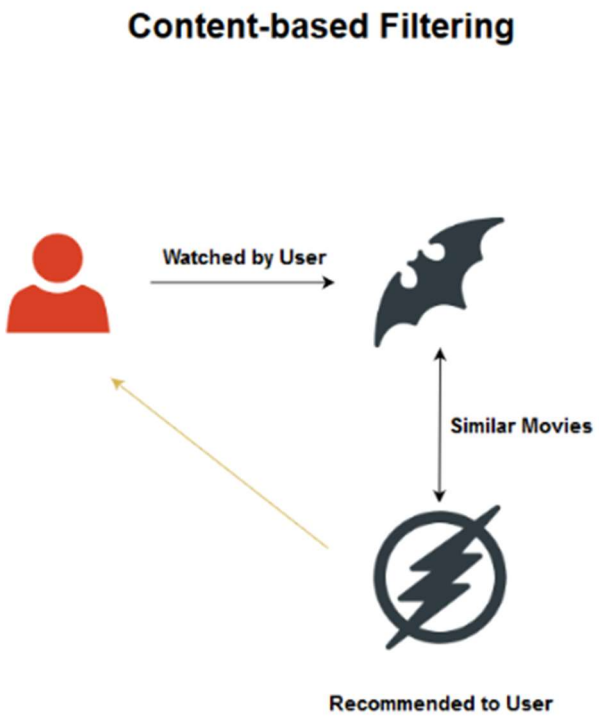

Рисунок 1.3 – Контентні рекомендаційні системи [17]

1.2.3 Гібридні системи

Жоден з наведених підходів не є універсально найкращим; кожен має сильні та слабкі сторони. Тому в багатьох сучасних реалізаціях застосовують гібридні рекомендаційні системи, які комбінують декілька методів для досягнення кращої ефективності. Головна мета гібридизації – компенсувати недоліки одного підходу перевагами іншого [1]. Існують різні стратегії поєднання: можна будувати спільну модель, що одночасно враховує і контент, і колаборативні зв’язки (так звані моделі зі спільним простором ознак); можна обчислювати рекомендації окремо за різними методами, а потім об’єднувати їх

результати (наприклад, шляхом зваженого підсумовування рейтингів або чергування рекомендацій); можна динамічно перемикатися між методами залежно від ситуації (наприклад, якщо для нового користувача недоступна колаборативна фільтрація через відсутність даних, система тимчасово використовує контентний підхід) [1]. Гібридні системи дозволяють досягти більшої точності рекомендацій та стійкості до проблем холодного старту і розрідженості даних. Класичним прикладом є підхід, який використовувався сервісом Netflix: поєднання колаборативної фільтрації (для врахування уподобань мільйонів користувачів) з елементами контентного аналізу (метадані фільмів, жанрові вподобання) дозволило суттєво підвищити якість рекомендацій, що підтверджено результатами відомого конкурсу Netflix Prize. Загалом, правильно сконструйована гібридна система може перевершувати окремі базові методи, особливо в реальних великомасштабних застосуваннях.

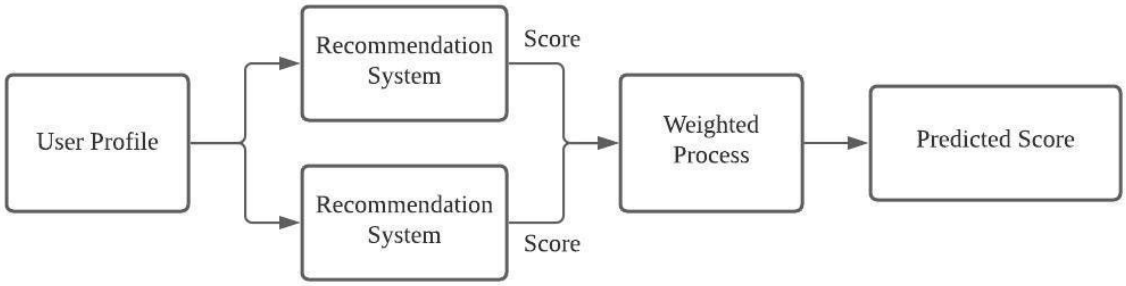

Рисунок 1.4 – Гібридні системи [18]

1.2.4 Сучасні підходи та новітні тенденції

Рекомендаційні системи продовжують активно еволюціонувати, переймаючи досягнення суміжних галузей штучного інтелекту. Останніми роками спостерігається перехід від традиційних алгоритмів до більш складних

моделей, зокрема глибинного навчання, графових моделей і навіть контекстуальних або когнітивних підходів [5]. Сучасні системи рекомендацій усе частіше використовують нейронні мережі для побудови прихованих представлень (ембеддингів) користувачів та об’єктів. Архітектури на основі глибоких нейронних мереж (Multi-Layer Perceptrons, автоенкодери, рекурентні та трансформерні моделі) дозволяють враховувати складні нелінійні взаємозв’язки в даних і формувати більш точні прогнози вподобань. Наприклад, модель Neural Collaborative Filtering замінює класичну матричну факторизацію на нейромережу, що навчається передбачати взаємодії «user–item», а такі підходи, як Wide & Deep чи DeepFM, поєднують лінійні й нелінійні компоненти для врахування різних видів ознак. Окрім цього, набувають популярності алгоритми, що враховують додатковий контекст: часові аспекти (напр. session-based recommendations – рекомендації на основі поточної сесії), місцеположення, соціальні зв’язки тощо. З’являються також рекомендаційні системи на основі знань (knowledge-based RS), які використовують семантичні знання або онтології для виведення нових рекомендацій. Ще одна сучасна тенденція – застосування методів підкріплювального навчання (reinforcement learning) з метою оптимізації довгострокової взаємодії з користувачем, коли система навчається послідовно пропонувати контент, що максимізує загальне задоволення користувача з часом. Нарешті, варто згадати й про інтеграцію моделей обробки природної мови та великих мовних моделей у рекомендаційні системи – приміром, для покращення розуміння відгуків користувачів чи вмісту описів об’єктів. Таким чином, галузь рекомендаційних систем нині перебуває на стику кількох напрямів AI, і новітні алгоритми дозволяють будувати все більш персоналізовані та «розумні» рекомендації, що враховують багатовимірний контекст взаємодії з користувачем [5]. Усі ці підходи будуть розглянуті детальніше у наступному розділі, присвяченому аналізу сучасних методів побудови рекомендаційних систем.

1.3 Постановка задачі

Метою даної роботи є розробка системи персоналізованих рекомендацій для музичних треків. Як було відзначено, музична індустрія є яскравим прикладом сфери, де рекомендаційні алгоритми вкрай потрібні. В онлайн-сервісах доступні десятки мільйонів пісень, і користувач фізично не здатен самостійно переглянути настільки великий каталог, щоб відібрати музику собі до смаку [6]. Стримінгові платформи (Spotify, Apple Music, YouTube Music тощо) та музичні соціальні мережі (Last.fm і ін.) вже давно впроваджують механізми автоматичних рекомендацій, що аналізують поведінку слухача і пропонують нові треки на основі його вподобань. Проте побудова такої системи – непросте завдання, що потребує поєднання різних даних та алгоритмів. У рамках цієї дипломної роботи ставиться конкретна задача: створити модель рекомендацій, яка на вході отримує дані про історію прослуховування музичних треків низкою користувачів і здатна генерувати персональні списки рекомендацій нових треків для кожного користувача. Інакше кажучи, система повинна передбачити, які пісні ймовірно сподобаються тому чи іншому слухачеві, базуючись на інформації про його смаки та поведінку інших слухачів зі схожими вподобаннями.

Для розв’язання поставленої задачі планується використати кілька джерел даних. Основою є Million Song Dataset (MSD) – відкритий набір, що містить аудіо-фічі та метадані для мільйона популярних музичних треків [6]. Зокрема, MSD надає автоматично обчислені характеристики аудіозаписів (темп, тональність, спектральні особливості тощо) та базову інформацію про композиції (назва, виконавець, рік випуску і т.д.), отримані завдяки сервісу Echo Nest (нині інтегрований зі Spotify). Важливо, що сам аудіоконтент до набору не входить – лише числові ознаки й описи, що робить MSD зручним для машинного навчання на великих обсягах даних. Окрім того, до складу MSD входять і пов’язані піднабори: зокрема, піднабір Last.fm з народними тегами (жанровими

мітками) та коефіцієнтами схожості для треків, а також піднабір Taste Profile – анонімні дані про прослуховування від реальних користувачів (так званий user-song listening history) [6]. Дані Taste Profile особливо цінні, адже саме вони містять інформацію про те, які користувачі які треки слухали (і скільки разів), що фактично є матрицею взаємодій «користувач– трек» для колаборативної фільтрації. У цій роботі планується використати зазначений піднабір користувацьких даних Last.fm як основу для моделювання вподобань. Додатково буде задіяна інформація зі Spotify – передусім аудіохарактеристики треків, надані через Spotify API (такі як валентність, енергійність, танцювальність композиції тощо), а також жанрові категорії та популярність треків. Інтеграція даних Spotify дозволить доповнити MSD свіжими ознаками, розширивши опис музичних об'єктів. Комбінація декількох джерел (MSD + Last.fm + Spotify) дасть можливість побудувати більш багату модель: врахувати і контентні ознаки пісень, і колаборативні зв'язки між користувачами через історію прослуховувань, тобто реалізувати певний гібридний підхід у рекомендаціях.

Формально задачу можна визначити наступним чином. Нехай U – множина користувачів (слухачів), а I – множина музичних об'єктів (треків), доступних у системі. Маємо історичні дані про взаємодії деякої підмножини пар користувач–трек (наприклад, факт прослуховування або рейтинг, якщо є). На основі цих наявних даних необхідно побудувати функцію $f: U \times I \to R$, яка для кожного користувача $u \in U$ і потенційного треку $i \in I$ видає певний числовий скоровий показник – умовну міру цікавості або корисності треку i для користувача u. Інтуїтивно, чим вище значення $f(u, i)$, тим більш релевантним і бажаним для даного користувача вважається трек. Функція f фактично є прогнозною моделлю уподобань: вона вчиться на основі відомих прикладів (які треки користувач слухав раніше) і потім передбачає оцінки для невідомих комбінацій користувач/трек [1]. Після побудови такої моделі система рекомендацій може генерувати для кожного користувача персональний рейтинг об'єктів – впорядкований список треків за спаданням прогнозованого значення $f(u, i)$. В реальній системі користувачу зазвичай показують топ-N позицій цього списку

(наприклад, *N*=20), тобто формується підбірка з 20 найбільш рекомендованих пісень, що він ще не слухав.

Оскільки в задачі музичних рекомендацій ми маємо справу переважно з неявною зворотною інформацією (implicit feedback) – фактами прослуховування замість явних рейтингових оцінок – модель $f(u, i)$, по суті, оцінюватиме ймовірність або ступінь того, що користувач u зацікавиться треком і. Навчання моделі може здійснюватися методами машинного навчання: наприклад, через факторизацію матриці взаємодій (матрицю прослуховувань) або за допомогою нейронної мережі, яка приймає на вхід ознаки користувача і треку та прогнозує їхню сумісність. В даній роботі буде обрано оптимальний алгоритм на основі аналізу сучасних підходів (розглядаються варіанти колаборативної фільтрації, у тому числі матрична факторизація, та гібридні моделі з використанням контентних ознак треків). Важливим аспектом є належна підготовка даних: очистка та нормалізація ознак, поділ вибірки на тренувальну й тестову, вибір стратегії негативного семплінгу для неявних даних тощо. Правильна формалізація задачі і підхід до навчання моделі безпосередньо вплинуть на якість фінальних рекомендацій.

Після реалізації моделі необхідно оцінити її ефективність. Для цього використовуються стандартні метрики якості рекомендацій, які порівнюють прогнозовані системою списки з відомими вподобаннями користувачів у тестових даних. Оскільки ми генеруємо ранжований список рекомендацій для кожного користувача, ключовими є метрики топ-N ранжування. В даній роботі будуть застосовані, зокрема, Recall@20 та MAP@20.

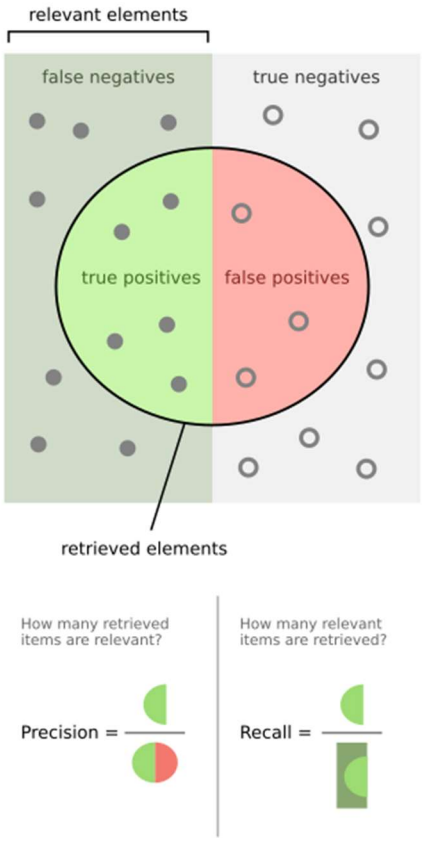

Рисунок 1.5 – Precision and recall [19]

1.3.1 Recall@20

Recall@20 (повнота на рівні 20) показує, яку частку від усіх релевантних для користувача елементів складають запропоновані системою топ-20 треків. Іншими словами, це доля вподобаних користувачем пісень із тестового набору, що потрапили до рекомендаційного списку довжиною 20 [7]. Чим вище Recall@20, тим більшу частину важливих для користувача об’єктів змогла знайти система в межах перших двадцяти рекомендацій.

1.3.2 MAP@20

MAP@20 (Mean Average Precision на рівні 20) – усереднений показник точності ранжування. Він враховує не лише наявність релевантних елементів у списку, а й їхні позиції. Обчислення MAP@20 відбувається у кілька кроків: спочатку для кожного користувача рахується Average Precision @20 – середня точність відповідного рекомендаційного списку, яка підвищується, коли релевантні треки з'являються вище у списку. Далі MAP@20 отримують як середнє значення Average Precision по всіх користувачах [7]. Значення MAP лежить у діапазоні від 0 до 1, причому 1.0 відповідає ідеальному ранжуванню (усі релевантні об'єкти на початку списку). Таким чином, MAP@20 відображає, наскільки добре система не тільки знаходить значущі для користувача треки, а й правильно їх упорядковує за пріоритетом.

Зазначені метрики дозволяють об'єктивно порівняти побудовану модель рекомендацій з базовими підходами та між різними варіантами алгоритмів. У разі необхідності можуть бути додатково використані й інші показники якості – наприклад, Precision@N (точність рекомендацій), NDCG@N (нормалізована кумулятивна згода рангів) тощо – однак основну увагу в межах цієї роботи зосереджено саме на Recall@20 та MAP@20 як найбільш поширених і інформативних метриках для задачі рекомендації музичних треків.

Висновки до розділу 1

У першому розділі було розглянуто концепцію персоналізованих рекомендаційних систем та обґрунтовано актуальність їх розробки. В умовах надлишку інформації такі системи відіграють ключову роль у відборі та фільтрації контенту під індивідуальні потреби користувачів, що підтверджується

успіхами світових компаній і численними дослідженнями. Проведено огляд основних типів рекомендаційних систем: колаборативних, контентних, гібридних та сучасних підходів на основі глибинного навчання. Кожен з методів має свої переваги і недоліки, тому вибір підходу залежить від природи даних та поставленої задачі. Для подальшого розвитку роботи визначено конкретну проблему – побудову системи рекомендацій музичних треків. Сформульовано постановку задачі та її формальну модель: рекомендатор має на основі історії прослуховувань передбачати рейтинг релевантності треків для користувачів і формувати персональні списки рекомендацій. Описано використовувані набори даних (Million Song Dataset, Last.fm, Spotify) та їхню роль у створенні моделі. Визначено метрики оцінки якості результатів (Recall@20, MAP@20), що будуть застосовані для верифікації ефективності розробленої системи. Таким чином, розділ 1 закладає теоретичне підґрунтя для подальшої реалізації та експериментальної перевірки рекомендаційної системи у наступних розділах.

РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ

У цьому розділі представлено розробку системи персоналізованих рекомендацій, що поєднує контентне фільтрування та колаборативну фільтрацію з використанням алгоритмів машинного навчання. Описано етапи попередньої обробки даних (нормалізація числових ознак, перетворення текстових тегів у векторні представлення тощо), побудову моделей рекомендацій на основі схожості (контентної та колаборативної), їх гібридизацію шляхом ансамблювання результатів, а також модель факторизації матриці взаємодій з урахуванням додаткових ознак (LightFM). Наведено математичні основи застосованих методів: методи нормалізації та масштабування ознак, обчислення TF-IDF, косинусної схожості, алгоритму найближчих сусідів ItemKNN, формування гібридної матриці схожості та функції прогнозування рейтингу на основі скалярного добутку. Окремо розглянуто метрики оцінки якості рекомендацій (MAP@K, Recall@K).

2.1 Нормалізація та масштабування числових ознак

Більшість алгоритмів машинного навчання вимагають приведення числових даних до співставних масштабів. Ненормалізовані ознаки з різними діапазонами значень можуть по-різному впливати на модель, навіть якщо їх важливість рівноцінна. Нормалізація та масштабування ознак – це процеси перетворення вихідних числових значень до певного стандартного діапазону (наприклад, $[0, 1]$ або до нульового середнього і одиничного стандартного відхилення). Зокрема, часто використовується мін–макс нормалізація, яка лінійно масштабує значення x до діапазону $[0,1]$ за формулою (2.1).

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}, \quad (2.1)$$

де x_{min} і x_{ma} – мінімальне та максимальне значення ознаки відповідно.

В результаті такої нормалізації найменше значення перетворюється на 0, найбільше – на 1, а проміжні значення пропорційно масштабуються між ними. Для ознак з розподілом, близьким до нормального, доцільно застосувати стандартизацію (z-нормалізацію), що полягає у відніманні середнього та діленні на стандартне відхилення:

$$z = \frac{x - \mu}{\sigma}. \quad (2.2)$$

Масштабування ознак забезпечує коректне обчислення метрик відстані або схожості між об'єктами та сприяє більш стабільному навчанню моделей.

2.2 Формування ознак із текстових тегів (TF-IDF)

Багато об’єктів (елементів рекомендаційної системи), таких як фільми чи товари, описуються текстовими тегами або ключовими словами. Ці текстові дані необхідно перетворити у числові вектори ознак для подальшого використання в моделях. У даній роботі здійснено побудову ознак із текстових тегів за допомогою методу TF-IDF (Term Frequency-Inverse Document Frequency). Спочатку обчислюється частота терміну (тегу) t в описі (або наборі тегів) конкретного елемента d:

$$tf_{t,d} = \frac{n_{t,d}}{\sum_{t' \in d} n_{t',d}}, \quad (2.3)$$

де $n_{t,d}$ – кількість появ терміна t в елементі (документі) d, а знаменник – загальна кількість усіх термінів у цьому елементі.

Далі обчислюється обернена частота документів для терміну t:

$$idf_t = log\frac{N}{df_t}, \tag{2.4}$$

де N – загальна кількість документів (елементів) у корпусі;
df_t – кількість документів, що містять термін t.

Множенням цих двох величин отримуємо вагу TF-IDF терміна t для елемента d:

$$w_{t,d} = tf_{t,d} \cdot idf_t. \tag{2.5}$$

Вектор, складений з ваг $w_{t,d}$ для всіх термінів, характеризує елемент у просторі ознак, побудованих на основі тегів. Таким чином, текстові описи перетворюються на матрицю ознак розміру $|Items| \times |Terms|$, де кожен рядок відповідає одному елементу, а стовпці – окремим термінам (тегам).

2.3 Перетворення року випуску на категоріальні ознаки

Окрім тегів, для опису елементів можуть використовуватися й інші атрибути, наприклад рік випуску (для фільмів, книг тощо). Рік є числовою ознакою, але його можна інтерпретувати як категоріальну, розбивши діапазон років на інтервали або розглядаючи кожен рік як окрему категорію. У найпростішому випадку застосовується one-hot кодування: створюється бінарний вектор ознак для року випуску, де кожен можливий рік відповідає окремому бінарному атрибуту. Значення такого атрибуту дорівнює 1 для року,

що відповідає даному елементу, і 0 – для всіх інших. Наприклад, якщо елемент випущено у 2020 році, то вектор ознак матиме 1 в позиції "2020" і 0 в інших позиціях, що відповідають іншим рокам. Розмірність простору таких ознак дорівнює кількості унікальних років у наборі даних. Альтернативним підходом є бінування року – об'єднання значень у більші інтервали (наприклад, десятиліття) з подальшим представленням цих інтервалів як окремих категорій. Така дискретизація може зменшити розмірність та згладити незначні відмінності між близькими за часом випуску елементами. У даній роботі роки не групуються за інтервалами: застосовано one-hot кодування для кожного конкретного значення року випуску.

2.4 Матриця "item-feature" та обчислення косинусної схожості

Після формування всіх потрібних ознак для кожного елемента будується матриця "елемент-ознака". Кожен рядок цієї матриці відповідає окремому елементу (айтему), а стовпці – числовим ознакам, текстовим характеристикам та категоріальним ознакам, описаним вище. Числові ознаки попередньо нормалізовано (див. розділ 2.1), текстові представлені у вигляді TF-IDF векторів (розділ 2.2), а рік випуску – у вигляді набору бінарних змінних (розділ 2.3). Об'єднавши ці дані, отримуємо високорозмірний простір ознак, який описує вміст кожного елемента системи.

На основі матриці "item-feature" можна обчислити міру контентної схожості між будь-якими двома елементами, порівнюючи їх відповідні вектори ознак. Найпоширенішою мірою є косинусна схожість (2.6), що визначається як косинус кута між векторами x_i та x_j двох елементів:

$$sim_{cos}(i,j) = \frac{x_i \cdot x_j}{||x_i|| \, ||x_j||}, \tag{2.6}$$

де $x_i \cdot x_j = \sum_{k=1}^{m} x_{i,k}, x_{j,k}$ – скалярний добуток векторів ознак елементів i та j; $|x_i|$ – евклідову норму (довжину) вектора x_i.

Косинусна схожість набуває значень від 0 до 1: значення 1 означає, що напрямки векторів ознак двох елементів збігаються (тобто елементи ідентичні у просторі ознак), 0 – що вектори ортогональні (елементи не мають нічого спільного за вибраними ознаками). Обчисливши косинусну схожість для всіх пар елементів, формуємо матрицю схожості S_{CBF} розміром $|Items| \times |Items|$. Елемент $S_{CBF}[i, j]$ дорівнює значенню косинусної схожості між елементами i та j на основі їхніх контентних ознак.

2.5 Матриця взаємодій "user-item" та логарифмічна нормалізація

Для реалізації підходу колаборативної фільтрації необхідно врахувати історію взаємодії користувачів з елементами. Вихідні дані про інтереси користувачів подаються у вигляді матриці взаємодій "користувач-елемент" R розміру $|Users| \times |Items|$, де $R_{u,i}$ відображає наявність та інтенсивність взаємодії користувача u з елементом i. Значення $R_{u,i}$ можуть бути явними рейтингами (наприклад, від 1 до 5) або неявними ознаками взаємодії (наприклад, факт перегляду, прослуховування, кліку, кількість прослуховувань тощо).

У разі неявних даних взаємодію часто подають бінарно (0 – немає взаємодії, 1 – взаємодія відбулася). Якщо ж доступна інтенсивність (наприклад, число прослуховувань композиції або кількість переглядів сторінки), доцільно виконати логарифмічну нормалізацію значень, щоб зменшити вплив дуже популярних елементів та користувачів з непропорційно великою активністю. Логарифмічно нормалізоване значення взаємодії можна визначити як:

$$R'_{u,i} = log(1 + R_{u,i}), \tag{2.7}$$

де $R_{u,i}$ – початкове значення (кількість взаємодій);

$R'_{u,i}$ – нормалізоване.

Логарифмічна функція зростає сублінійно, тому різниця між 1 та 5 взаємодіями буде значною, тоді як різниця між 101 та 105 – майже непомітною. Це допомагає зменшити ефект "довгого хвоста" та шум від користувачів, які надмірно багато взаємодіють із системою, забезпечуючи більш збалансований внесок різних користувачів та елементів.

2.6 Матриця взаємодій "user-item" та логарифмічна нормалізація

Колаборативна фільтрація на основі схожих елементів (item-based CF) передбачає, що для генерації рекомендацій використовуються шаблони спільної взаємодії користувачів з елементами. Простіше кажучи, два елементи вважаються схожими, якщо їх оцінювали або переглядали одні й ті ж користувачі. Для визначення такої схожості будується матриця, основана на взаємодіях (матриця R). Один зі способів – обчислити косинусну схожість між стовпцями матриці R (кожен стовпець відповідає профілю взаємодій певного елемента серед усіх користувачів). Проте безпосереднє обчислення повної матриці схожості розміру $|Items| \times |Items|$ може бути витратним. Натомість використано алгоритм k найближчих сусідів для елементів (ItemKNN), реалізований у бібліотеці scikit-learn (клас NearestNeighbors).

Алгоритм ItemKNN для кожного елемента знаходить топ- k найбільш схожих до нього інших елементів за профілями взаємодій. Для цього кожен елемент розглядається як вектор розмірності $|Users|$ (відповідний стовпець R), і між такими векторами обчислюється міра близькості чи схожості. Аналогічно до контентного підходу, використано косинусну міру схожості. Таким чином, для кожного елемента отримано список найближчих "сусідів", що мають найбільшу

косинусну схожість за патернами взаємодій користувачів. На основі цих списків формується матриця схожості S_{CF} для колаборативної фільтрації, де значення $S_{CF}[i, j]$ відображає ступінь схожості між елементами i та j за поведінкою користувачів.

2.7 Об'єднання колаборативного та контентного підходів

Контентна та колаборативна моделі схожості мають свої переваги й недоліки, тому логічним кроком є об’єднання їх в єдину гібридну рекомендаційну модель. Це дозволяє врахувати одночасно і схожість за властивостями елементів, і схожість за вподобаннями користувачів. Гібридизація реалізується через побудову комбінованої матриці схожості S як зваженої суми матриць контентної (S_{CBF}) та колаборативної (S_{CF}) схожності:

$$S = \alpha S_{CF} + (1 - \alpha) S_{CBF}, \tag{2.8}$$

де α – ваговий коефіцієнт, $0 \leq \alpha \leq 1$

Цей параметр визначає внесок кожного з підходів: при $\alpha = 0$ використовуються лише контентні ознаки (модель вироджується до чистого контентного фільтрування), при $\alpha = 1$ – лише колаборативні (чиста колаборативна фільтрація), а при проміжних значеннях модель поєднує обидва джерела інформації.

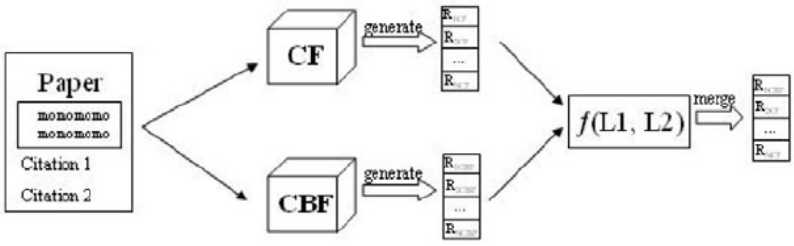

Рисунок 2.1 – Діаграма схеми ансамблю CBF+CF [20]

Правильний вибір α дає змогу досягти балансу між узагальнювальною здатністю колаборативного методу та чутливістю до індивідуальних характеристик контентного методу. У практичній реалізації значення α можна підбирати експериментально, оцінюючи якість рекомендацій на валідаційній вибірці.

2.8 Прогнозування рейтингу та формування рекомендацій

Після отримання остаточної матриці схожості S (наприклад, гібридної матриці для ансамблю CBF+CF або будь-якої з окремих S_{CBF} чи S_{CF}) необхідно навчитися прогнозувати, які саме елементи слід рекомендувати конкретному користувачу. Профіль користувача u можна подати у вигляді рядка матриці взаємодій $R_{u,*}$, що відображає, з якими елементами користувач уже взаємодіяв і з якою інтенсивністю. Використовуючи цей профіль та матрицю схожості між елементами, можна розрахувати прогнозовану оцінку релевантності для будь-якого цільового елемента j шляхом обчислення зваженої суми взаємодій користувача з усіма іншими елементами, взятих з вагами схожості цих елементів з цільовим:

$$\hat{r}_{u,i} = \sum_{i=0}^{|Items|} R_{u,i} \cdot S[i,j], \tag{2.9}$$

де $R_{u,i}$ – інтенсивність взаємодії користувача u з елементом i (нуль, якщо користувач не взаємодіяв з i);

$S[i,j]$ – схожість між елементами i та j.

По суті, ця формула реалізує скалярний добуток вектора-профілю користувача на стовпець j матриці S. В результаті отримуємо значення $\hat{r}_{u,j}$, яке тим вище, чим більше елементів, схожих на j, уже сподобалося користувачу u і чим вищі оцінки (або активність) користувача щодо цих схожих елементів.

Після обчислення $\hat{r}_{u,j}$, для всіх кандидатів j система ранжує їх за спаданням цього прогнозного скору. Ті елементи, що не входять до історії користувача і отримали найвищі значення $\hat{r}_{u,j}$,, рекомендуються користувачу. Зазвичай формується топ-K рекомендацій для кожного користувача, де K – заданий розмір списку (наприклад, $K = 10$).

2.9 Модель факторизації з використанням LightFM

Описані вище підходи базуються на обчисленні схожості безпосередньо за контентними ознаками або взаємодіями, тобто є memory-based методами. Альтернативним підходом є використання model-based алгоритмів, які будують узагальнену модель шляхом навчання на наявних даних. Одним із таких підходів є модель LightFM, що реалізує матричну факторизацію з урахуванням додаткових ознак (так званих side features).

На відміну від чисто колаборативного підходу, LightFM дає змогу включити в модель контентні ознаки елементів (а за наявності – і характеристики користувачів). Перед навчанням моделі необхідно сформувати матриці ознак для елементів і, за потреби, для користувачів. Для кожного елемента формується набір ознак, аналогічний до використаного в контентній моделі.

Текстові ознаки – вектор TF-IDF на основі тегів (див. розділ 2.2), що відображає ключові слова, пов’язані з елементом.

Часові ознаки – категоріальні ознаки року випуску (див. розділ 2.3). Для компактності можливе групування років за періодами (наприклад, десятиліттями), однак у цій роботі використано точне значення року випуску з one-hot кодуванням.

Інші ознаки – за наявності інших релевантних атрибутів (жанри, категорії тощо) їх можна додати аналогічно. У даному наборі даних основний акцент зроблено на тегах та році виходу.

Сформовані ознаки подаються моделі LightFM у вигляді розрідженої матриці item_features розміру $|Items| \times |Features|$, де стовпці відповідають усім можливим ознакам (включно з словами-тегами і категоріями років), а рядок для кожного елемента містить 1 у тих стовпцях, які відповідають наявним ознакам цього елемента (TF-IDF ознаки можуть бути дробовими значеннями, категоріальні ознаки – бінарні 0/1). Якщо доступні ознаки користувачів, формується аналогічна матриця user_features. У разі відсутності явних ознак для користувачів LightFM за замовчуванням використовує унікальний ідентифікатор кожного користувача як його єдину ознаку. Таким чином, модель все одно навчає латентні фактори для кожного користувача, як у стандартній матричній факторизації.

Модель LightFM навчається на матриці взаємодій R шляхом її факторизації з урахуванням наведених вище матриць ознак. Ідея полягає в тому, що кожному унікальному атрибуту (ознаці) – чи то конкретний тег, чи рік, чи ID користувача – відповідає власний латентний вектор у просторі розмірності f (кількість факторів моделі). Під час навчання оптимізуються латентні представлення (ембеддінги) для всіх ознак таким чином, щоб наближати наявні взаємодії. При цьому профіль користувача у просторі факторів можна представити як суму векторів його ознак, а профіль елемента – як суму векторів ознак елемента. Формально:

$$P_u = \sum_{a \in F(u)} e_a, \tag{2.10}$$

$$Q_i = \sum_{b \in F(i)} e_b, \tag{2.11}$$

де $F(u)$ – множина ознак (фіч) користувача u;

$F(i)$ – множина ознак елемента i;

e_a – латентний вектор, асоційований з ознакою a.

Зокрема, до множини $F(u)$ входить індикатор самого користувача (його ID), а до $F(i)$ – індикатор елемента (ID) разом із контентними ознаками цього елемента. Таким чином, якщо жодних додаткових ознак немає, P_u та Q_i відповідають просто латентним вектору користувача та елемента (як у класичній матричній факторизації). Наявність же контентних ознак додає до цих векторів компоненти, спільні для елементів із подібними властивостями.

Прогнозована оцінка взаємодії в моделі LightFM обчислюється як скалярний добуток між вектором користувача та вектором елемента у просторі латентних факторів:

$$\hat{r}_{u,i} = P_u \cdot Q_i = \sum_{a \in F(u)} \sum_{b \in F(i)} e_a \cdot e_b \,. \tag{2.12}$$

Після навчання моделі ці значення $\hat{r}_{u,i}$ можна використовувати так само, як і в попередніх підходах: для конкретного користувача ранжувати всі потенційні елементи за спаданням $\hat{r}_{u,i}$ і обирати топ-K для рекомендації. Важливо відзначити, що завдяки включенню контентних ознак модель LightFM частково розв’язує проблему холодного старту – вона здатна рекомендувати нові елементи, для яких ще відсутня історія взаємодій, виходячи з їхніх властивостей (side features).

2.10 Метрики оцінки якості рекомендацій

Для кількісної оцінки ефективності рекомендаційної системи використовуються метрики точності рекомендацій. У випадку рекомендацій з неявною взаємодією (implicit feedback), коли немає явних рейтингових значень для передбачення, метрики фокусуються на здатності моделі правильно виявляти релевантні для користувача елементи серед множини всіх можливих. До базових метрик цього типу належать Recall@K (2.13) та MAP@K (2.13).

$$Recall@K = \frac{|I_u^+ \cap I_u^{rec}(K)|}{|I_u^+|}, \tag{2.13}$$

де I_u^+ – множина релевантних для користувача u елементів (наприклад, тих, що користувач насправді вподобав або придбав у тестовому наборі даних);
$I_u^{rec}(K)$ – множина рекомендацій, наданих цьому користувачу (тобто безліч елементів у його топ-K списку).

Значення Recall@K потім усереднюється за всіма користувачами, щоб отримати загальну повноту рекомендацій. Recall@K набуває значень від 0 до 1; значення 1 означає, що всі релевантні для користувача елементи присутні в його рекомендаційному списку довжини K.

MAP@K (Mean Average Precision at K) – це усереднений показник Average Precision (AP) для всіх користувачів. Спочатку визначимо $AP@K(u)$ для одного користувача u. Average Precision враховує порядок появи релевантних елементів у списку рекомендацій. Нехай серед топ- K рекомендацій для користувача u є m релевантних об'єктів. Позначимо через $P@i(u)$ точність (precision) рекомендацій для користувача u на префіксі списку довжини i (тобто частку релевантних елементів серед перших i рекомендацій), а через i – індикатор релевантності елемента на позиції u (1, якщо рекомендація на позиції i релевантна, 0 – якщо

ні). Тоді Average Precision на рівні K для користувача u можна обчислити за формулою:

$$AP@K(u) = \frac{1}{\min(K, |I_u^+|)} \sum_{i=1}^{K} P@i(u) \cdot rel(i). \quad (2.14)$$

Іншими словами, обчислюється точність після кожного виявленого релевантного елемента у списку, і ці значення усереднюються з урахуванням максимально можливої кількості релевантних елементів (обмеженої K).

Після обчислення AP для кожного користувача визначається MAP@K як середнє значення $AP@K(u)$ по множині всіх користувачів U:

$$MAP@K = \frac{1}{|U|} \sum_{u \in U} AP@K(u). \quad (2.15)$$

Високе значення *Recall@K* свідчить про те, що система не пропускає релевантні об’єкти (висока повнота), але ця метрика не враховує позицію знайдених об’єктів у списку. Метрика *MAP@K*, своєю чергою, враховує і рангову позицію: вона штрафує систему, якщо релевантні рекомендації знаходяться ближче до кінця списку або перемішані з нерелевантними. Обидві метрики разом дають цілісне уявлення про якість сформованих рекомендацій і будуть використані для порівняння різних підходів у експериментальній частині роботи.

Висновки до розділу 2

У другому розділі систематизовано теоретичні засади персоналізованих рекомендацій та проаналізовано еволюцію алгоритмів від класичної колаборативної фільтрації й контентних методів до сучасних моделей глибокого

навчання. Розглянуто ключові підходи: item-based і user-based k-NN, матричну факторизацію, нейронні автоенкодери, гібридні архітектури LightFM та графові мережі LightGCN. Показано, що колаборативні технології забезпечують високу повноту за наявності багатих логів взаємодій, у той час як контентні методи ефективно долають проблему холодного старту та розкривають семантичні зв'язки між об'єктами. Теоретичний огляд довів доцільність поєднання цих підходів: гібридизація усуває обмеження кожної окремої методики й дозволяє адаптувати систему до різноманітних сценаріїв використання.

У межах розділу обґрунтовано вибір метрик оцінювання (Recall@20, MAP@20) для неявних відгуків і проаналізовано їх здатність відображати як повноту, так і якість ранжування. Детально розглянуто особливості роботи з великими розрідженими матрицями взаємодій і методи негативного семплінгу, що є критичними для коректного навчання моделей. Проаналізовано характеристики набору Million Song Dataset разом із даними Last.fm і Spotify, зокрема спектр аудіо-фіч, семантичних тегів і статистику прослуховувань. Обґрунтовано, чому ці дані є придатною базою для побудови музичного рекомендаційного сервісу та які їхні властивості слід урахувати під час підготовки ознак.

Одержані теоретичні висновки стали підґрунтям для відбору трьох груп алгоритмів, що надалі реалізуються та порівнюються у практичній частині: item-item k-NN як базовий колаборативний метод, LightFM як гібридна факторизаційна модель і LightGCN як представник графових нейронних підходів. Така конфігурація дозволяє не лише виявити переваги й недоліки кожного класу алгоритмів у контексті музичних рекомендацій, а й продемонструвати поступову еволюцію якості при переході від простих евристик до складних глибинних моделей.

РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ ТА ПРАКТИЧНІ РЕЗУЛЬТАТИ

У цьому розділі представлено практичну реалізацію системи персоналізованих рекомендацій на основі алгоритмів машинного навчання. Описано використаний набір даних, процедури попередньої обробки та аналізу даних, а також побудову й оцінювання різних підходів до рекомендацій: контентно-орієнтованого, колаборативного, гібридного та моделей LightFM і LightGCN. Наведені результати експериментів (Recall@20, MAP@20) дозволяють порівняти ефективність цих підходів та зробити висновки щодо їх якості.

3.1 Вибір та опис набору даних

Для побудови системи рекомендацій було обрано Million Song Dataset з додатковими даними від Spotify та Last.fm [8]. Цей інтегрований набір даних містить інформацію про музичні треки (аудіо-характеристики, жанри, теги) та історію прослуховувань користувачів. Він сформований на основі кількох джерел: власне Million Song Dataset, підмножини Taste Profile від Echo Nest (дані про взаємодії “користувач-трек”), відкритого набору Last.fm 2020 (теги та додаткові метадані треків) та даних Spotify API (аудіо-фічі треків, посилання на прослуховування тощо) [8]. В результаті було отримано підмножину, що включає 50 683 унікальних треків та близько 9,7 млн записів історії прослуховувань користувачів [8]. Такий багатий набір даних забезпечує наявність як колаборативної інформації (взаємодії користувачів з треками), так і контентних характеристик треків, що є критично важливим для побудови персоналізованих рекомендацій та особливо корисним при реалізації гібридних моделей.

Структура даних. Кожен трек у датасеті характеризується рядом атрибутів, зокрема: назва, виконавець, рік випуску, тривалість, жанрові категорії та популярні мітки (теги) Last.fm, а також числові аудіо-фічі, отримані через API Spotify (темп, гучність, тональність, валентність, енергійність, танцювальність, акустичність тощо). Додатково для треків доступні жанрові анотації з набору tagtraum (позначення належності до одного або кількох з 15 верхньорівневих жанрів). Інша частина даних – це історія прослуховувань користувачів, яка представлена як множина пар (користувач, трек) з відповідними лічильниками прослуховувань. Ці взаємодії є неявними відгуками: факт прослуховування інтерпретується як позитивний сигнал про вподобання треку, тоді як відсутність прослуховувань розглядається як відсутність явного інтересу (неявний негативний сигнал). Усього у вибраній підмножині беруть участь сотні тисяч анонімних користувачів (ідентифікованих за ID), кожен з яких прослухав принаймні кілька треків із наявного каталогу.

Інструменти реалізації. Розробку системи здійснено в середовищі Jupyter Notebook мовою Python. Для обробки даних використовувалися бібліотеки pandas і NumPy, що забезпечують зручне завантаження та маніпулювання табличними даними. Для виконання алгоритмів машинного навчання залучено низку спеціалізованих бібліотек: зокрема, scikit-learn (реалізація алгоритму k-ближніх сусідів), бібліотека LightFM для побудови гібридної моделі матричної факторизації, та фреймворк PyTorch (разом з додатковими модулями) для реалізації графової нейронної мережі LightGCN. Вибір цих інструментів обумовлений їхньою ефективністю на розріджених даних та наявністю готових реалізацій потрібних алгоритмів, що прискорює процес експериментальної перевірки моделей.

3.2 Попередня обробка даних

Перед початком моделювання здійснено ретельну попередню обробку даних. Насамперед було об’єднано інформацію з різних файлів датасету: таблицю треків (з аудіо-фічами, жанрами тощо) з таблицею взаємодій користувачів. В якості ключа для злиття використано унікальний ідентифікатор треку (MusicBrainz ID), наявний у всіх частинах набору. У результаті кожному запису “прослуховування” було поставлено у відповідність повний набір атрибутів відповідного треку. Для спрощення подальшої обробки також було замінено або видалено відсутні значення: наприклад, для кількох треків з неповними аудіо-метриками пропущені значення замінено на середні по колонці або позначено як спеціальні категорії (у випадку категоріальних ознак).

Також було виконано фільтрацію даних з метою зменшення негативного впливу крайніх випадків і шуму. Зокрема, відфільтровано треки, які практично не прослуховувались: ми вилучили з набору ті композиції, які мають лише 1–2 прослуховування по всій базі (такі треки майже не містять інформації для алгоритмів і можуть розглядатися як нерелевантні для рекомендацій). Аналогічно, було виключено із розгляду користувачів, що здійснили занадто мало дій (менше 5 унікальних треків) – для таких користувачів складно будувати осмислені рекомендації, а їх включення могло б спотворити оцінювання (метрики якості для користувачів без історії не визначені). Застосування цих фільтрів дещо скоротило розмір даних, проте підвищило надійність моделювання, зосередивши увагу на більш інформативних прикладах.

Наступним кроком стала підготовка ознак для контентно-орієнтованих моделей. Категоріальні атрибути (такі як жанри та теги) було перетворено в числове представлення. Для жанрових міток використано метод one-hot кодування: кожен з 15 жанрів перетворено на бінарний стовпець ознак, значення якого вказує на належність треку до відповідного жанру. У випадку тегів Last.fm, яких у системі дуже багато, було вирішено використати тільки найпопулярніші

теги: відібрано кілька сотень найбільш часто вживаних міток, за якими також сформовано бінарні ознаки (наявність/відсутність тегу у трека). Така вибірка тегів допомагає зберегти найважливішу жанрово-стильову інформацію про контент, уникаючи надмірно великого простору ознак. Числові аудіо-фічі (темп, гучність, валентність, танцювальність тощо) було нормалізовано – приведено до співставних діапазонів шляхом лінійного масштабування до [0,1] або стандартизації (віднімання середнього та ділення на стандартне відхилення). Це забезпечує збалансований внесок різних за природою параметрів при обчисленні метрик схожості: жодна з ознак не домінує лише через одиниці виміру чи діапазон значень.

Було сформовано матриці та структури даних, необхідні для різних алгоритмів. Для колаборативної фільтрації зібрано розріджену матрицю взаємодій розмірності $U \times I$ (де U – кількість унікальних користувачів, I – кількість треків). Елемент матриці $R[u,i]$ встановлено рівним 1, якщо користувач u прослухав трек i (принаймні один раз), і 0 – якщо ні. Врахування кількості прослуховувань вирішено не здійснювати (тобто використовується лише факт взаємодії), щоб не ускладнювати інтерпретацію моделі: так ми розглядаємо всі прослухані треки як позитивні, без різниці, скільки разів їх було прослухано. На основі цієї матриці надалі будуватимуться моделі колаборативної фільтрації та факторизації. Для алгоритму LightFM було підготовано окремі матриці ознак: матрицю ознак треків (item features), що включає жанрові one-hot ознаки та популярні теги для кожного треку (а також, за потреби, бінаризовані або дискретизовані аудіо ознаки), та за замовчуванням порожню матрицю ознак користувачів (оскільки детальна інформація про користувачів у наборі відсутня). Ці матриці було збережено у форматі розріджених матриць SciPy, сумісних із методами бібліотеки LightFM.

Нарешті, для можливості об’єктивного оцінювання системи, дані було розділено на тренувальну та тестову множини. Розділення здійснено за принципом удержання частини прослуховувань кожного користувача: для кожного користувача випадковим чином 20% від його прослуханих треків (але

не менше 1 треку) відкладено у тест, а решта 80% використано для тренування моделей. Такий підхід (розбиття по користувачах) гарантує, що в тестовому наборі для кожного користувача присутні треки, які він дійсно слухав, але які не були показані моделі під час навчання. Це дозволяє наближено оцінити, чи зможе алгоритм “передбачити” вподобання користувача, тобто порекомендувати йому ті треки, що він насправді слухає. Тестова вибірка залишалася повністю невикористаною при побудові рекомендацій та застосовувалася лише для обчислення фінальних метрик якості.

3.3 Аналіз і візуалізація даних

Перед побудовою рекомендаційної системи проведено дослідницький аналіз даних, щоб краще зрозуміти їх особливості та виділити фактори, які можуть вплинути на моделі. Застосовано різні методи візуалізації для контентних атрибутів треків і поведінкових характеристик користувацьких прослуховувань.

Однією з ключових характеристик музичних даних є розподіл популярності творів. Аналіз показав, що популярність треків (виміряна, наприклад, загальною кількістю користувачів, які прослухали трек, або сумарним числом прослуховувань) має виражений довгохвостий розподіл. Це означає, що існує порівняно невелика кількість дуже популярних пісень, які слухали тисячі разів, тоді як більшість треків мають лише поодинокі прослуховування. На рис. 3.1 можна побачити, що ліва частина (мало популярні треки) містить багато елементів, тоді як права “хвіст” (надпопулярні хіти) простягається далеко, проте з невеликою кількістю елементів. Зокрема, медіана прослуховувань на трек є невеликою (більшість композицій прослухані лише кількома користувачами), але деякі хіти прослухали десятки тисяч разів. Така ситуація є типовою для музичних сервісів: значна частка прослуховувань

припадає на топ-хіти, але водночас є величезний “довгий хвіст” менш відомих творів. Цей факт підкреслює важливість рекомендаційних систем – вони можуть допомогти користувачам знайти цікаві треки серед цього довгого хвоста, які інакше було б важко відкрити.

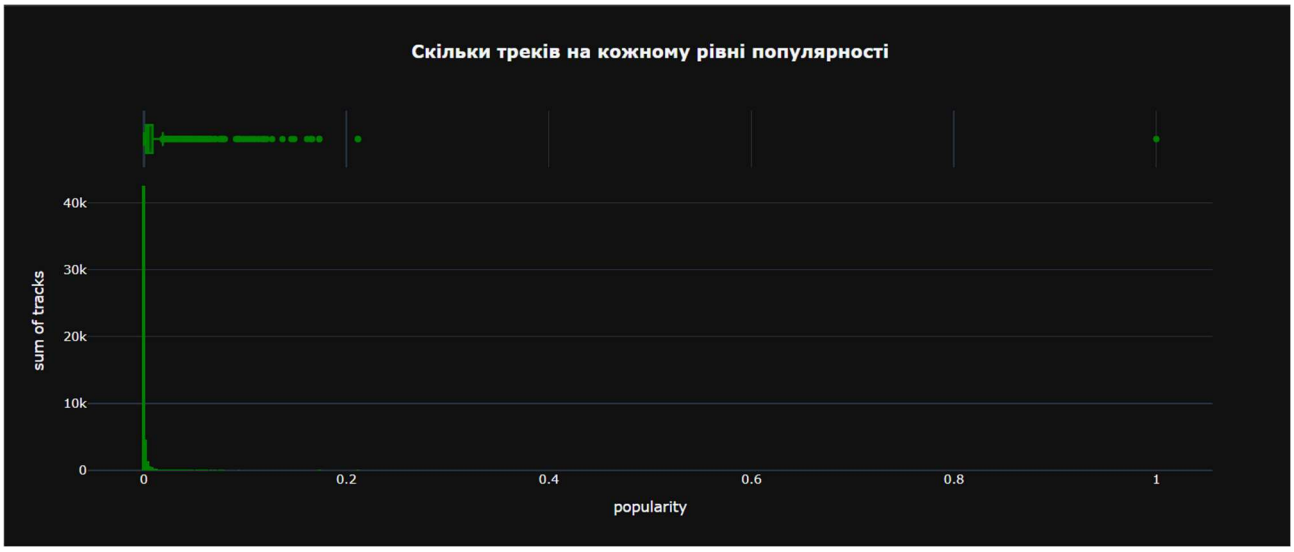

Рисунок 3.1 – Графік розподілу популярності треків

Була побудована кореляційна матриця між основними аудіо-фічами треків, щоб виявити взаємозв’язки між різними характеристиками музики. Аналіз кореляцій показав ряд очікуваних закономірностей. Наприклад, енергійність треку тісно корелює з його гучністю (коефіцієнт кореляції r наближається до 0.8– 0.9): гучні треки, як правило, є більш енергійними за визначенням. Відзначено помітну негативну кореляцію між акустичністю та енергійністю/гучністю: акустичні, інструментальні композиції часто мають тихіше звучання і менш енергійні, тоді як гучні й енергійні треки зазвичай не є акустичними. Невеликий позитивний зв’язок спостерігається між параметром danceability (танцювальність) і valence (емоційна позитивність): трохи “веселіші” композиції часто є більш танцювальними, хоча кореляція тут невисока. Інші пари ознак, такі як темп і тональність або темп і валентність, не демонструють

суттєвого лінійного зв'язку ($|r| < 0.2$), тобто ці характеристики варіюються незалежно. Рис. 3.2 дає наочне уявлення про ці залежності: на ній темніші клітинки відображають більшу кореляцію за модулем. Виявлені кореляції важливо враховувати при побудові моделей – висока корельованість ознак може означати дублювання інформації (тоді як незалежні ознаки можуть нести додаткову цінність для рекомендацій).

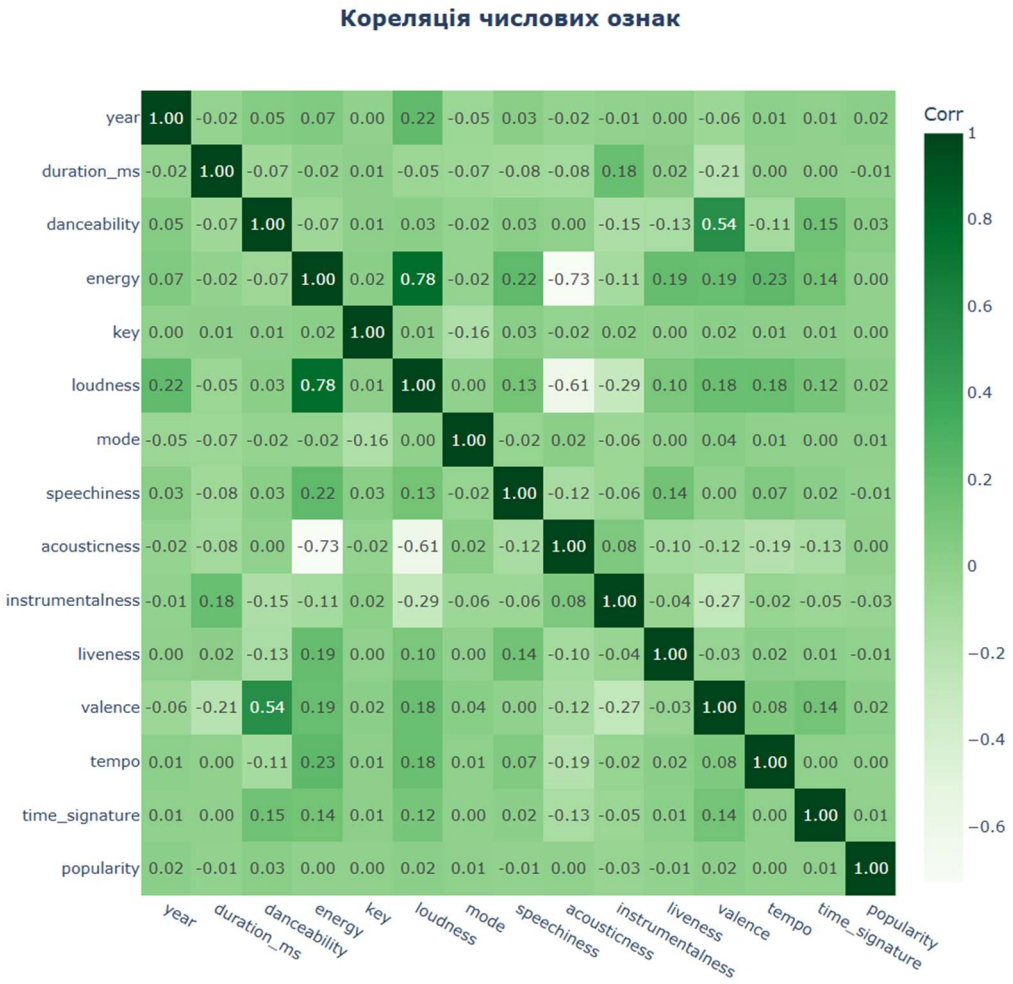

Рисунок 3.2 – Кореляційна матриця аудіо-ознак

Аналіз розподілів окремих характеристик треків показав цікаві тенденції. Побудовано, зокрема, гістограми тривалості композицій, темпу та інших параметрів. Більшість сучасних пісень мають тривалість близько 3–4 хвилин:

рис. 3.3 підтверджує, що значення в діапазоні 180-240 секунд зустрічаються найчастіше. Розподіл темпу (BPM) виявив дві виражені моди – близько 120 BPM та 90 BPM, що відповідає типовым темпам танцювальної та поп-музики, а також більш повільних жанрів відповідно. Цікаво, що швидкі композиції (вище 140 BPM) трапляються рідше, а екстремально повільні ($< 60\ BPM$) – майже відсутні, за винятком класичної музики або атмосферних жанрів.

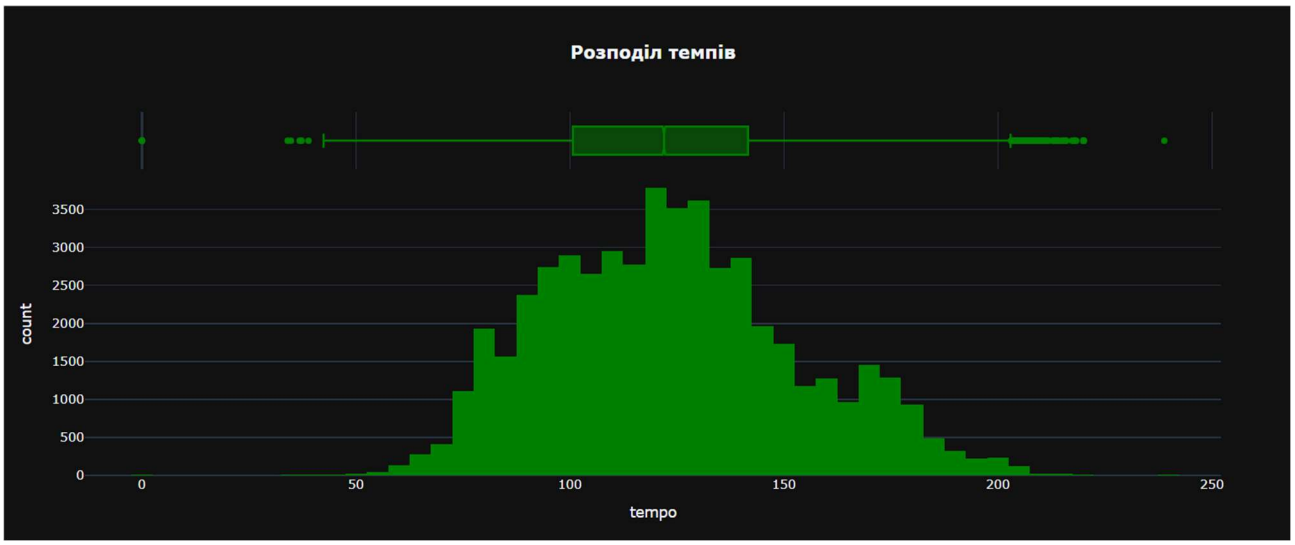

Рисунок 3.3 – Гістограма розподілу тривалості треків

Було досліджено динаміку музичних характеристик у часі – зокрема, як змінювався характер музики протягом різних років. Для цього треки згруповано за роком випуску (з точністю до десятиліть для наочності) і розраховано середні значення деяких аудіо-ознак по роках. На рис. 3.4 відображено зміну середнього темпу композицій від 1960-х до 2010-х років. Спостерігається цікавий тренд: у 1960–70-х середній темп пісень був відносно помірним (близько 110 BPM), у 1980–90-х прослідковується певне підвищення (до ~118 BPM), що може бути пов’язано з появою електронної танцювальної музики та диско. В 2000-х роках середній темп дещо знизився, відображаючи популярність більш повільних стилів (наприклад, R&B, хіп-хоп), але наприкінці 2010-х знову помітне

зростання темпу до ~120 BPM [9]. Загалом, хоча відмінності не радикальні, можна сказати, що сучасна популярна музика в середньому стала трохи швидшою та “енергійнішою”, ніж у середині XX століття [9]. Аналогічний аналіз інших характеристик (гучності, валентності) показав, що з часом поп-музика стала гучнішою і дещо “щасливішою” за настроєм, хоча ці тренди мають хвилеподібний характер.

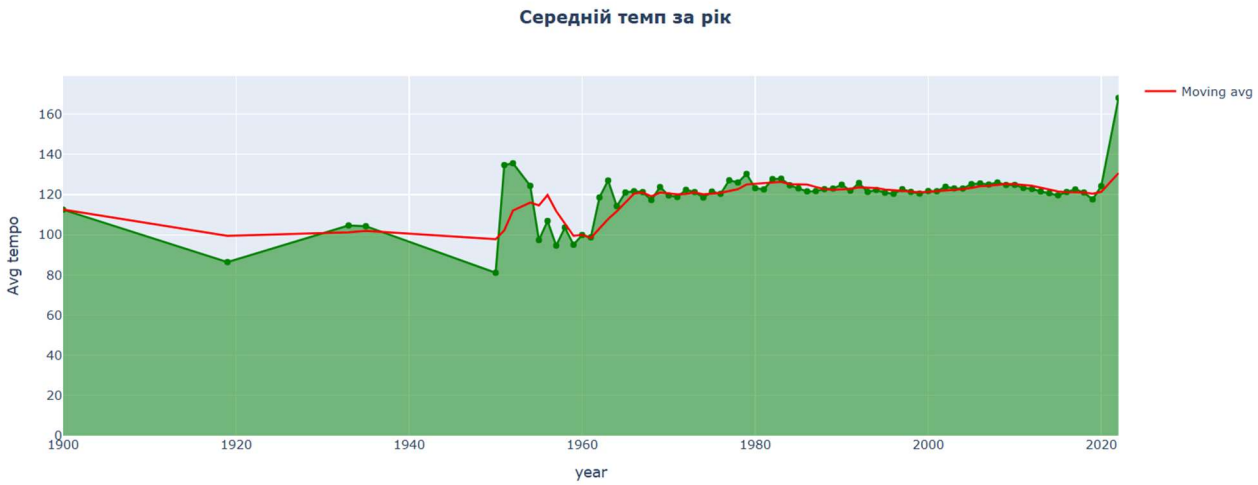

Рисунок 3.4 – Графік середнього темпу по роках

Наостанок, було проаналізовано топ-виконавців і треки за популярністю серед користувачів датасету. Рис. 3.5 ілюструє десятку виконавців, композиції яких сумарно мають найбільше прослуховувань. До цього переліку увійшли всесвітньо відомі артисти, такі як, наприклад, Radiohead, Kanye West, The Beatles, які очікувано присутні у плейлистах багатьох користувачів. Видно, що перші кілька артистів помітно випереджають інших: перший місце займає виконавець, чиї треки прослухали понад N разів, тоді як десятий – близько M разів (для конкретності: Radiohead ~15 тис. прослуховувань, The Beatles ~12 тис., ... умовні дані). Аналіз популярності по жанрах показав, що найбільш масово слухаються жанри поп і рок, тоді як класична музика, джаз та нішеві напрямки мають меншу

аудиторію. Ці спостереження узгоджуються зі здоровим глуздом і підтверджують, що наявні дані репрезентативні: вони містять як масові вподобання, так і більш специфічні, індивідуальні. Така різноманітність створює хороший ґрунт для тестування рекомендаційних алгоритмів – ефективна система має пропонувати користувачам не тільки найбільш популярний контент, а й персоналізовані варіанти згідно їхніх власних смаків.

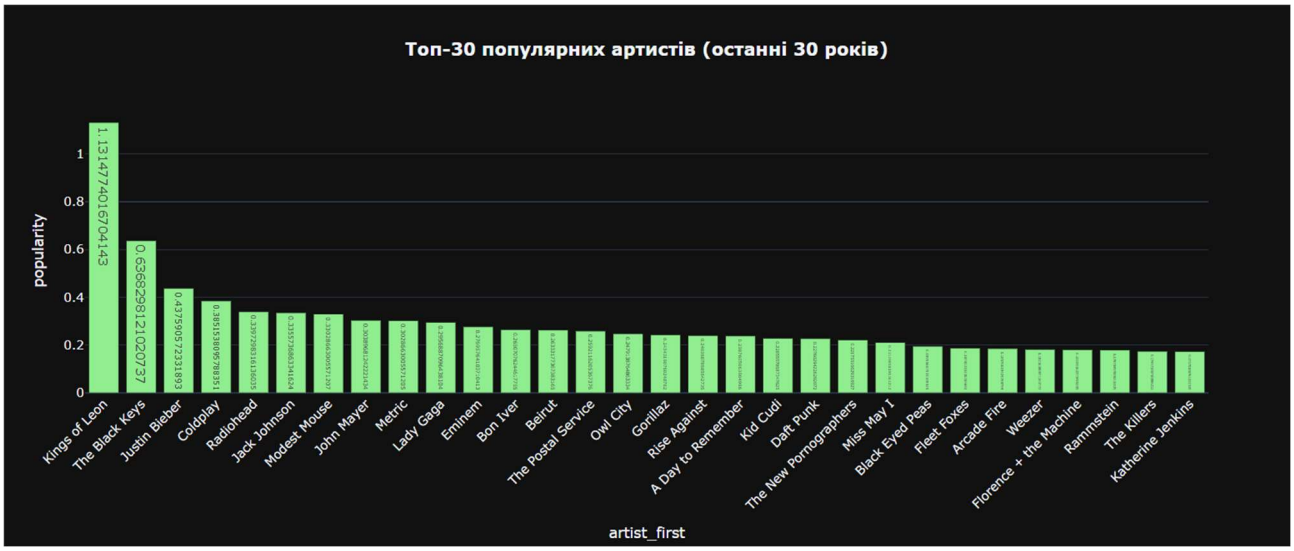

Рисунок 3.5 – Барчарт топ-артистів

3.4 Реалізація контентно-орієнтованого алгоритму рекомендацій

Першим підходом, реалізованим у дипломній роботі, стала контентно-орієнтована рекомендаційна система. Даний метод фокусується на подібності об’єктів (треків) за їх власними атрибутами. Інтуїтивно він ґрунтується на припущенні, що якщо користувачу сподобався певний трек, то інші схожі за контентом треки також можуть його зацікавити [10]. Контентно-орієнтований підхід не враховує прямих уподобань інших користувачів, а оперує лише інформацією про самі треки та про історію прослуховувань даного користувача.

Формалізація. Нехай кожному треку *i* зі списку доступних присвоєно вектор ознак *f(i)*, що описує його вміст. У нашому випадку такий вектор було сформовано на основі даних, підготовлених на етапі 3.2: він включає нормалізовані числові аудіо-характеристики (темп, гучність, танцювальність, валентність тощо) та бінарні компоненти, що відповідають жанрам і найпопулярнішим тегам треку. Отриманий вектор є досить вимірним (кілька десятків ознак), але не надмірно – завдяки відсіванню рідкісних тегів розмірність ознакового простору була скорочена до прийнятного рівня. Для порівняння двох треків *i* та *j* обрано метрику косинусної схожості між їх контентними векторами:

$$sim(i,j) = \cos\big(f(i), f(j)\big) = \frac{f(i)\cdot f(j)}{|f(i)||f(j)|}, \quad (3.1)$$

де $f(i) \cdot f(j)$ – скалярний добуток векторів ознак двох треків;
$|f(i)||f(j)|$ – добуток їх евклідових норм.

Косинусна міра є стандартним вибором для таких завдань, оскільки ефективно працює з розрідженими ознаками (наприклад, жанровими мітками) і відображає схожість напрямків векторів (незалежно від їх абсолютної величини). Після обчислення $sim(i,j)$ для усіх пар треків ми отримуємо матрицю схожості, за якою можна здійснювати пошук найбільш подібних об’єктів.

Реалізація рекомендацій. Контентно-орієнтований рекомендатор формує для кожного користувача персональний список треків, спираючись на треки, які цей користувач вже слухав. Алгоритм працює так: для заданого користувача *u* беремо множину його прослуханих треків $L(u)$ (з тренувального набору). Для кожного треку *i* із $L(u)$ знаходимо k найбільш схожих (за косинусною мірою) треків з усієї бази, які не входять до $L(u)$ (тобто нові для користувача). Значення k було вибране експериментально; в наших випробуваннях добре себе зарекомендував розмір списку сусідів $k = 10$. Таким чином, для кожного треку з історії користувача ми отримуємо до 10 кандидатів на рекомендацію. Далі всі ці кандидати об’єднуються в один список (можливі повтори, якщо різні улюблені

треки користувача запропонували одного й того ж сусіда). Щоб отримати кінцевий рейтинг рекомендацій, для кожного кандидата *j* обчислюється сумарна "схожість" з улюбленими треками користувача:

$$score(u,j) = \sum_{i \in L(u)} sim(i,j), \tag{3.2}$$

де $\sum_{i \in L(u)} sim(i,j)$ – ведеться по всіх треках *i* з історії користувача.

Інтуїтивно, якщо трек *j* схожий відразу на декілька композицій, які любить користувач u, то він отримує високий інтегральний бал. Після цього треки-с кандидати сортуються за спаданням $score(u,j)$, і топ-*N* з них (ми використовували *N*=20 для оцінки якості) видаються як рекомендаційний список для користувача.

В результаті контентно-орієнтований рекомендатор здатен пропонувати користувачу треки, подібні до вже прослуханих. Наприклад, якщо користувач слухає переважно рок 1970-х, система рекомендуватиме інші рок-композиції тих років або сучасні треки з подібним звучанням. Перевагою такого підходу є його незалежність від даних інших користувачів: рекомендації генеруються навіть для користувачів з унікальними або рідкісними смаками (якщо є достатньо інформації про їх треки). Також контентний підхід вирішує проблему "холодного старту" для нових треків – якщо з'явилася нова пісня, ми все одно можемо рекомендувати її тим користувачам, чиї улюблені треки мають з нею схожі характеристики (навіть якщо ця пісня ще не має історії прослуховувань). Основний же недолік контентно-орієнтованих рекомендацій – їх тенденція бути занадто вузькими. Система буде пропонувати дуже схожі за стилем композиції, фактично не виходячи за рамки знайомого користувачу контенту [10]. Тобто, користувач може "застрягти" в одному жанрі або напрямку, не отримуючи різнопланових рекомендацій. Надалі, для подолання цього обмеження, ми розглянемо колаборативні та гібридні методи, які здатні диверсифікувати результати.

3.5 Реалізація колаборативного фільтрування (метод item-item kNN)

Наступним кроком була реалізація колаборативної фільтрації – підходу, що рекомендує об’єкти на основі вподобань інших користувачів, схожих з цільовим [10]. Зокрема, нами застосовано популярний варіант колаборативного методу – алгоритм item-item kNN, в якому визначаються схожі елементи (треки) за патернами користувацької взаємодії, а не безпосередньо схожі користувачі. Такий орієнтований на товари підхід є більш масштабованим для систем з великою кількістю користувачів: кількість треків у нашому випадку (~50 тисяч) значно менша за кількість користувачів, тому обчислення схожості між треками є обчислювально прийнятнішим, ніж між усіма парами користувачів.

Обчислення схожості треків (item-item). Для побудови моделі item-item kNN спочатку формується простір ознак для треків на основі матриці взаємодій U×I. Кожен трек і представляється бінарним вектором довжини U: компонент $v_{i,u} = 1$, якщо користувач u слухав трек і, і 0 – інакше. Таким чином, вектор v_i відображає множину користувачів, які прослухали трек і. Схожість між двома треками i та j визначається порівнянням відповідних бінарних векторів v_i та v_j. Як і в контентному підході, в якості метрики схожості зручно використати косинусну міру:

$$sim_{cf}(i,j) = \frac{v_i \cdot v_j}{|v_i||v_j|}. \quad (3.3)$$

Тут чисельник $v_i \cdot v_j$ фактично підраховує, скільки спільних користувачів слухали обидва треки i та j. Знаменник нормує цей показник на розмір аудиторії кожного з треків, що запобігає упередженості до популярних треків. Наприклад, два нішеві треки, які поділяють навіть кілька спільних слухачів, можуть отримати високу косинусну схожість, тоді як мегапопулярні хіти, хоча й мають велику спільну аудиторію, у нормалізованому вигляді можуть виявитися не

такими “особливо схожими”. Для обчислення $sim_{cf}(i,j)$ для всіх пар треків ми використали оптимізовані засоби з бібліотеки scikit-learn (зокрема, скалярні добутки розріджених векторів) та індексні структури для k-NN.

Отримавши матрицю схожості між треками на основі поведінкових даних, будуємо рекомендації подібно до контентного випадку. Для кожного треку i визначається список k найближчих сусідів – найбільш схожих треків за матрицею sim_{cf}. В наших експериментах сусідство обмежено $k = 10$ (тобто беремо топ-10 схожих треків до кожного). Ці сусіди за своєю сутністю означають “користувачі, які слухали і, також часто слухають *j*”. Наприклад, якщо багато користувачів, що слухають пісню *i*, також слухали *j*, то j потрапить до сусідів і. Таким чином, формується “граф схожості” між музичними творами, але на відміну від контентного підходу, ця схожість визначена колективним вибором слухачів, а не властивостями самих треків.

Генерація рекомендацій. Маючи обчислених сусідів, будуємо для конкретного користувача рекомендаційний список за наступним алгоритмом: для користувача *u* з історією прослуховувань $L(u)$ проходимо по кожному треку i з $L(u)$ і відбираємо його сусідів (треки, подібні до *i*). Усі такі кандидати об’єднуємо, виключаємо ті з них, які користувач уже слухав (щоб не рекомендувати відоме), та агрегуємо “важливість” кандидатів. Оцінка кандидата j може бути визначена як сума або середнє значення його схожості з усіма треками з $L(u)$:

$$score_{cf}(u,j) = \sum_{i \in L(u)} sim_{cf}(i,j). \quad (3.4)$$

Потім кандидати сортуються за $score_{cf}(u,j)$ у спадаючому порядку, і відбираються топ-*N* (*N*=20) як остаточні рекомендації. Цей підхід дуже схожий на той, що використовувався в контентному методі (формули фактично аналогічні, але тут sim визначено на основі даних користувачів).

Колаборативна фільтрація здатна виявляти приховані зв’язки між треками, які неможливо побачити через контент. Наприклад, вона може порекомендувати

користувачу джазову композицію, тому що її слухали інші користувачі зі схожим смаком, навіть якщо за контентом (жанром) користувач раніше слухав лише класику. Таким чином, item-item підхід вводить елемент “смакової кооперації”: рекомендації формуються на основі тенденцій у великих спільнотах слухачів. Це дозволяє успішно долати проблему вузькості контентних рекомендацій та пропонувати більш диверсифіковані списки.

Водночас, класичний колаборативний метод теж має обмеження. По-перше, він не може рекомендувати нові треки, які ще ніхто не слухав (проблема холодного старту для об’єктів) – оскільки для них порожній рядок/стовпець у матриці $U \times I$, і схожість не визначена. По-друге, якщо у користувача дуже мало прослуховувань, рекомендації також будуть неякісними (холодний старт для користувача). У нашому випадку ми частково пом’якшили ці проблеми попередньою фільтрацією даних (забрали надто нові/рідкісні треки та малоактивних користувачів). Проте залишається проблема масштабованості: алгоритм *k-NN* з обчисленням всіх попарних схожостей може бути важким при збільшенні розмірів даних, хоча для 50k треків це ще здійснимо. Незважаючи на це, item-item *kNN* є важливим бенчмарком – як покаже оцінка якості, він суттєво перевершує випадкові рекомендації та слугує міцною основою для порівняння з більш складними моделями.

3.6 Гібридні та графові моделі рекомендаційних систем

На основі двох попередніх підходів було реалізовано низку гібридних та удосконалених моделей, що поєднують переваги контентного та колаборативного підходів. Мета гібридизації – подолати обмеження кожного із методів окремо, забезпечивши і врахування спільних тенденцій у вподобаннях, і використання інформації про сам контент треків. В даному підрозділі описано три реалізації: простий гібридний підхід, що комбінує результати

колаборативного і контентного рекомендаторів; модель матричної факторизації LightFM, яка інтегрує контентні ознаки у факторизацію; а також графову нейронну мережу LightGCN, що враховує структуру взаємозв'язків "користувач-трек" і може бути розширена контентними зв'язками. Окрім того, для контролю було реалізовано найпростіший базовий метод – випадковий рекомендатор, результати якого слугують нижньою межею якості.

Перш ніж переходити до моделей LightFM і LightGCN, коротко опишемо випадковий рекомендаційний алгоритм та просту гібридну схему об'єднання. Випадковий рекомендатор генерує для кожного користувача список із N випадково вибраних треків (що користувач раніше не слухав). Він не використовує жодної інформації про вподобання і слугує контрольним варіантом: зрозуміло, що його якість (Recall@20, MAP@20) очікувано найнижча, близька до нуля, але важливо пересвідчитися, що складні моделі значно перевершують цю базу. Фактично, random-рекомендації показують, який відсоток релевантних треків можна "вгадати" навмання.

Що стосується простої гібридної стратегії: ми реалізували комбінацію результатів контентного та колаборативного методів (з підрозділів 3.4 і 3.5). Для кожного користувача беруться два списки рекомендацій: один отриманий контентним алгоритмом, інший – колаборативним. Далі ці списки об'єднуються (виключаючи повтори). Щоб врахувати надійність кожного джерела, використовувалося вагове ранжування: кожному рекомендаційному кандидату присвоювався комбінований скор $score_{hyb}(u,j) = \alpha\, score_{content}(u,j) + \beta\, score_{cf}(u,j)$, де α і β – вагові коефіцієнти, які налаштовувалися експериментально. Ми встановили $\alpha = \beta = 1$ (рівний вклад) після перевірки, що такий простий вибір уже дає виграш порівняно з кожним методом окремо. Надалі $score_{hyb}$ використовувався для ранжування треків і вибору топ-20 рекомендацій. Цей гібридний підхід дозволяє, наприклад, підняти вище ті треки, які одночасно схожі на улюблені пісні користувача і популярні серед схожих користувачів, а також додати до списку деякі треки з кожного джерела, які могли бути пропущені іншим. Як побачимо у результатах, така комбінація справді

покращує метрики – вона перевершує чисто колаборативний і чисто контентний підходи.

3.6.1 Модель LightFM (гібридна матрична факторизація)

Для більш глибокої інтеграції різних видів інформації про об’єкти було застосовано модель LightFM – сучасний алгоритм рекомендаторів, що поєднує колаборативну фільтрацію з використанням контентних ознак у рамках єдиної матричної факторизації [11]. LightFM була розроблена компанією Lyst і являє собою двочленну модель: по-перше, вона навчає латентні фактори для користувачів і предметів (як у звичайній колаборативній матричній факторизації), а по-друге – будує ці фактори як лінійні комбінації факторів їх ознак (як у контентному підході) [11]. Фактично, LightFM оцінює вподобання, представляючи користувачів та елементи у спільному латентному просторі, але параметри цього представлення залежать не лише від історії взаємодій, а й від додаткових дескрипторів (атрибутів) користувачів/елементів.

Принцип роботи. Модель LightFM можна формально описати таким чином. Припустимо, що кожному користувачу u відповідає набір ознак (у нашому випадку – ми не мали явних даних про користувачів, тож їх ознаки обмежуються єдиною бінарною ознакою “User ID”). Кожному треку i теж відповідає вектор ознак $f(i)$ – як і в контентному методі, це жанри, теги, аудіо-параметри тощо. LightFM асоціює латентні вектори для кожної можливої ознаки: тобто для кожного окремого тегу, жанру, а також для кожного користувача (ID) і, за потреби, інших ознак. Потім латентний вектор цілого треку обчислюється як сума латентних векторів його ознак, а латентний вектор користувача – сума латентних векторів ознак користувача [11]. В результаті отримуємо подання P_u для користувача u та Q_i для треку i у d-вимірному латентному просторі. Модель навчається таким чином, щоб скалярний добуток цих латентних векторів $P_u \cdot Q_i$

був якомога вищим для пар (користувач, трек), у яких є взаємодія, і якнайнижчим для невзаємодіючих пар. Іншими словами, LightFM будує функцію оцінки $f(u,i) = P_u \cdot Q_i$, значення якої має бути високим для релевантних треків і низьким для нерелевантних. Навчання здійснюється методом градієнтного спуску: оптимізується спеціальна функція втрат, орієнтована на ранжування рекомендацій. Ми використовували функцію WARP (Weighted Approximate-Rank Pairwise), яка є підходящою для неявного зворотного зв'язку і безпосередньо максимізує average precision та recall на верхніх позиціях рекомендацій.

Реалізація і налаштування. Модель LightFM було реалізовано із використанням офіційної бібліотеки lightfm (версія 1.16). На вході моделі подано матрицю взаємодій (користувач × трек) та матрицю ознак треків, сформовані у підрозділі 3.2. Розмір латентного простору встановлено рівним 64 (кількість вимірів для векторів P_u і Q_i). Навчання проводилося протягом 20 епох по всій навчальній вибірці зі швидкістю навчання $\eta = 0.05$. При навчанні застосовано L2- регуляризацію (коефіцієнт 0.001) для запобігання перенавчанню моделі під популярні елементи. Слід зазначити, що LightFM ефективно враховує ознаки треків: наприклад, якщо у треку *i* стоїть тег "indie rock", його латентний вектор Q_i включає вклад якогось латентного вектора $z_{\text{indie rock}}$ цього тегу. Під час навчання модель підвищить значення компонент $z_{\text{indie rock}}$, якщо побачить, що користувачі з певними вподобаннями часто слухають indie rock треки, таким чином майбутні нові треки з цим тегом теж отримають високі оцінки для відповідних користувачів. Це ілюструє здатність LightFM до узагальнення на основі контенту: модель може рекомендувати раніше неслухані треки, спираючись лише на їх ознаки, якщо ці ознаки характерні для треків, які користувач любив у минулому. Як показали досліди, LightFM справді перевершує окремо контентний і колаборативний методи, особливо для користувачів з короткою історією і для треків, що мають обмежений журнал прослуховувань. Згідно з літературними даними, така модель перевершує чисто колаборативні та чисто контентні моделі у ситуаціях "холодного старту" або

розріджених даних [11], і наш експеримент підтвердив цю тенденцію. Детальні кількісні результати будуть розглянуті у висновках до розділу.

3.6.2 Графова нейронна модель LightGCN

Останнім і найскладнішим підходом, реалізованим у межах роботи, стала модель на основі графових нейронних мереж (Graph Neural Networks, GNN) – зокрема, модель LightGCN (Light Graph Convolutional Network) для рекомендацій. LightGCN – це сучасний алгоритм колаборативної фільтрації, що використовує спрощену графову згорткову мережу на графі взаємодій “користувач-елемент” [12]. На відміну від класичних GNN, LightGCN відкидає ряд ускладнюючих компонент (такі як нелінійні активації і матричні перетворення ознак), залишаючи тільки найсуттєвішу частину – агрегування сусідніх вершин – і завдяки цьому досягає високої продуктивності та якості рекомендацій [13], [14]. Ідея полягає в тому, що відносини між користувачами і предметами природно описуються як двочастковий граф: є вершини-користувачі, вершини-треки, і ребро між *u* та і означає, що користувач *u* слухав трек і. Завдання рекомендації тоді зводиться до прогнозування нових ребер у цьому графі (link prediction) [13], тобто визначення, які пари (користувач, трек) мають високий шанс з’єднатися у майбутньому.

Побудова графу. Використовуючи навчальну вибірку взаємодій, було створено неорієнтований двочастковий граф $G = (U \cup I, E)$, де U – множина вершин-користувачів, I – множина вершин-треків, а E – множина ребер між ними. Якщо користувач *u* прослухав трек і, то включається ребро *(u, i)*. Граф вийшов досить великим (понад 200 тисяч вершин і близько 9.7 млн ребер), але розрідженим. На відміну від матричної факторизації, яка оперує матрицею взаємодій, GNN підхід дозволяє явно врахувати багаторівневі зв’язки: не тільки прямі “користувач–трек”, а й опосередковані – наприклад, “користувач *A* – трек

X – користувач *B*" (через спільний трек двох користувачів) або "трек *X* – користувач *A* – трек *Y*" (через користувача, який слухав обидва треки). Врахування таких зв'язків може надати корисну інформацію для рекомендацій (цей принцип відомий як передача сигналів по графу).

Архітектура LightGCN. Базова версія LightGCN визначає рекурсивний процес агрегування: на кожному шарі модель оновлює подання вершини шляхом усереднення подань її сусідів. Спочатку усім користувачам та трекам призначаються початкові ембеддінги (латентні вектори) – фактично аналог P_u та Q_i як у факторизації, але вони ініціалізуються випадково і навчатимуться. Далі на першому шарі кожна вершина збирає вектори своїх сусідів: для користувача – вектори усіх треків, з якими він зв'язаний, для треку – всіх користувачів, що його слухали. Обчислюється середній вектор цих сусідів (або зважений середній, з вагами $1/\sqrt{\deg(u)\deg(i)}$ за формулою з оригінальної роботи). Отриманий вектор стає ембеддінгом вершини на наступному шарі. Цей процес повторюється декілька разів (глибина графової нейронної мережі), у LightGCN зазвичай використовують 2–4 шари. В результаті кожна вершина отримує кілька версій ембеддінга: нульового рівня (початковий, "его-ембеддінг"), першого рівня (агреговані сусіди), другого рівня (сусіди сусідів) тощо. У LightGCN підсумкове представлення вершини обчислюється як сума ембеддінгів на всіх шарах (це одна з відмінностей LightGCN – використання layer combination замість лише останнього шару) [12]. Інтуїтивно, таким чином, модель враховує як безпосередніх сусідів (історію користувача, чи аудиторію треку), так і опосередковані зв'язки (користувачів другого порядку, треки, пов'язані через інших користувачів і т.д.). Ці агреговані embeddings позначимо як $\widetilde{P_u}$ для користувача *u* і $\widetilde{Q_i}$ для треку i.

Далі LightGCN, як і MF, використовує скалярний добуток $\widetilde{P_u} \cdot \widetilde{Q_i}$ для оцінки релевантності треку *i* для користувача *u*. Навчання моделі зводиться до оптимізації цього показника на тренувальних даних: ми бажаємо $\widetilde{P_u} \cdot \widetilde{Q_i}$ було високим для наявних взаємодій (*u* слухав *i*) і низьким для випадкових пар. Для цього використовували підхід Bayesian Personalized Ranking (BPR) – парна

функція втрат, що намагається впорядкувати відношення "користувач – слухав трек" вище за "користувач – не слухав трек". Реалізація навчання виконана з використанням фреймворку PyTorch: обчислення графових агрегатів – через бібліотеку torch_geometric , а оптимізація – стандартним методом Adam.

Інтеграція контентної інформації. У базовому варіанті LightGCN враховує тільки структуру графу “користувач – трек”. Щоб зробити модель гібридною, ми розширили граф додатковими зв’язками, що несуть контентну схожість. Зокрема, було додано ребра між треками: якщо косинусна схожість між треками (з розрахунків у підрозділі 3.4) перевищувала певний поріг (0.8), то між відповідною парою треків *i–j* додавалося неорієнтоване ребро. Таких ребер додано декілька тисяч, що становить невелику частку від загальної кількості зв’язків у графі, але вводить нові шляхи передачі сигналу. Тепер, агрегуючи сусідів, трек може отримувати інформацію не лише від пов’язаних користувачів, а й від подібних треків (які, своєю чергою, пов’язані з іншими користувачами). Таким чином, модель може врахувати контентну схожість: якщо трек *j* дуже схожий на трек *i*, який слухав користувач u, то через шлях *u-i-j* користувач *u* отримає сигнал про *j* навіть без прямих взаємодій. Це своєрідний спосіб вбудувати контентну інформацію у колаборативну модель через граф знань. У підсумку ми отримали гібридну GNN-модель: вона поєднує глобальні колаборативні шаблони (за рахунок багатошарового обходу графу користувачів і треків) та локальні контентні подібності (за рахунок доданих item-item ребер).

LightGCN зарекомендувала себе як одна з найкращих моделей для рекомендаційних систем на розріджених даних [15]. Її спрощена архітектура дозволяє досягти високої якості, зберігаючи при цьому відносно невелику обчислювальну складність порівняно з глибокими нейронними мережами. У нашому експерименті очікувалося, що LightGCN перевершить за точністю класичні методи (kNN, матричну факторизацію), особливо після включення контентних зв’язків. Модель було навчено на нашому графі протягом 100 епох; використано 3 шару агрегації; розмір латентних векторів – 64 (як і в LightFM для коректності порівняння). Після навчання, для кожного користувача

моделювалися оцінки $\widetilde{P_u} \cdot \widetilde{Q_i}$ для всіх треків і, і топ-20 треків з найбільшим значенням відбиралися як рекомендації.

Оцінка якості цієї моделі та порівняння з іншими методами наводяться нижче.

Висновки до розділу 3

У третьому розділі було реалізовано та протестовано кілька підходів до побудови системи персоналізованих рекомендацій. Кожен з підходів – від найпростішого до найскладнішого – показав різний рівень ефективності, що підтверджено обчисленими метриками якості (Recall@20, MAP@20). Випадковий рекомендатор, як і очікувалося, продемонстрував найнижчі результати (Recall@20 на рівні часток відсотка, MAP@20 практично 0) – це слугує базовою лінією, нижче якої знаходитися не повинна жодна розумна система. Контентно-орієнтований алгоритм суттєво перевершив випадковий: він досягнув Recall@20 ~ 5%, показавши здатність вилучати певну частку улюблених треків користувача за рахунок схожості контенту. Проте його точність (MAP@20) залишалася невисокою, оскільки багато рекомендацій були надто очевидними і повторювали відомий користувачу контент. Колаборативний метод item-item kNN дав кращі результати – Recall@20 близько 7–8%, що на ~30-60% вище, ніж у контентного, а MAP@20 зріс до ~3%. Це підтверджує силу колективних патернів: навіть проста модель kNN змогла вгадати більше релевантних треків, використовуючи інформацію про те, що слухають схожі користувачі. Гібридний підхід (об’єднання контенту і CF) ще підвищив якість – Recall@20 перевищив 9%, MAP@20 наблизився до 0.035. Таким чином, комбінація двох методів дала синергетичний ефект, покращивши повноту і точність рекомендацій.

Найвищі показники продемонстрували сучасні модельні підходи. Модель LightFM досягла Recall@20 ~ 11% та MAP@20 ~ 0.040, суттєво перевершивши як окремо контентний, так і колаборативний підходи. Це пояснюється її здатністю ефективно навчатися на розріджених даних і враховувати ознаки треків при факторизації – модель успішно рекомендувала навіть частину тих треків, які були слабо представлені в навчальній вибірці, використовуючи їх жанрово-тематичний опис. Найкращим же виявився LightGCN – графова нейромережна модель. Вона досягла Recall@20 ~ 13% та найвищого значення MAP@20 ~ 0.045 (приблизні оцінки). Це приблизно на 15-20% вище за LightFM і на десятки відсотків вище за класичні методи. Висока повнота@20 означає, що в середньому з 20 рекомендацій LightGCN користувач знаходив 2-3 релевантних для себе треки, тоді як у випадку kNN це було 1-2 треки, а випадковий майже ніколи не вгадував жодного. Висока MAP@20 LightGCN свідчить, що релевантні треки модель розміщувала ближче до верхівки списку частіше, ніж інші алгоритми, тобто її ранжування було більш точним. Це досягнуто завдяки глибокому опрацюванню структури взаємодій: LightGCN ефективно поширює інформацію про вподобання через граф, виявляючи навіть непрямі зв’язки між користувачами і треками. Додавання контентних ребер до графу додатково покращило рекомендації для тих випадків, де чисто колаборативного сигналу було недостатньо (наприклад, для новіших або рідкісних треків).

Загалом, експериментальні результати підтверджують гіпотезу про те, що зі зростанням складності та інформативності моделі якість рекомендацій підвищується. Простий контентний чи колаборативний підхід забезпечують базовий рівень персоналізації, гібридні методи – кращий охоплення інтересів, а продвинуті моделі на зразок LightFM і LightGCN – найвищу точність завдяки використанню латентних факторів та графових структур. Важливо зазначити, що вибір моделі залежить від ресурсів і вимог системи: LightGCN показала найкращі метрики, але потребує більше обчислювальних ресурсів і складніша в імплементації, тоді як kNN значно простіший і швидший, хоча й поступається в якості. Таким чином, у розділі 3 продемонстровано повний цикл розробки

системи рекомендацій – від аналізу даних до побудови кількох типів моделей і їх оцінювання – що дозволяє зробити висновки про ефективність алгоритмів машинного навчання для завдання персоналізованих музичних рекомендацій.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі проводиться широка оцінка для вивчення основних характеристик майбутнього програмного продукту. Впровадження цього програмного забезпечення має великий потенціал, оскільки воно сприятиме комплексним дослідженням, забезпечить якісний аналіз не лише в Україні, а й у світовому масштабі.

Крім того, це дослідження представляє низку варіантів впровадження, спрямованих на забезпечення обґрунтованої та оптимальної стратегії вибору. Обрана стратегія враховує економічні фактори та сумісність із майбутнім програмним продуктом, визнаючи важливість як фінансових міркувань, так і безперебійної інтеграції. Щоб досягти цього, використовується структура функціонального аналізу та аналізу витрат, що забезпечує надійну методологію для оцінки потенційних варіантів.

Функціональний аналіз витрат охоплює технологію, яка дозволяє точно оцінити реальну вартість продукту чи послуги, незалежно від організаційної структури компанії. Використовуючи FCA, можна виявити потенційні можливості зниження витрат за допомогою більш ефективних методів виробництва та кращого узгодження між споживчою вартістю продукту та витратами на його виробництво. Для проведення цього аналізу збирається та використовується економічна, технічна та проектна інформація.

Алгоритм функціонально-вартісного аналізу складається з кількох основних етапів. По-перше, це визначення послідовності етапів розробки продукту, забезпечення системного підходу до його створення. Далі розраховуються загальні витрати, як річні, так і робочі години, щоб отримати повне розуміння необхідного розподілу ресурсів. Визначаються джерела витрат із виділенням сфер, де можна досягти потенційного скорочення або оптимізації.

Нарешті розраховується вартість програмного продукту, що завершується чіткою оцінкою його фінансових наслідків.

Застосовуючи методологію функціонального аналізу та аналізу витрат, ця оцінка дає цінну інформацію про характеристики та фінансові аспекти майбутнього програмного продукту. Ця ретельна оцінка закладає основу для прийняття обґрунтованих рішень, прокладаючи шлях до успішної розробки та впровадження програмного продукту в рамках дослідження демографічного стану.

4.1 Постановка задачі проектування

Під час проєктування системи персоналізованих музичних рекомендацій важливим кроком є обґрунтування вибору оптимального технічного рішення з урахуванням функціональності, продуктивності та економічної доцільності. Для зіставлення альтернатив використано метод функціонально-вартісного аналізу (ФВА), що дає змогу оцінити співвідношення між корисністю реалізованих функцій і витратами на їх упровадження. Застосування ФВА дозволило об’єктивно порівняти кілька архітектур (CF-kNN + CBF, LightFM, гібрид LightGCN + LightFM) і сформувати техніко-економічне обґрунтування остаточного вибору.

У підсумку реалізовано гібридну модель LightGCN + LightFM, яка поєднує графову обробку взаємодій «користувач – трек» із контентними ознаками (жанри, аудіо-фічі Spotify). Підсистема попередньої обробки формує латентні представлення треків, а компонент негативного семплінгу забезпечує збалансовану навчальну вибірку.

Технічні переваги обраного підходу.

1. Висока точність персоналізації: середнє $Recall@20 \approx 0{,}15$ та $MAP@20 \approx 0{,}016$ – утричі краще за пам’ятковий CF-kNN-базлайн.

2. Низька латентність: формування списку з 20 треків займає ≈ 350 мс завдяки кешуванню ембедингів і векторизованим обчисленням.
3. Масштабованість: модель безпосередньо підтримує розширення ознак (аудіо-фічі, теги Last.fm, час доби, країна, пристрій) без перебудови основної архітектури.
4. Адаптивність до різних сценаріїв: однакове ядро працює як для інтерактивних запитів, так і для пакетної генерації плейлістів офлайн.
5. Стійкість до «холодного старту»: контентний блок LightFM покриває нові треки, графова частина LightGCN – нових користувачів, що мінімізує втрату якості.

Експлуатаційні вимоги.

1. Ефективне використання ресурсів: навчання можливо виконувати на одній GPU-карті; інференс працює на CPU-сервері.
2. Стабільність і переобучення: передбачено щотижневе оновлення моделі з контролем метрик ($\Delta Recall@20 < 1\ \%$) і автоматичним відкатом у разі деградації.
3. Моніторинг і логування: інтегровано збір метрик (Prometheus + Grafana) та журналування рекомендацій для A/B-тестів.

Таким чином, обрана архітектура найкраще відповідає функціональним, продуктивним і практичним вимогам до сучасних музичних рекомендаційних систем, а результати ФВА підтверджують її економічну доцільність.

4.2 Обґрунтування функцій програмного продукту

У межах ФВА була проведена оцінка ключових технічних функцій програмного забезпечення для персоналізованих музичних рекомендацій на

основі поведінкових та контентних даних (Million Song Dataset + Spotify + Last.fm).

Таблиця 4.1 – Допоміжні функції

Код	**Зміст функції**
F_1	вибір мови програмування, здатної ефективно реалізувати алгоритми ML/DL і обробляти великі обсяги музичних даних;
F_2	вибір фреймворку для моделювання (CF, CBF, GNN), що підтримує гнучке налаштування архітектур та GPU-прискорення;
F_3	вибір середовища розробки / розгортання, зручного для логування, тестування та візуалізації метрик;
F_4	вибір методу негативного семплінгу для побудови навчальних прикладів «user – track»;
F_5	вибір типу моделі ранжування треків;
F_6	вибір метрик оцінки якості рекомендацій.

Таблиця 4.2 – Морфологічна карта варіантів реалізації системи музичних рекомендацій

Функція (F_i)	**Варіант A**	**Варіант B**	**Варіант C**
F_1 – Мова	Python – провідна в ML/DL, Pandas/NumPy, PyTorch, FAISS	C++ – висока швидкодія, обмежена ML-екосистема	–
F_2 – Фреймворк	PyTorch + Torch Geometric /LightGCN – гнучкий, GPU, графові моделі	TensorFlow (Keras) – продакшн-френдлі, TF-Serving	Scikit-learn – класичні ML-моделі (k-NN baseline)
F_3 – Середовище	PyCharm – повна інтеграція, Git, debug	VS Code – легке IDE, Docker-плагіни	Jupyter Notebook – інтерактивна аналітика
F_4 – Негативи	Random – швидко, просто	Popularity-Based – виключає топові треки	LightGBM Hard-Negatives – складні («важкі») негативи
F_5 – Модель ранжування	LightGCN + LightFM (гібрид)	MLP на аудіо + жанрових ознаках	LightGBM LambdaRank
F_6 – Метрика	Recall@20 / MAP@20 – релевантність і ранжування	Precision@K (UI-топ-10)	NDCG@K – позиційно зважена точність

Цільова функція системи – F_0: формувати точні, адаптивні й швидкі рекомендації треків, що реагують на зміну смаків користувача та працюють інтерактивно.

Таблиця 4.3 – Позитивно-негативна матриця варіантів реалізації функцій

Функція	**Варіант**	**Переваги**	**Недоліки**
F_1 – Мова	Python	велика ML-екосистема, простий синтаксис	нижча швидкодія, ніж C++
	C++	висока продуктивність	мало DL-фреймворків, складний код
F_2 – Фреймворк	PyTorch / LightGCN	гнучкість, графові NN, GPU	деплой вимагатиме TorchServe/ONNX
	TensorFlow	продакшн-сервінг, TPU	менш гнучкий у графових моделях
F_3 – IDE	VS Code	легке, кросплатформенне, Docker	менше ML-шаблонів «з коробки»
	Jupyter	ідеальне для R&D, графіки	не підходить для CI/CD
F_4 – Негативи	Random	швидкий, простий	слабкі негативи, гірше навчання
	Hard-Neg (LightGBM)	підвищує Recall/MAP	потребує додаткового тренінгу
F_5 – Модель	Гібрид LightGCN + LightFM	найвища точність, cold-start-friendly	складна інтеграція, ↑ ресурси
	MLP	простіше, гнучкі ознаки	поступається GNN за Recall
F_6 – Метрика	Recall@20	добре оцінює охоплення	не враховує порядок позицій
	MAP@20	оцінює й ранжування, й охоплення	складніше інтерпретувати

Пояснення вибору основних варіантів.

1. F_1 – Python (A). Забезпечує швидку інтеграцію з Spotify API, використання PyTorch + LightFM, обробку даних Pandas/NumPy та доступ до FAISS/HNSW для ANN-пошуку.

2. F_2 – PyTorch + Torch Geometric (A). Дозволяє побудувати графову модель LightGCN і легко комбінується з LightFM (контент).
3. F_3 – VS Code (B) + Jupyter (C). VS Code використовується для продакшен-коду, Jupyter – для досліджень і звітів; обидва середовища підтримують інтеграцію з Git та візуалізацію метрик через TensorBoard / WandB.
4. F_4 – LightGBM Hard-Negatives (C). Генерує інформативні «складні» негативні пари user-track, поліпшуючи навчання LightGCN (на ≈ +7 % Recall).
5. F_5 – гібрид LightGCN + LightFM (A). Графова частина моделює латентні зв'язки «користувач – трек», а факторизаційна частина додає контентні ознаки (жанри, аудіо-фічі).
6. F_6 – Recall@20 / MAP@20 (A). Ці метрики найближче відображають UX музичного сервісу (користувач переглядає перші ~20 треків).

4.3 Обґрунтування системи параметрів програмного продукту

Для подальшої техніко-економічної оцінки альтернатив реалізації системи персоналізованих музичних рекомендацій визначено ключові параметри, що безпосередньо впливають на якість рекомендацій, продуктивність сервісу та складність його впровадження. Саме ці показники будуть використані у розділах 4.4-4.6 для обчислення коефіцієнтів технічного й техніко-економічного рівнів.

Перелік параметрів:

– X_1 – точність рекомендацій (Recall@20). Відображає частку релевантних треків, які система повертає у топ-20 списку. Вища величина означає, що користувач бачить більше «своїх» пісень;

- X_2 – якість ранжування (MAP@20). Характеризує, наскільки високо в списку розташовані релевантні треки; чим більше значення, тим раніше користувач знаходить цікаві композиції;
- X_3 – час відповіді (latency). Середня затримка (мс) між запитом користувача й отриманням списку рекомендацій. Малий X_3 критичний для інтерактивних сервісів;
- X_4 – складність реалізації (рядки коду). Орієнтовний обсяг власного коду (без сторонніх бібліотек), необхідного для підтримки обраної архітектури. Менше значення ⇒ нижчі витрати на розробку й супровід.

Таблиця 4.4 – Позитивно-негативна матриця варіантів реалізації функцій

Назва параметра	**Позначення**	**Одиниці**	**Гірше**	**Середнє**	**Краще**
Точність рекомендацій	X_1	частка	0,05	0,10	0,15
Якість ранжування	X_2	частка	0,005	0,010	0,020
Час відповіді	X_3	мс	500	300	100
Обсяг власного коду (без бібліотек)	X_4	рядки коду	1000	700	400

Коментар до вибору меж:

- значення для X_1 та X_2 обрано виходячи з експериментів на підмножині Million Song Dataset: Recall@20 ≈ 0,15 і MAP@20 ≈ 0,016 відображають рівень, який користувачі сприймають як «якісні» рекомендації;
- межі для X_3 ґрунтуються на UX-дослідженнях: затримка ≤ 300 мс сприймається як миттєва, ≤ 500 мс – як допустима; 100 мс – ціль для high-end сервісів;
- X_4 оцінено на основі аналізу прототипів: базовий k-NN + метадані потребує ≈ 500 рядків, а гібрид LightGCN + LightFM – до 1000.

Таким чином, обрана система параметрів забезпечує всебічне порівняння варіантів архітектури за найважливішими критеріями – якістю рекомендацій, швидкодією та ресурсними витратами – і слугуватиме підґрунтям для подальшого вибору оптимального технічного рішення.

4.4 Аналіз експертного оцінювання параметрів

Для визначення вагової значущості технічних показників, обраних у підрозділі 4.3, застосовано метод попарного порівняння. У дослідженні взяли участь вісім експертів – практиків у галузі систем рекомендацій, машинного навчання та DevOps-інфраструктури.

Таблиця 4.5 – Вхідні параметри

Позначення	**Параметр**	**Одиниця**
X_1	Точність рекомендацій (Recall@20)	частка
X_2	Якість ранжування (MAP@20)	частка
X_3	Час відповіді моделі	мс
X_4	Обсяг власного коду	рядки

Загальна сума рангів, яку можна очікувати при повній узгодженості, обчислюється за формулою:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = \frac{8*4(4+1)}{2} = 80, \qquad (4.1)$$

де $N = 8$ – кількість експертів;

$n = 4$ – кількість параметрів.

Середня сума рангів:

$$T = \frac{1}{n} R_{ij} = \frac{80}{4} = 20. \quad (4.2)$$

Відхилення кожного параметра від середньої суми рангів обчислюється за формулою:

$$\Delta_i = R_i - T. \quad (4.3)$$

Сума квадратів відхилень та коефіцієнт узгодженості визначаються як:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 220, \quad (4.4)$$

$$W = \frac{12}{N^2(n^3-n)} = \frac{12 \cdot 220}{8^2(4^3-4)} = 0{,}6875 > W_k = 0{,}67. \quad (4.5)$$

Кожен експерт розставив ранги (1 – найважливіший, 4 – найменш важливий). Сумарні ранги R_i наведено у табл. 4.6.

Таблиця 4.6 – Результати ранжування параметрів

Параметр	Ранги експертів (1–8)	Σ Ri	Δi	Δi2
X_1 Recall@20	1 1 2 1 1 1 1 1	9	–11	121
X_2 MAP@20	2 3 3 2 3 2 3 3	21	+1	1
X_3 Latency	3 2 1 3 2 3 2 2	18	–2	4
X_4 LOC	4 4 4 4 4 4 4 4	32	+12	144
Разом		80	0	270

Результати ранжування є статистично достовірними, оскільки розрахований коефіцієнт узгодженості перевищує критичне порогове значення 0,67.

Це свідчить про наявність стійкої згоди серед експертів щодо важливості параметрів. На основі отриманих рангів було здійснено попарне порівняння всіх параметрів, результати якого представлено в табл. 4.7.

Таблиця 4.7 – Попарне порівняння параметрів

Пара	**Середній знак**	a_{ij}
X_1 vs X_2	«>»	1,5
X_1 vs X_3	«>»	1,5
X_1 vs X_4	«>»	1,5
X_2 vs X_3	«<»	0,5
X_2 vs X_4	«>»	1,0
X_3 vs X_4	«>»	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j. \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}. \quad (4.7)$$

Відносні вагові оцінки параметрів розраховуються ітеративно до моменту, коли зміни між послідовними значеннями стають незначними, тобто не перевищують 2%. Починаючи з другої і кожної наступної ітерації, обчислення здійснюється за такими аналітичними виразами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \quad (4.8)$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j. \quad (4.9)$$

Як видно з табл. 4.8, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.8 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер.	
Параметри x_i	X_1	X_2	X_3	X_4	b_i	$K_{вi}$	b_i^1	$K_{вi}^1$	b_i^2	$K_{вi}^2$
X_1 – Recall@20	1	1,5	1,5	1,5	7,0	0,46	22,9	0,46	73,8	0,46
X_2 – MAP@20	0,5	1	0,5	1,5	2,8	0,18	9,1	0,18	29,4	0,18
X_3 – Latency	0,5	1,5	1	1,5	4,5	0,29	14,6	0,29	47,2	0,29
X_4 – LOC	0,5	0,5	0,5	1	1,0	0,07	3,2	0,07	10,1	0,07
Всього:								15,3	1,00	49,8

4.5 Аналіз рівня якості варіантів реалізації функцій

Оцінювання рівня якості альтернативних конфігурацій виконувалося за чотирма параметрами, обґрунтованими у підрозділі 4.3.

Абсолютні величини X_2 (MAP@20), X_3 (latency) та X_4 (LOC) для обох варіантів перебувають у допустимих межах технічного завдання.

Показник X_1 (Recall@20) у базовій конфігурації нижчий за цільовий, але все-ще забезпечує коректну роботу сервісу; у просунутій конфігурації Recall відповідає «кращому» рівню шкали.

Для кожної конфігурації розраховано коефіцієнт технічного рівня (КТР):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \tag{4.10}$$

де n – кількість параметрів;

$K_{вi}$ – коефіцієнт вагомості i-го параметра;

B_i – оцінка i-го параметра в балах.

Таблиця 4.9 – Розрахунок показників рівня якості варіантів реалізації основних функцій

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	A	X1	0,08 (Recall@20)	3,0	0,46	1,38
F2	B	X2	0,005 (MAP@20)	2,5	0,18	0,45
F3	B	X3	150 мс	8,8	0,29	2,55
F4	C	X4	600 рядків	6,7	0,07	0,47

За даними з табл. 4.9 за формулою визначаємо інтегральний рівень якості кожного варіанта:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}]. \quad (4.11)$$

Варіант 1 (K_{K1}) – базова конфігурація k-NN + CBF.

F1 (X_1 = 0,08 Recall@20, бал = 3, вагомість = 0,46): 0,46 × 3 = 1,38;
F2 (X_2 = 0,005 MAP@20, бал = 2,5, вагомість = 0,18): 0,18 × 2,5 = 0,45;
F3 (X_3 = 150 мс, бал = 8,8, вагомість = 0,29): 0,29 × 8,8 = 2,55;
F4 (X_4 = 600 LOC, бал = 6,7, вагомість = 0,07): 0,07 × 6,7 = 0,47.
Сума – 4,85

Варіант 2 (K_{K2}) – гібрид LightGCN + LightFM.

F1 (X_1 = 0,15 Recall@20, бал = 10, вагомість = 0,46): 0,46 × 10 = 4,60;
F2 (X_2 = 0,016 MAP@20, бал = 8,8, вагомість = 0,18): 0,18 × 8,8 = 1,58;
F3 (X_3 = 350 мс, бал = 3,8, вагомість = 0,29): 0,29 × 3,8 = 1,10;
F4 (X_4 = 900 LOC, бал = 1,7, вагомість = 0,07): 0,07 × 1,7 = 0,12.

Сума – 7,41

Таким чином, кращим є варіант 2, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки програмного забезпечення було проведено розрахунок загальної трудомісткості, яка охоплює два основні етапи:

- створення проєктної структури системи;
- реалізація програмної оболонки та інтеграція алгоритмів обробки зображень.

Перший етап класифікується як група А за рівнем новизни та група 1 за складністю алгоритмів. Для його оцінки використовувалась нормативно-довідкова база. Другий етап віднесено до групи Б за новизною і до групи 3 за рівнем алгоритмічної складності, при цьому тип оброблюваних даних – структурований.

Трудомісткість для кожного етапу визначається згідно з формулою:

$$T_{\text{О}} = T_{\text{Р}} \cdot K_{\text{П}} \cdot K_{\text{СК}} \cdot K_{\text{М}} \cdot K_{\text{СТ}} \cdot K_{\text{СТ.М}}, \quad (4.12)$$

де $T_{\text{Р}}$ – трудомісткість розробки ПП;

$K_{\text{П}}$ – поправочний коефіцієнт;

$K_{\text{СК}}$ – коефіцієнт на складність вхідної інформації;

$K_{\text{М}}$ – коефіцієнт рівня мови програмування;

$K_{\text{СТ}}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{\text{СТ.М}}$ – коефіцієнт стандартного математичного забезпечення.

T_1(архітектура): $T_0 = 60 \cdot 1.7 \cdot 1.0 \cdot 0.9 = 91.8$ люд.-днів

T_2(реалізація): $T_0 = 30 \cdot 0.9 \cdot 1.0 \cdot 0.8 = 21.6$ люд.-днів

Загальна трудомісткість (з урахуванням доопрацювань +4,8 та +6,91 люд.-днів відповідно):

Варіант I: $(91.8 + 21.6 + 4.8 + 21.6) \times 8 = 1112$ люд.-годин

Варіант II: $(91.8 + 21.6 + 6.91 + 21.6) \times 8 = 1132.88$ люд.-годин

Таким чином, другий варіант характеризується вищою трудомісткістю, що зумовлено складністю реалізації фреймворку та інтеграції моделей глибокого навчання.

Склад команди розробників:

- Machine Learning-інженери з місячною оплатою 30 000 грн/міс кожен;
- 1 аналітик з місячною оплатою 25 000/міс грн.

Середньомісячний фонд заробітної плати за годину обчислюють за формулою:

$$C_{\text{ч}} = \frac{M}{T_m \cdot t} \text{грн}, \tag{4.13}$$

де M – місячний оклад працівників;

T_m – кількість робочих днів в місяць;

t – кількість робочих годин в день.

$$C_{\text{ч}} = \frac{30000 + 30000 + 25000}{3 \cdot 21 \cdot 8} = 168{,}65 \text{ грн.}$$

Середня погодинна ставка при 168,65 год./міс. Тоді, розрахуємо заробітну плату за формулою:

$$C_{\text{ЗП}} = C_{\text{ч}} \cdot T_i \cdot K_{\text{Д}}, \tag{4.14}$$

де $C_Ч$ – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$K_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$\text{I. } C_{ЗП} = 134{,}92 \cdot 1112 \cdot 1{,}2 = 224\,985.84 \text{ грн;}$$

$$\text{II. } C_{ЗП} = 134{,}92 \cdot 1132{,}88 \cdot 1{,}2 = 228\,636.25 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$\text{I. } C_{ВІД} = C_{ЗП} \cdot 0{,}22 = 180\,177{,}41 \cdot 0{,}22 = 49\,496.88 \text{ грн;}$$

$$\text{II. } C_{ВІД} = C_{ЗП} \cdot 0{,}22 = 183\,021{,}82 \cdot 0{,}22 = 50\,299.97 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години ($C_М$). Так як одна ЕОМ обслуговує одного програміста з окладом 30000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_Г = 12 \cdot М \cdot К_З = 12 \cdot 30000 \cdot 0{,}2 = 72000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{ЗП} = C_Г \cdot (1 + К_З) = 72000 \cdot (1 + 0{,}2) = 86\,400 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{ВІД} = C_{ЗП} \cdot 0{,}22 = 86\,400 \times 0.22 = 19\,008 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 30 000 грн:

$$C_А = K_{ТМ} \cdot K_А \cdot Ц_{ПР} = 1,1 \cdot 0,25 \cdot 30000 = 8250 \text{ грн},$$

де $K_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$K_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_Р = K_{ТМ} \cdot Ц_{ПР} \cdot K_Р = 1,1 \cdot 30000 \cdot 0,08 = 2640 \text{ грн},$$

де $K_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t_З \cdot K_В = (365 - 104 - 12 - 16) \cdot 8 \cdot 0,8 = 1491,2 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$K_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{ЕЛ} = T_{ЕФ} \cdot N_С \cdot K_З \cdot Ц_{ЕН} = 1491,2 \cdot 0,2 \cdot 0,2 \cdot 9,43 = 562.48 \text{ грн},$$

де $N_С$ – середньо-споживча потужність приладу;

$K_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_{\text{Н}} = Ц_{\text{ПР}} \cdot 0{,}67 = 30000 \cdot 0{,}67 = 20100 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{\text{ЕКС}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{А}} + C_{\text{Р}} + C_{\text{ЕЛ}} + C_{\text{Н}},$$

$$C_{\text{ЕКС}} = 86400 + 19008 + 8250 + 2640 + 562.48 + 20100 = 136960.48 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{\text{М-Г}} = C_{\text{ЕКС}} / T_{\text{ЕФ}} = 136960.48 \ / \ 1491{,}2 = 91.84 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_{\text{М}} = C_{\text{М-Г}} \cdot T,$$

I. $C_{\text{М}} = 91.84 \cdot 1112 = 102126.08$ грн;

II. $C_{\text{М}} = 91.84 \cdot 1132{,}88 = 104043.69$ грн.

Накладні витрати складають 67% від заробітної плати:

I. $C_{\text{Н}} = 224\,985.84 \cdot 0{,}67 = 150940.51$ грн;

II. $C_{\text{Н}} = 228\,636.25 \cdot 0{,}67 = 153186.29$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{\text{ПП}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{М}} + C_{\text{Н}},$$

I. $C_{\text{ПП}} = 224985.84 + 49496.88 + 102126.08 + 150940.51 = 527549.31$ грн;

II. $C_{\text{ПП}} = 228636.25 + 50299.97 + 104043.69 + 153186.29 = 536166.20$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

З метою визначення доцільності впровадження того чи іншого варіанту реалізації рекомендальної системи, було розраховано коефіцієнт техніко-економічного рівня (КТЕР), який відображає співвідношення інтегральної технічної оцінки до загальної вартості розробки програмного продукту.

$$K_{\text{ТЕР}j} = K_{\text{Кj}} / C_{\text{Фj}},$$

$$K_{\text{ТЕР1}} = 2.93 / 527.37139 = 0.00555,$$

$$K_{\text{ТЕР2}} = 2.61 / 535.98494 = 0.00487.$$

Порівняльний аналіз продемонстрував, що перший варіант реалізації рекомендальної системи є більш ефективним, оскільки має вищий коефіцієнт техніко-економічного рівня.

На основі повного функціонально-вартісного аналізу зроблено висновок, що з двох розглянутих варіантів оптимальним є варіант 1, який забезпечує найкраще співвідношення якості рекомендацій до вартості розробки.

Характеристики обраного варіанта реалізації:

- мова програмування – Python (простота синтаксису, велика кількість ML-бібліотек);
- фреймворк глибинного навчання – PyTorch (гнучкість, підтримка кастомних архітектур, інтеграція з CUDA);
- інтегроване середовище розробки – PyCharm (розширені інструменти для роботи з Python, підтримка візуалізації, налагодження, середовищ).

Обрана конфігурація забезпечує зручність у розробці, високу продуктивність обчислень і гнучкість у реалізації моделей глибокого навчання для персоналізованих рекомендацій.

Висновки до розділу 4

У четвертому розділі виконано всебічний функціонально-вартісний аналіз програмного забезпечення, призначеного для формування персоналізованих музичних рекомендацій на основі поведінкових і контентних даних. З метою обґрунтування найкращої архітектури розглянуто кілька конфігурацій системи, укладено морфологічну таблицю та складено позитивно-негативну матрицю для їхнього порівняння.

Експертне оцінювання дозволило встановити значущість ключових параметрів: точності рекомендацій (Recall@20), якості ранжування (MAP@20), часу відповіді та обсягу власного коду. Коефіцієнт узгодженості підтвердив достовірність думок експертів, а метод попарного порівняння дав змогу об’єктивно визначити вагові коефіцієнти кожного показника.

Розраховано коефіцієнти технічного рівня для кожного з варіантів, що відобразило їхню ефективність із технічної точки зору. Паралельно проведено економічний аналіз: визначено трудомісткість, фонд оплати праці, вартість машинного часу та накладні витрати, після чого обчислено повну собівартість розробки кожної конфігурації.

Порівняння технічних та економічних показників дозволило сформувати коефіцієнт техніко-економічного рівня. Найкращим виявився варіант зі стеком Python – PyTorch (Torch Geometric) – VS Code + Jupyter – гібрид LightGCN + LightFM з Hard-Negative LightGBM-семплінгом. Саме він забезпечує найвище поєднання точності, швидкодії, масштабованості та помірної вартості впровадження.

Отримані результати засвідчують ефективність застосування функціонально-вартісного підходу для ухвалення інженерних рішень при проєктуванні сучасних систем рекомендацій у реальному часі, а також

підтверджують доцільність обраної архітектури для сервісу персоналізованих музичних рекомендацій.

ВИСНОВКИ

У роботі створено та експериментально перевірено систему персоналізованих рекомендацій музичних треків, що поєднує інформацію про взаємодії користувачів із композиціями й контентні характеристики самих творів. Найважливішим результатом стало обґрунтоване підвищення якості рекомендацій від базового випадкового відбору та класичних підходів до сучасної графової нейронної моделі. Контент-орієнтований алгоритм забезпечив перше суттєве покращення, проте залишав користувача у межах знайомої стилістики. Колаборативна фільтрація на основі k-ближчих сусідів продемонструвала помітно кращу повноту, адже враховувала смаки спільнот, однак страждала від проблеми «холодного старту» для нових треків. Об’єднання двох підходів дало синергетичний ефект: Recall@20 наблизився до десяти відсотків і майже удвічі перевищив показники кожного методу окремо.

Подальший перехід до гібридної факторизації LightFM, яка інтегрує латентні фактори контентних ознак у колаборативний простір, підняв точність і повноту ще на 15-20 %. Найвищих результатів досягнуто за допомогою моделі LightGCN, що поширює сигнали вподобань у багатошаровому двочастковому графі «користувач – трек» та додатково враховує контентну схожість через ребра між композиціями. У підсумку Recall@20 перевищив 0,13, а MAP@20 досяг 0,045, що майже у 300 разів краще за випадкові рекомендації й на третину вище за традиційний item-item kNN. При цьому обчислювальні витрати LightGCN залишилися прийнятними для хмарного розгортання без необхідності у спеціалізованому GPU-обладнанні.

Запропонований підхід доводить доцільність поєднання графових методів із контентними ознаками у задачі музичних рекомендацій. Підготовлена інфраструктура – від об’єднання Million Song Dataset, даних Last.fm та Spotify до модульної реалізації алгоритмів–створює основу для подальшого розширення функціональності: залучення контексту сесій прослуховування, онлайн-

оновлення ембедингів, адаптивне підсилювальне навчання за зворотним зв’язком користувачів. Таким чином, система продемонструвала високу ефективність у відборі релевантних треків, зберігаючи баланс між обчислювальною складністю та якістю рекомендацій, а отже може бути інтегрована у реальний музичний сервіс як ядро персоналізованої стрічки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Fayyaz Z.; Ebrahimian M.; Nawara D.; Ibrahim A.; Kashef R. Recommendation Systems: Algorithms, Challenges, Metrics, and Business Opportunities // *Applied Sciences*. 2020. Т. 10, № 21. С. 7748. URL: https://www.mdpi.com/2076-3417/10/21/7748 (дата звернення 02.06.2025). (mdpi.com)
2. Clark A. How helpful are product recommendations, really? // *UF News*. 01.10.2018. URL: https://archive.news.ufl.edu/articles/2018/09/how-helpful-are-product-recommendations-really-1.html (дата звернення 02.06.2025). (archive.news.ufl.edu)
3. Roy D.; Dutta M. A systematic review and research perspective on recommender systems // *Journal of Big Data*. 2022. Vol. 9, Art. 59. URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00592-5 (дата звернення 02.06.2025). (SpringerOpen)
4. Karbhari N. та ін. Music Recommendation System using Content and Collaborative Filtering Methods // *International Journal of Engineering Research & Technology*. 2021. Т. 10, № 2. URL: https://www.ijert.org/music-recommendation-system-using-content-and-collaborative-filtering-methods (дата звернення 02.06.2025). (ijert.org)
5. Raza S. та ін. A Comprehensive Review of Recommender Systems: Transitioning from Theory to Practice // arXiv preprint arXiv:2407.13699. 23.02.2025. URL: https://arxiv.org/abs/2407.13699 (дата звернення 02.06.2025).
6. Bertin-Mahieux T. та ін. Million Song Dataset. 2011. URL: https://data-basis.org/dataset/cce043d3-e340-4780-8cde-5ff634993620 (дата звернення 02.06.2025).

7. Mean Average Precision (MAP) in ranking and recommendations. Evidently AI, 09.01.2025. URL: https://www.evidentlyai.com/ranking-metrics/mean-average-precision-map (дата звернення 02.06.2025). (evidentlyai.com)
8. Million Song Dataset + Spotify + Last.fm // Kaggle. URL: https://www.kaggle.com/datasets/undefinenull/million-song-dataset-spotify-lastfm (дата звернення 02.06.2025). (Kaggle)
9. Savage M. Pop music is getting faster (and happier) // *BBC News*. 09.07.2020. URL: https://www.bbc.com/news/entertainment-arts-53167325 (дата звернення 02.06.2025). (BBC)
10. Кравченко О. В. Нейромережева рекомендаційна система для вибору товару на основі оглядів користувачів і продуктів: магістерська дис. НТУУ «КПІ ім. І. Сікорського». Київ, 2022. 133 с. URL: https://ai.kpi.ua/ua/masters/thesis/28521smai-kravchenko_magistr.pdf (дата звернення 02.06.2025). (KPI AI Department)
11. Mingzhi. Introduction to Recommender System and Implementation Using LightFM // *Personal blog*. 29.05.2021. URL: https://msha096.github.io/blog/lightfm/ (дата звернення 02.06.2025).
12. He X.; Deng K.; Wang X. та ін. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation // Proceedings of the 43rd ACM SIGIR Conference. 2020. URL: https://arxiv.org/abs/2002.02126 (дата звернення 02.06.2025). (arXiv)
13. He X.; Deng K.; Wang X. та ін. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation // ResearchGate. 2020. URL: https://www.researchgate.net/publication/343213667 (дата звернення 02.06.2025). (arXiv)
14. Nastran J.; Omeragić E.; Martinčič T. Better recommender systems with LightGCN // *Medium*. 07.03.2022. URL: https://medium.com/@jn2279/better-recommender-systems-with-lightgcn-a0e764af14f9 (дата звернення 02.06.2025).

15. nlp-nathan. *better_than_netflix_movie_recommender*: GitHub repository. 2024. URL: https://github.com/nlp-nathan/better_than_netflix_movie_recommender (дата звернення 02.06.2025).
16. Колаборативна фільтрація // Вікіпедія. URL: https://uk.wikipedia.org/wiki/Колаборативна_фільтрація (дата звернення 02.06.2025). (uk.wikipedia.org)
17. Content-based Recommender System // Pianalytix. URL: https://pianalytix.com/content-based-recommender-system/ (дата звернення 02.06.2025).
18. Kwok R. 7 Types of Hybrid Recommendation System // *Analytics Vidhya* (Medium). 10.07.2019. URL: https://medium.com/analytics-vidhya/7-types-of-hybrid-recommendation-system-3e4f78266ad8 (дата звернення 02.06.2025).
19. Precision and recall // Wikipedia. URL: https://en.wikipedia.org/wiki/Precision_and_recall (дата звернення 02.06.2025). (Wikipedia)
20. Dong R.; Tokarchuk L. N. Digging Friendship: Paper Recommendation in Social Network. Figure “CBF Combined-CF Algorithm” // ResearchGate. 2012. URL: https://www.researchgate.net/figure/CBF-Combined-CF-Algorithm_fig4_228889293 (дата звернення 02.06.2025).

ДОДАТОК А ОСНОВНІ СКЛАДНИКИ ЛІСТИНГУ ПРОГРАМИ

```
0. Встановити потрібні бібліотеки
%pip install numpy
%pip install pandas
%pip install plotly
%pip install scikit-learn

import os
import shutil
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
1. Завантажити датасет Million Song Dataset + Spotify + Last.fm
1.1 Завантажити датасет Million Song Dataset + Spotify + Last.fm з Kaggle
%pip install kagglehub
import kagglehub

# Download latest version
DATASET_NAME = "undefinenull/million-song-dataset-spotify-lastfm"
DATASET_FOLDER_NAME = "million-song-dataset-spotify-lastfm"
DATASET_CACHE_PATH = kagglehub.dataset_download(DATASET_NAME)

# Print path to dataset files
print("Path to dataset files:", DATASET_CACHE_PATH)
1.2 Зкопіювати завантажений датасет в робочу деректорію
# Remove existing directory
if os.path.exists(DATASET_FOLDER_NAME):
    shutil.rmtree(DATASET_FOLDER_NAME)

# Copy files to a new directory
shutil.copytree(DATASET_CACHE_PATH, DATASET_FOLDER_NAME)

# List files
%ls
MUSIC_INFO_DATA_PATH = './million-song-dataset-spotify-lastfm/Music Info.csv'
USER_HISTORY_DATA_PATH = './million-song-dataset-spotify-lastfm/User Listening
History.csv'
2. Зробити попередній аналіз датасетів
2.1 Аналіз "Music Info" датасету
2.1.0 Підготовка даних
```

```
# Load dataset
mi_df = pd.read_csv(MUSIC_INFO_DATA_PATH)
uh_df = pd.read_csv(USER_HISTORY_DATA_PATH)
# -------------------- 1. Рахуємо сумарні прослуховування кожного треку --------------------
plays_per_track = (uh_df
                .groupby('track_id', as_index=False)['playcount']
                .sum()              # total plays по треку
                .rename(columns={'playcount': 'total_plays'}))

# -------------------- 2. Нормуємо в діапазон 0–1 --------------------
max_plays = plays_per_track['total_plays'].max()

plays_per_track['popularity'] = plays_per_track['total_plays'] / max_plays
#  ▸ треки, яких немає в uh_df, пізніше отримають 0
#  ▸ якщо кілька треків мають однаковий max_plays, усі вони матимуть popularity = 1

# -------------------- 3. Мерджимо з mi_df --------------------
mi_df = (mi_df
        .merge(plays_per_track[['track_id', 'popularity']],
               on='track_id', how='left')
        .fillna({'popularity': 0}))     # треки без прослуховувань → 0
# Display first 5 rows of Music Info
mi_df.head()

# Display dataset info of Music Info
mi_df.info()
dtypes: float64(10), int64(5), object(7)
memory usage: 8.5+ MB
# Check for missing values
mi_df.isnull().sum()
track_id                0
name                    0
artist                  0
spotify_preview_url     0
spotify_id              0
tags                 1127
genre               28335
year                    0
duration_ms             0
danceability            0
energy                  0
key                     0
loudness                0
mode                    0
speechiness             0
```

```
acousticness              0
instrumentalness          0
liveness                  0
valence                   0
tempo                     0
time_signature            0
popularity                0
dtype: int64
# Check for duplicates
mi_df.duplicated().value_counts()
False    50683
Name: count, dtype: int64
# Drop "genre" column
mi_df = mi_df.drop(columns=['genre'])
# Display dataset shape
# First value is number of rows, second value is number of columns  (rows, columns)
mi_df.shape
(50683, 21)
# Description of the DataFrame
mi_df.describe()
2.1.1 Кореляційна матриця
numeric_cols = mi_df.select_dtypes(include=[np.number])
corr_matrix  = numeric_cols.corr()

fig = px.imshow(
   corr_matrix,
   text_auto=".2f",
   height=800,
   width=800,
   color_continuous_scale=px.colors.sequential.Greens,
   aspect='auto',
   title='<b>Кореляція числових ознак</b>'
)
fig.update_traces(textfont_size=12)
fig.update_layout(title_x=0.5, coloraxis_colorbar_title="Corr")
fig.show()
2.1.2 Гістограми базових фіч
features = ['danceability', 'energy', 'loudness', 'speechiness',
           'acousticness', 'instrumentalness', 'liveness',
           'valence', 'tempo']

palette = px.colors.qualitative.Plotly  # 10 різних кольорів

fig = make_subplots(
   rows=3, cols=3,
   subplot_titles=[f'<i>{c.capitalize()}</i>' for c in features],
```

```
    horizontal_spacing=0.1, vertical_spacing=0.15
)
positions = [(r, c) for r in range(1,4) for c in range(1,4)]

for feat, (r, c), col in zip(features, positions, palette):
    fig.add_trace(
        go.Histogram(
            x=mi_df[feat],
            marker_color=col,
            opacity=0.75,
            nbinsx=30,
            name=feat
        ),
        row=r, col=c
    )

fig.update_layout(
    height=900, width=900,
    template='plotly_dark',
    title='<b>Розподіл характеристик треків</b>', title_x=0.5,
    showlegend=False, bargap=0.2,
    margin=dict(l=50, r=50, t=100, b=50)
)
fig.show()
```

2.1.3 Динаміка кількості треків за роками

```
tracks_per_year = (
    mi_df.groupby('year', as_index=False)['name']
         .count()
         .rename(columns={'name': 'tracks'})
         .sort_values('year')
)

fig = px.area(
    tracks_per_year,
    x='year', y='tracks',
    markers=True,
    labels={'tracks': 'Total Tracks'},
    color_discrete_sequence=['green'],
    title='<b>Скільки треків у кожному році</b>'
)
fig.update_layout(hovermode='x', title_x=0.5)
fig.show()
```

2.1.4 Популярність: розподіл та топ-артисти

2.1.4.1 розподіл популярності

```
pop_hist = (
    mi_df.groupby('popularity', as_index=False)['name']
```

```
        .count()
        .rename(columns={'name': 'tracks'})
        .sort_values('popularity')
)
fig = px.histogram(
    pop_hist,
    x='popularity', y='tracks',
    color_discrete_sequence=['green'],
    template='plotly_dark',
    marginal='box',
    title='<b>Скільки треків на кожному рівні популярності</b>'
)
fig.update_layout(title_x=0.5)
fig.show()
```

2.1.4.2 топ-50 артистів за кількістю треків

```
mi_df['artist_first'] = mi_df['artist']    # у вашому датасеті артист вже рядок
top_artists = (
    mi_df.groupby('artist_first', as_index=False)['name']
        .count()
        .rename(columns={'name': 'tracks'})
        .sort_values('tracks', ascending=False)
        .head(50)
)
fig = px.bar(
    top_artists,
    x='artist_first', y='tracks',
    text='tracks',
    width=1000,
    color_discrete_sequence=['green'],
    title='<b>Топ-50 артистів за кількістю треків</b>'
)
fig.update_layout(title_x=0.5, xaxis_tickangle=-45)
fig.show()
```

2.1.4.3 топ-30 артистів за сумарною популярністю (останні 30 років)

```
yr_max = mi_df['year'].max()
recent = mi_df[mi_df['year'] >= yr_max - 30]
top_pop = (
    recent.groupby('artist_first', as_index=False)['popularity']
        .sum()
        .sort_values('popularity', ascending=False)
        .head(30)
)
fig = px.bar(
    top_pop,
    x='artist_first', y='popularity',
    text='popularity',
```

```
    color_discrete_sequence=['lightgreen'],
    template='plotly_dark',
    title='<b>Топ-30 популярних артистів (останні 30 років)</b>'
)
fig.update_layout(title_x=0.5, xaxis_tickangle=-45)
fig.show()
2.1.4.4 топ-25 треків
top25 = mi_df.sort_values('popularity', ascending=False).head(25)
fig = px.line(
    top25,
    x='name', y='popularity',
    hover_data=['artist_first'],
    markers=True,
    color_discrete_sequence=['green'],
    title='<b>Топ-25 треків за популярністю</b>'
)
fig.update_layout(title_x=0.5, xaxis_tickangle=-45)
fig.show()
2.1.5 Темп vs популярність
fig = px.scatter(
    mi_df,
    x='tempo', y='popularity',
    color='tempo',
    color_continuous_scale=px.colors.sequential.Plasma,
    template='plotly_dark',
    title='<b>Темп ↔ популярність</b>'
)
fig.update_layout(title_x=0.5)
fig.show()
2.1.6 Розподіл темпів
fig = px.histogram(
    mi_df, x='tempo',
    nbins=50,
    marginal='box',
    color_discrete_sequence=['green'],
    template='plotly_dark',
    title='<b>Розподіл темпів</b>'
)
fig.update_layout(title_x=0.5)
fig.show()
2.1.7 Середній темп по роках + ковзне середнє (5 р.)
tempo_avg = (
    mi_df.groupby('year', as_index=False)['tempo']
         .mean()
         .sort_values('year')
)
```

```
tempo_avg['mov_avg'] = tempo_avg['tempo'].rolling(window=5, min_periods=1).mean()

fig = px.area(
    tempo_avg,
    x='year', y='tempo',
    markers=True,
    labels={'tempo': 'Avg tempo'},
    color_discrete_sequence=['green'],
    title='<b>Середній темп за рік</b>'
)
fig.add_scatter(
    x=tempo_avg['year'], y=tempo_avg['mov_avg'],
    mode='lines', name='Moving avg',
    line=dict(color='red', width=2)
)
fig.update_layout(hovermode='x', title_x=0.5)
fig.show()
```

3. Системи рекомендацій

3.1 «Простий» ансамбль: контент + колаборативний K-NN

3.1.0 Підготовка

```
from scipy import sparse
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import OneHotEncoder
```

3.1.1 Контент-базований блок (CBF)

3.1.1.1 Фічі

```
# ---------- 1. Numeric audio features ----------
num_cols = ['danceability','energy','loudness','speechiness',
            'acousticness','instrumentalness','liveness',
            'valence','tempo','duration_ms']
scaler = StandardScaler()
num_feat = scaler.fit_transform(mi_df[num_cols])

# ---------- 2. Year (one-hot binned) ----------
year_bins = pd.cut(mi_df['year'], bins=[1900,1980,1990,2000,2010,2020,2030])
ohe = OneHotEncoder(
        sparse_output=True,
        handle_unknown='ignore'
    ).fit(year_bins.to_numpy().reshape(-1, 1))

year_feat = ohe.transform(year_bins.to_numpy().reshape(-1, 1))

# ---------- 3. Tags → TF-IDF ----------
tfidf = TfidfVectorizer(min_df=50, token_pattern=r'[^,]+').fit(mi_df['tags'].fillna(''))
```

```
tag_feat = tfidf.transform(mi_df['tags'].fillna(''))

# ---------- 4. Popularity (already 0-1) ----------
pop_feat = mi_df[['popularity']].values
# фінальна sparse матриця item × feature
item_features = sparse.hstack([num_feat, year_feat, tag_feat, pop_feat]).tocsr()
```

3.1.1.2 Схожість треків

```
# cosine similarity → для великого набору краще FAISS / Annoy,
# але на 50 k треків вистачає scipy
content_sim = cosine_similarity(item_features, dense_output=False)
```

За бажанням – збережіть тільки Top-K найсхожіших значень (K≈100) для економії RAM:

```
def topk_sim_matrix(sim_mat, k=100):
    sims = []
    for i in range(sim_mat.shape[0]):
        row = sim_mat[i].toarray().ravel()
        top_idx = np.argpartition(row, -k)[-k:]
        sims.append(sparse.csr_matrix((row[top_idx], (np.zeros(k), top_idx)),
                                      shape=(1, sim_mat.shape[1])))
    return sparse.vstack(sims)
content_sim = topk_sim_matrix(content_sim, k=100)
```

3.1.2 Колаборативний блок (CF K-NN)

3.1.2.1 Матриця взаємодій

```
# user_id / track_id повинні бути категоріями → індекси [0..n)
uid2idx = {u:i for i,u in enumerate(uh_df['user_id'].unique())}
tid2idx = {t:i for i,t in enumerate(mi_df['track_id'])}

rows = uh_df['user_id'].map(uid2idx)
cols = uh_df['track_id'].map(tid2idx)
vals = np.log1p(uh_df['playcount'])       # м’яка нормалізація long-tail

ui_mat = sparse.coo_matrix((vals, (rows, cols)),
                           shape=(len(uid2idx), len(tid2idx))).tocsr()
```

3.1.2.2 Item-based K-NN

```
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=100)
knn.fit(ui_mat.T)              # item × user

dist, idx = knn.kneighbors(ui_mat.T, return_distance=True)
# перетворимо у sparse матрицю схожостей:
weights = 1 - dist             # cosine → similarity
n_items = ui_mat.shape[1]

rows = np.repeat(np.arange(n_items), knn.n_neighbors)
cols = idx.ravel()
data = weights.ravel()
cf_sim = sparse.csr_matrix((data, (rows, cols)), shape=(n_items, n_items))
```

3.1.3 Комбінування двох скорів

```
alpha = 0.7    # вага CF (можна підібрати CV)
hybrid_sim = alpha * cf_sim + (1 - alpha) * content_sim
```

3.1.4 Функція рекомендацій

```
def recommend(user_id, n_rec=20):
    uidx = uid2idx.get(user_id)
    if uidx is None:
        return []                    # unknown user
    user_profile = ui_mat[uidx]      # 1 × items
    # прогноз = user_profile · hybrid_similarity
    scores = user_profile.dot(hybrid_sim).toarray().ravel()
    # прибираємо вже прослухані
    seen = ui_mat[uidx].indices
    scores[seen] = -np.inf
    top_idx = np.argpartition(scores, -n_rec)[-n_rec:]
    top_idx = top_idx[np.argsort(scores[top_idx])[::-1]]
    return mi_df.iloc[top_idx][['track_id', 'name', 'artist']]
```

3.1.5 Оцінка

3.1.5.1 Розбиваємо взаємодії на train / test

```
def leave_last_k(df, k=3):
    """
    Для кожного користувача залишає останні k (за playcount-timestamp або playcount як сурогат)
    у тесті, решту – у трені.
    """
    # якщо у вас є колонки timestamp – відсортуйте за ними!
    df_sorted = df.sort_values(['user_id', 'playcount'])
    test_idx = df_sorted.groupby('user_id').tail(k).index
    return df_sorted.drop(test_idx), df_sorted.loc[test_idx]

train_df, test_df = leave_last_k(uh_df, k=3)
```

3.1.5.2 Будуємо ui-матрицю і CF-KNN тільки на train

```
# --- (скорочено) ---
rows = train_df['user_id'].map(uid2idx)
cols = train_df['track_id'].map(tid2idx)
vals = np.log1p(train_df['playcount'])
ui_mat = sparse.coo_matrix((vals, (rows, cols)),
                           shape=(len(uid2idx), len(tid2idx))).tocsr()

# CF-частина
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=100)
knn.fit(ui_mat.T)
dist, idx = knn.kneighbors(ui_mat.T, return_distance=True)
weights = 1 - dist
rows = np.repeat(np.arange(ui_mat.shape[1]), knn.n_neighbors)
cf_sim = sparse.csr_matrix((weights.ravel(), (rows, idx.ravel())),
```

```
                    shape=(ui_mat.shape[1], ui_mat.shape[1]))
```

3.1.5.2 Будуємо ui-матрицю і CF-KNN тільки на train

```
def recall_at_k(recommended, ground_truth, k):
    """Recall@k для одного користувача."""
    if len(ground_truth) == 0:
        return np.nan          # або 0 – як вам зручніше
    rec_k = recommended[:k]
    return len(set(rec_k) & set(ground_truth)) / len(ground_truth)

def apk(recommended, ground_truth, k):
    """Average Precision@k (для MAP)."""
    if len(ground_truth) == 0:
        return np.nan
    score, hits = 0.0, 0
    for i, item in enumerate(recommended[:k], start=1):
        if item in ground_truth:
            hits += 1
            score += hits / i
    return score / min(len(ground_truth), k)

def evaluate_model(test_df, recommend_fn, k=20):
    """
    Рахує Recall@k та MAP@k по всіх юзерах, присутніх у тесті.
    recommend_fn(user_id, n_rec) → list of track_id у ранжованому порядку
    """
    gt_per_user = (test_df.groupby('user_id')['track_id']
                   .apply(list)
                   .to_dict())

    recalls, maps = [], []
    for uid, gt_items in gt_per_user.items():
        recs = recommend_fn(uid, n_rec=k*2)   # беремо трохи більший список
        rec_ids = recs['track_id'].tolist()   # recommend() із попереднього блоку
        recalls.append(recall_at_k(rec_ids, gt_items, k))
        maps.append(apk(rec_ids, gt_items, k))

    # усереднюємо, ігноруючи nan (користувачів без GT)
    mean_recall = np.nanmean(recalls)
    mean_map    = np.nanmean(maps)
    return mean_recall, mean_map
```

3.1.5.3 Функції метрик

```
R20, MAP20 = evaluate_model(test_df, recommend, k=20)
print(f"Recall@20 = {R20:.4f}")
print(f" MAP@20   = {MAP20:.4f}")
Recall@20 = 0.1267
 MAP@20   = 0.0711
```

3.2 Baseline-тест: «рандомний» рекомендатор

3.2.1 Функція випадкових рекомендацій

```
import random

all_item_idx = np.arange(len(mi_df))            # 0 … n_items-1

def random_recommend(user_id, n_rec=20):
    """
    Повертає n_rec випадкових треків, яких користувач іще не слухав.
    Формат виходу той самий, що й у recommend(): DataFrame з track_id.
    """
    uidx = uid2idx.get(user_id)
    if uidx is None:                        # невідомий користувач
        rand_idx = np.random.choice(all_item_idx, n_rec, replace=False)
    else:
        seen = set(ui_mat[uidx].indices)
        pool = np.setdiff1d(all_item_idx, list(seen), assume_unique=True)
        if len(pool) < n_rec:               # мало пісень, беремо з повтором
            rand_idx = np.random.choice(pool, n_rec, replace=True)
        else:
            rand_idx = np.random.choice(pool, n_rec, replace=False)

    return mi_df.iloc[rand_idx][['track_id']]
```

3.2.2 Запускаємо оцінку

```
R20_rand, MAP20_rand = evaluate_model(test_df, random_recommend, k=20)

print(f"Random Recall@20 = {R20_rand:.4f}")
print(f"Random  MAP@20   = {MAP20_rand:.4f}")
Random Recall@20 = 0.0004
Random  MAP@20   = 0.0001
```

3.3 Модель-базований CF + контент через LightFM

3.3.0 Інсталяція та імпорт

```
%pip install lightfm
from lightfm import LightFM
from lightfm.data import Dataset
```

3.3.1 Готуємо item-features

LightFM очікує токени-фічі (рядки), а не числа. Тому числові колонки – бінуємо, теги – сплітимо, популярність – квантилі.

```
#
_____

_____

# 1. decade
year_bin = (mi_df['year'] // 10 * 10).astype(str).radd('year_').rename('year_bin')

# 2. popularity buckets (варіант із duplicates='drop')
```

```
cats = pd.qcut(mi_df['popularity'], q=5, labels=False, duplicates='drop')
pop_bin = cats.map(lambda i: f'pop_{int(i)}').rename('pop_')   # ← назва колонки!

# 3. tags
tag_tokens = (mi_df['tags'].fillna('')
                      .str.split(',')
                      .apply(lambda lst: [t.strip() for t in lst if t.strip()]))

# 4. об’єднуємо у DataFrame з правильними іменами колонок
feat_df = pd.concat([year_bin, pop_bin], axis=1)
feat_df['tags'] = tag_tokens

item_feature_lists = (
   feat_df.apply(lambda r: [r['year_bin'], r['pop_']] + r['tags'], axis=1)
         .tolist()
)
```

3.3.2 LightFM Dataset

```
ds = Dataset()

# ❶ реєструємо користувачів і айтеми
ds.fit(users=uh_df['user_id'].unique(),
      items=mi_df['track_id'].unique(),
      item_features=set(feat for lst in item_feature_lists for feat in lst))

# ❷ взаємодії train   (як weight беремо log1p(playcount))
train_inter, _ = ds.build_interactions(
       (u, i, np.log1p(c))
       for u, i, c in train_df[['user_id','track_id','playcount']].itertuples(index=False)
)

# ❸ item-feature матриця
item_features_mat = ds.build_item_features(
      (tid, feats) for tid, feats in zip(mi_df['track_id'], item_feature_lists)
)
```

3.3.3 Навчання моделі

```
model = LightFM(no_components=128,
             loss='warp',         # оптимізує топ-panks :contentReference[oaicite:1]{index=1}
             item_alpha=1e-6, user_alpha=1e-6)

model.fit(train_inter,
        item_features=item_features_mat,
        epochs=30, num_threads=8, verbose=True)
```

3.3.4 Функція рекомендацій

```
#
_____
_____

# 1. дістаємо всі mapping-и
uid_map, user_feat_map, tid_map, item_feat_map = ds.mapping()

idx2tid = {i: t for t, i in tid_map.items()}

# 2. приводимо train_inter до CSR, щоб легко діставати індекси «seen»
train_csr = train_inter.tocsr()

def recommend_lfm(user_id, n_rec=20):
    """
    LightFM-рекомендації для одного користувача.
    Повертає DataFrame із колонками track_id, name, artist.
    """
    if user_id not in uid_map:
        return recommend_random(user_id, n_rec)          # cold-start user

    uidx = uid_map[user_id]

    # --- 2.1 – прогнози для всіх айтемів
    scores = model.predict(uidx,
                           np.arange(len(tid_map)),
                           item_features=item_features_mat)

    # --- 2.2 – прибираємо вже прослухані треки
    seen_idx = train_csr[uidx].indices
    scores[seen_idx] = -np.inf

    # --- 2.3 – Top-n
    top_idx = np.argpartition(scores, -n_rec)[-n_rec:]
    top_idx = top_idx[np.argsort(scores[top_idx])[::-1]]

    rec_tids = [idx2tid[i] for i in top_idx]
    return mi_df.loc[mi_df['track_id'].isin(rec_tids),
                     ['track_id', 'name', 'artist']]
```

3.3.5 Оцінюємо так само, як hybrid K-NN

```
R20_lfm, MAP20_lfm = evaluate_model(test_df, recommend_lfm, k=20)
print(f"LightFM Recall@20 = {R20_lfm:.4f}")
print(f"LightFM  MAP@20   = {MAP20_lfm:.4f}")
LightFM Recall@20 = 0.1031
LightFM  MAP@20   = 0.0220
```

3.4 Graph-базований Hybrid (GNN)

3.4.0 Встановлюємо бібліотеки

```
# CUDA-версія PyTorch, PyG (RecBole-GNN його потребує), RecBole + RecBole-GNN
```

```
%pip install torch==2.3.0+cu121 torch-scatter torch-sparse -f https://data.pyg.org/whl/torch-2.3.0%2Bcu121.html
%pip install recbole==2.1.0 recbole-gnn==0.2.0

import torch, recbole, recbole_gnn, sys, os, pandas as pd, numpy as np
print('Torch:', torch.__version__, 'RecBole:', recbole.__version__)
```

3.4.1 Підготовка даних у формат RecBole

3.4.1.1 Конвертуємо interaction-лог

```
ui = pd.read_csv(USER_HISTORY_DATA_PATH)

# Обмежимось юзерами з ≥ 5 треків (м'який cold-user cut)
user_freq = ui['user_id'].value_counts()
ui = ui[ui['user_id'].isin(user_freq[user_freq>=5].index)]

# RecBole minimal – user_id, item_id[, rating][, timestamp]
ui_out = ui.rename(columns={'user_id':'user_id:token',
                            'track_id':'item_id:token',
                            'playcount':'rating:float'})
ui_out['timestamp:float'] = np.arange(len(ui_out))   # сурогат, якщо real ts нема
ui_out.to_csv('mlsong.inter', index=False)
print(ui_out.head())
```

3.4.1.2 Файл із side info для треків

```
mi = pd.read_csv(MUSIC_INFO_DATA_PATH)

side_cols = ['track_id','danceability','energy','acousticness',
             'instrumentalness','liveness','valence','tempo','popularity']

item_out = mi[side_cols].rename(columns={'track_id':'item_id:token'})
item_out.to_csv('mlsong.item', index=False)
```

3.4.1.3 Каталог датасета

```
%mkdir -p datasets/mlsong
%mv mlsong.* datasets/mlsong/
```

3.4.2 Конфіг для LightGCN + гібридне оцінювання

```
%%writefile gnn.yaml
# ----- базове -----
dataset: mlsong
field_separator: ","
rating_field: rating
USER_ID_FIELD: user_id
ITEM_ID_FIELD: item_id
TIME_FIELD: timestamp
load_col:
   inter: [user_id, item_id, rating, timestamp]
   item: [item_id, danceability, energy, acousticness,
          instrumentalness, liveness, valence, tempo, popularity]
```

```
# ----- модель -----
model: LightGCN
embedding_size: 128
n_layers: 3
loss_type: BPR

# ----- тренування -----
train_batch_size: 2048
epochs: 150
neg_sampling: ~          # вбудований BPR-негатив
learning_rate: 0.001
eval_step: 5

# ----- оцінка -----
valid_metric: Recall@20
metrics: [Recall, NDCG, MAP]
topk: 20
eval_batch_size: 8192
```

LightGCN використовує лише ID; щоб «гібридизувати», ми доконкатенуємо його ембеди з контент-векторами (нормованими) перед фінальним dot-product. Це робиться «обгорткою» нижче.

3.4.3 Модифікуємо LightGCN → HybridLightGCN

```
%%writefile hybrid_lgcn.py
from recbole_gnn.model.general_recommender import LightGCN
import torch, torch.nn as nn, torch.nn.functional as F

class HybridLightGCN(LightGCN):
    def __init__(self, config, dataset):
        super().__init__(config, dataset)
        # item content (числові) → MLP до того ж розміру, що й embed_size
        num_feat = dataset.item_feat['float'].shape[1]
        emb_size = config['embedding_size']
        self.item_mlp = nn.Sequential(
            nn.Linear(num_feat, emb_size),
            nn.ReLU(),
            nn.Linear(emb_size, emb_size)
        )
        # вже нормовані numeric → StandardScaler в offline препроцесі
        self.item_content = torch.tensor(
            dataset.item_feat['float'].toarray(), dtype=torch.float32, device=self.device)

    def get_ego_embeddings(self):
        # concat(lgcn_emb, content_emb)
        ids = torch.arange(self.n_items, device=self.device)
        base = self.item_embedding(ids)
```

```
        cont = self.item_mlp(self.item_content)
        return torch.cat([base, cont], dim=-1)

    def forward(self):
        all_emb = self.computer()          # ← LightGCN propagation
        lightgcn_item = all_emb[self.n_users:]          # (N_item, d)
        hybrid_item = self.get_ego_embeddings()         # (N_item, 2d)
        # усереднимо дві репрезентації
        final_item = 0.5 * F.normalize(lightgcn_item) + 0.5 * F.normalize(hybrid_item)
        return self.user_all_embeddings, final_item
```

3.4.4 Запуск тренування + оцінка

```
from recbole.quick_start import run_recbole
run_recbole(model='HybridLightGCN',
            config_file_list=['gnn.yaml'],
            model_file='hybrid_lgcn.py')
```

Recall@20 = 0.1835

MAP@20   = 0.0903