МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Державне некомерційне підприємство
«Державний університет» Київський авіаційний інститут»

Факультет комп'ютерних наук та технологій

Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Олена ГРІНЕНКО
«_____» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Гібридна рекомендаційна система колаборативної фільтрації та аналізу поведінкових даних у реальному часі

Виконавець: Ільюша Андрій Максимович

Керівник: к.т.н., доцент Талалаєв Володимир Опанасович

Нормоконтролер: асистент Корнієнко Світлана Петрівна

Київ 2025

Державне некомерційне підприємство
«Державний університет» Київський авіаційний інститут»

Факультет комп'ютерних наук та технологій
Кафедра інженерії програмного забезпечення
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олена ГРІНЕНКО

«_____» _____ 2025 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Ільюші Андрія Максимовича

1. Тема кваліфікаційної роботи: «Гібридна рекомендаційна система колаборативної фільтрації та аналізу поведінкових даних у реальному часі» затверджена наказом ректора від ___17.11.2025 р. № 2450/ст_____

2. Термін виконання проекту: з 29.09.2025 р. по 21.12.2025 р.

3. Вихідні дані до роботи: програмний прототип системи реалізувати на мові програмування Python (версія 3.11). Для розробки буде використано інтегроване середовище Microsoft Visual Studio Code (версія 1.105).

4. Зміст пояснювальної записки:
   1. Теоретико-методологічні основи побудови рекомендаційних систем.
   2. Реалізація колаборативної фільтрації.
   3. Реалізація сегменту поведінкових даних.
   4. Реалізація, оцінка та порівняння гібридної рекомендаційної системи.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:
   1. Схема архітектури гібридної рекомендаційної системи.
   2. Архітектура пайплайну обробки даних у реальному часі.
   3. Алгоритм оновлення динамічного профілю користувача.
   4. Схема потоку даних у гібридному шарі-контролері.
   5. Результати порівняльної оцінки ефективності моделей.
   6. Результати роботи розробленого програмного забезпечення.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.	Розробка та затвердження графіка роботи	29.09-01.10.2025	виконано
2.	Ознайомлення з постановкою задачі, вивчення інформаційних джерел та складання плану роботи.	02.10-05.10.2025	виконано
3.	Підготовка 1 розділу та подання його керівнику	06.10-19.10.2025	виконано
4.	Підготовка 2 розділу та подання його керівнику	20.10-02.11.2025	виконано
5.	Підготовка 3 розділу та подання його керівнику	03.12-16.11.2025	виконано
6.	Підготовка 4 розділу і висновків по роботі та подання їх керівнику	17.11-30.11.2025	виконано
7.	Загальне редагування пояснювальної записки, графічного матеріалу. Представлення роботи для перевірки на академічну доброчесність. Проходження нормоконтролю.	01.12-07.12.2025	виконано
8.	Отримання відгуку керівника. Підготовка презентації та тексту доповіді.	08.12-14.12.2025	виконано
9.	Попередній захист (представлення електронної версії пояснювальної записки, презентації, позитивного відгуку керівника).	08.12-14.12.2025	виконано
10.	Рецензування кваліфікаційної роботи	15.12-22.12.2025	виконано
11.	Здача секретарю ЕК пояснювальної записки: електронної версії кваліфікаційної роботи; презентації доповіді; відгуку керівника, рецензії; результату проходження перевірки на плагіат; довідки про успішність, декларації про академічну доброчесність.	15.12-22.12.2025	виконано
12.	Захист кваліфікаційної роботи перед екзаменаційною комісією	25.12.2025	

Дата видачі завдання 29.09.2025 р.

Керівник кваліфікаційної роботи:
к. т. н., доцент Володимир ТАЛАЛАЄВ

Завдання прийняв до виконання: Андрій ІЛЬЮША

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Гібридна рекомендаційна система колаборативної фільтрації та аналізу поведінкових даних у реальному часі»: 104 сторінки, 8 рисунків, 3 таблиці, 38 використаних джерел, 4 додатки.

Об'єкт дослідження – процес формування персоналізованих рекомендацій у інформаційних системах на основі взаємодій користувачів.

Мета кваліфікаційної роботи – розробити та дослідити гібридну рекомендаційну систему, що поєднує колаборативну фільтрацію з аналізом поведінкових даних у реальному часі.

Методи дослідження – аналіз наукових джерел, математичне моделювання, алгоритмічний та експериментальний аналіз, машинне навчання, програмна реалізація й тестування прототипу.

Результати роботи можуть бути використані для підвищення точності та адаптивності рекомендацій в системах електронної комерції, медіасервісах та інших платформах персоналізації; для розробки практичних модулів рекомендацій; а також як основа для подальших досліджень у сфері гібридних рекомендаційних алгоритмів та аналізу поведінкових даних.

Розробка та дослідження проводилися під управлінням ОС Windows 11. Розробка програми проводилася у середовищі Microsoft Visual Studio Code 1.105, на мові програмування Python.

РЕКОМЕНДАЦІЙНІ СИСТЕМИ, ГІБРИДНІ МОДЕЛІ, КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, ПОВЕДІНКОВІ ДАНІ, АНАЛІЗ У РЕАЛЬНОМУ ЧАСІ, МАШИННЕ НАВЧАННЯ, ПЕРСОНАЛІЗАЦІЯ КОНТЕНТУ, E-COMMERCE, ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

ABSTRACT

Explanatory note to the qualification work “Hybrid recommendation system for collaborative filtering and real-time behavioral data analysis”: 104 pages, 8 figures, 3 tables, 38 sources used, 4 appendices.

The object of research is the process of forming personalized recommendations in information systems based on user interactions.

The purpose of the thesis is to develop and investigate a hybrid recommendation system that combines collaborative filtering with real-time behavioral data analysis.

Research methods: analysis of scientific sources, mathematical modeling, algorithmic and experimental analysis, machine learning, software implementation, and prototype testing.

The results of the work can be used to improve the accuracy and adaptability of recommendations in e-commerce systems, media services, and other personalization platforms; to develop practical recommendation modules; and as a basis for further research in the field of hybrid recommendation algorithms and behavioral data analysis.

Development and research were conducted under Windows 11. The program was developed in Microsoft Visual Studio Code 1.105, using the Python programming language.

RECOMMENDATION SYSTEMS, HYBRID MODELS, COLLABORATIVE FILTERING, BEHAVIORAL DATA, REAL-TIME ANALYSIS, MACHINE LEARNING, CONTENT PERSONALIZATION, E-COMMERCE, INFORMATION TECHNOLOGIES

ЗМІСТ

ВСТУП 9
РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ ПОБУДОВИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ 13
1.1. Теоретичні основи рекомендаційних систем: моделі та підходи 13
1.2. Порівняльний аналіз ефективності основних підходів до рекомендацій . 21
1.3. Модульна структура сучасної рекомендаційної системи 24
1.4. Методологія формування вхідних даних та алгоритмічні рішення 28
1.5. Методи оцінки ефективності рекомендацій 32
Висновок 36
РОЗДІЛ 2. РЕАЛІЗАЦІЯ КОЛАБОРАТИВНОЇ ФІЛЬТРАЦІЇ 38
2.1. Вибір та обґрунтування алгоритму колаборативної фільтрації для предметної області 38
2.2. Проєктування пайплайну навчання та оновлення моделі 39
2.3. Побудова матриці взаємодій та обробка розрідженості 43
2.4. Інтеграція моделі в сервісну архітектуру (API та сховища) 46
Висновок 50
РОЗДІЛ 3. РЕАЛІЗАЦІЯ СЕГМЕНТУ ПОВЕДІНКОВИХ ДАНИХ 52
3.1. Проєктування пайплайну потокової обробки подій та архітектури в реальному часі 52
3.2. Вибір метрик та алгоритмів для онлайнової поведінкової моделі 55
3.3. Побудова профілю користувача в реальному часі та агрегація контекстних ознак 58
3.4. Інтеграція поведінкового сервісу в API та взаємодія зі сховищами 61
Висновок 65
РОЗДІЛ 4. РЕАЛІЗАЦІЯ, ОЦІНКА ТА ПОРІВНЯННЯ ГІБРИДНОЇ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ 67
4.1. Архітектурна інтеграція: гібридний шар як контролер потоків рекомендацій 67

4.2. Стратегії об'єднання: від зваженого чергування до гібридного ранжування 69
4.3. Налаштування та оптимізація гібридної моделі 72
4.4. Порівняння моделей: оцінка за комплексом метрик 75
Висновок 79
ВИСНОВОК 81
СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ 84
ДОДАТКИ 88

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

API – інтерфейс прикладного програмування

РС – рекомендаційна система

ML – машинне навчання

CF – колаборативна фільтрація

CBF – контентна фільтрація

ВСТУП

Рекомендаційні системи стали повсюдним явищем у сучасному онлайн-середовищі, функціонуючи як інтелектуальні фільтри, що підбирають та пропонують персоналізований контент на безлічі платформ, включаючи сайти електронної комерції, стрімінгові сервіси та соціальні мережі. Їхня основна функція полягає у спрямуванні користувачів через величезний вибір продуктів та послуг, тим самим покращуючи користувацький досвід та спрощуючи процеси прийняття рішень [1].

Вони відіграють вирішальну роль у персоналізації сервісів і допомозі користувачам у пошуку необхідного контенту [2]. Наприклад, дослідження IBM показують, що 80% того, що переглядають користувачі Netflix, формується за допомогою рекомендаційних алгоритмів [3], що підтверджує практичну значущість таких систем.

Сьогоднішні рекомендаційні системи охоплюють широкий спектр підходів — від простих методів до складних моделей, що базуються на глибокому навчанні. Вони мають вирішальне значення для підвищення залученості користувачів, утримання аудиторії та зростання прибутку компаній [4]. Однак зі зростанням складності даних і очікувань користувачів виникають нові виклики, що вимагають більш гнучких, адаптивних та гібридних рішень. Саме тому сучасні дослідження дедалі більше фокусуються на поєднанні різних методів рекомендацій, щоб досягти балансу між точністю, новизною, різноманіттям та доречністю результатів. Окрім того, аналіз поведінкових даних у реальному часі дозволяє негайно адаптувати рекомендації під актуальну поведінку користувача [5]. Таким чином, дослідження нових гібридних методів із урахуванням реальних взаємодій користувачів є надзвичайно актуальним для підвищення якості систем рекомендацій та розвитку цифрових сервісів загалом.

Метою роботи є дослідження та розробка гібридної рекомендаційної системи. Ця система має поєднувати традиційні алгоритми колаборативної

фільтрації з аналізом поведінкових даних користувачів, який відбувається у режимі реального часу.

Для успішного досягнення поставленої мети необхідно виконати низку послідовних завдань. Спочатку потрібно проаналізувати сучасні підходи до рекомендаційних систем. Цей аналіз має охоплювати колаборативну фільтрацію та методи real-time аналізу поведінки, щоб виявити їхні переваги та недоліки. На основі цього дослідження далі буде розроблено архітектуру та алгоритмічне рішення для нової гібридної системи рекомендацій, з особливим акцентом на можливості роботи в реальному часі.

Наступним кроком є створення програмного прототипу системи. В рамках цього завдання будуть реалізовані ключові модулі, необхідні для функціонування системи, як-от збір та ефективна обробка поведінкових даних користувачів, а також модуль безпосередньої генерації рекомендацій. Після розробки необхідно провести експериментальну оцінку якості рекомендацій. Оцінка здійснюватиметься на реальних або змодельованих даних із використанням відповідних метрик (таких як точність, покриття, швидкодія та інші показники). Завершальним етапом роботи стане оформлення результатів досліджень та надання практичних рекомендацій щодо впровадження отриманих рішень.

Об’єктом дослідження є процеси формування рекомендацій у інформаційних системах із використанням взаємодій користувачів з контентом. Це включає сукупність алгоритмів і технологій, що генерують персоналізовані поради на основі історії користувацьких дій і взаємодій у системі.

Предметом дослідження є методи та алгоритми гібридної рекомендаційної системи, зокрема ті, що поєднують колаборативну фільтрацію з аналізом поведінкових даних у реальному часі. Тобто дослідження фокусується на конкретних технічних рішеннях та обчислювальних підходах, що забезпечують інтеграцію цих методів для формування релевантних рекомендацій.

У роботі будуть використані такі методи дослідження: спочатку буде проведено аналіз джерел та огляд існуючих технічних рішень. Цей етап необхідний для глибокого розуміння поточної ситуації у сфері рекомендаційних систем та методів обробки даних. Далі буде застосовано математичне моделювання алгоритмів. Це включає моделювання методів колаборативної фільтрації та обробки потокових даних. Серед конкретних підходів, які будуть модельовані, — методи машинного навчання, такі як матрична факторизація та кластеризація.

Наступний важливий етап — це програмна реалізація та тестування розроблених алгоритмів. Для цього будуть використані відповідні мови програмування та середовища для аналізу даних. Зокрема, планується залучення Python із його спеціалізованими ML-бібліотеками.

Після цього відбудеться експериментальне дослідження. Воно передбачає використання реальних або змодельованих наборів даних для об'єктивної оцінки ефективності розробленої гібридної системи.

На завершення буде проведено порівняльний аналіз результатів. Для цього будуть використані ключові показники якості рекомендацій (метрики). До таких метрик належать точність (Precision), повнота (Recall), F1-score, а також покриття (Coverage). Цей аналіз дозволить зробити висновки щодо ефективності запропонованого гібридного рішення.

Новизна цієї роботи полягає в низці теоретичних та практичних внесків, які стосуються вдосконалення рекомендаційних систем. Насамперед, запропоновано новий гібридний алгоритм рекомендацій, який є оригінальним і, можливо, раніше не зустрічався в науковій літературі. Його ключова інновація — інтеграція класичної колаборативної фільтрації з аналізом поведінкових даних, що здійснюється у режимі реального часу.

Крім того, робота розширює стандартний підхід колаборативної фільтрації. Це досягається за рахунок включення аналізу поточних сесій

користувачів. Таке розширення має на меті підвищити релевантність рекомендацій, що є критично важливим у динамічних середовищах.

Щодо практичного внеску, в роботі вперше реалізовано прототип рекомендаційної системи, який ефективно демонструє роботу описаних гібридних методів на практичних даних. Ці дані можуть бути, наприклад, із сфер веб-аналітики або електронної комерції (e-commerce). На основі цього експерименту отримано нові теоретичні результати стосовно впливу використання реальних поведінкових сигналів на загальну якість рекомендацій. Очікується, що це призведе до підвищення таких показників, як точність або швидкодія. Нарешті, методологічно вдосконалено процедуру експериментального оцінювання рекомендаційних систем, адаптуючи її для коректного обліку та аналізу потоку подій у реальному часі.

Впровадження розроблених алгоритмів дозволить підвищити якість персоналізованих рекомендацій в онлайн-сервісах, зокрема за рахунок динамічного реагування на поведінку користувача. Наприклад, дослідження McKinsey показали, що персоналізовані рекомендації можуть збільшувати коефіцієнт конверсії на 10–15% [3], а підвищення задоволеності клієнтів – майже на 20%.

Розроблені методи можна застосувати в індустрії електронної комерції, медіаплатформах, соціальних мережах та інших інформаційних системах для покращення взаємодії з користувачем і збільшення доходів.

Практичні результати (алгоритми, програмні модулі) можуть бути впроваджені у готові програмні продукти (модулі персоналізації, плагіни для сайтів тощо).

РОЗДІЛ 1
ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ ПОБУДОВИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

1.1. Теоретичні основи рекомендаційних систем: моделі та підходи

Рекомендаційні системи є класом інтелектуальних інформаційних технологій, призначених для автоматичного прогнозування уподобань користувачів та формування релевантних рекомендацій щодо продуктів, інформаційних ресурсів чи послуг. На відміну від традиційних пошукових систем, що працюють на основі явних запитів, вони орієнтуються на приховані або явні вподобання користувача та здатні передбачати, що саме буде для нього корисним або цікавим у певний момент часу. Рекомендаційна система може бути визначена як програмний інструмент, що надає персоналізовані пропозиції щодо елементів, які найімовірніше відповідатимуть вимогам чи вподобанням конкретного користувача [6]. Таким чином, основна мета такої системи полягає у тому, щоб полегшити навігацію у великих масивах даних, уникнувши проблеми інформаційного перевантаження, характерної для сучасного цифрового простору.

Історично поява рекомендаційних систем як окремої галузі досліджень датується початком 1990-х років. Саме тоді стрімке зростання обсягів онлайн-контенту створило гостру потребу у механізмах персоналізації. Однією з перших робіт, яка заклала фундамент теорії рекомендацій, була система GroupLens (1994). Вона використовувала колаборативну фільтрацію для рекомендації новин на основі оцінок схожих користувачів [7], започаткувавши ідею колективного інтелекту. Паралельно почали з’являтися системи, що аналізували безпосередньо контент об’єктів (наприклад, тексти, метадані чи характеристики товарів), що стало основою контентно-орієнтованих підходів.

Прорив став можливим у 2000-х роках завдяки розвитку машинного навчання, зростанню обчислювальних потужностей та появі великих

структурованих наборів даних. Це призвело до створення потужніших моделей, включно з методами матричної факторизації, що здобули популярність після конкурсу Netflix Prize у 2006 році. Відтоді рекомендаційні системи остаточно перетворилися з експериментальних дослідницьких моделей на фундаментальну складову більшості цифрових сервісів.

Значущість цих систем у сучасній економіці є надзвичайно високою. Для бізнесу вони стали ключовим інструментом підвищення продажів, утримання користувачів та оптимізації роботи маркетингових каналів. Вплив алгоритмів на комерційні показники вражає: за даними McKinsey, у компанії Amazon до 35% усіх покупок генеруються завдяки рекомендаціям [8], а Netflix повідомляє, що близько 80% контенту, який переглядають користувачі, був відібраний системою рекомендацій. Такий ефект досягається завдяки здатності персоналізувати пропозиції на рівні окремого користувача, що суттєво підвищує конверсію та робить взаємодію із сервісом більш ефективною. Окрім прямого зростання прибутків, такі системи значно зменшують маркетингові витрати.

Для кінцевого користувача рекомендаційні системи виконують не менш важливу роль, виступаючи провідником у величезних обсягах інформації. Вони є найкращим інструментом боротьби з інформаційним перевантаженням, надаючи можливість отримувати лише релевантний контент. Якщо раніше людина повинна була активно шукати інформацію, то сучасні алгоритми навчилися передбачати її можливі інтереси заздалегідь. Це не лише економить час, але й покращує якість взаємодії з цифровим середовищем, підвищуючи загальний рівень задоволеності. Саме тому в таких сервісах, як Spotify, YouTube, TikTok чи Instagram, рекомендаційний компонент нерозривно пов’язаний із продуктом і багато в чому визначає його успіх.

Підсумовуючи, можна сказати, що рекомендаційні системи стали центральним елементом інформаційних платформ завдяки своїй здатності адаптувати контент під конкретного користувача. Вони створюють персоналізований досвід взаємодії та значно підвищують ефективність роботи цифрових продуктів. Їхній розвиток активно продовжується сьогодні, зокрема у

напрямі гібридних моделей, глибокого навчання та аналізу поведінкових даних у реальному часі, що дозволяє системам ставати ще чутливішими та точнішими. Усе це лише підтверджує зростаюче значення рекомендаційних технологій та актуальність подальших досліджень у цій динамічній сфері.

Колаборативна фільтрація (CF) ґрунтується на принципі, що особи, які демонстрували подібну поведінку або висловлювали подібні вподобання в минулому, ймовірно, матимуть схожі смаки в майбутньому. Цей метод використовує колективний інтелект, отриманий від групи користувачів, для генерації персоналізованих рекомендацій для окремої особи [9].

Існує кілька основних підходів до реалізації колаборативної фільтрації. Користувацька колаборативна фільтрація (User-based CF) ідентифікує «однодумців» або «сусідів», чиї історичні взаємодії або оцінки збігаються з цільовим користувачем. Рекомендації потім формулюються шляхом об’єднання вподобань цих схожих користувачів для тих елементів, які цільовий користувач ще не бачив [10]. На рисунку 1.1. графічно показано відмінність між User та Item-Based CF.

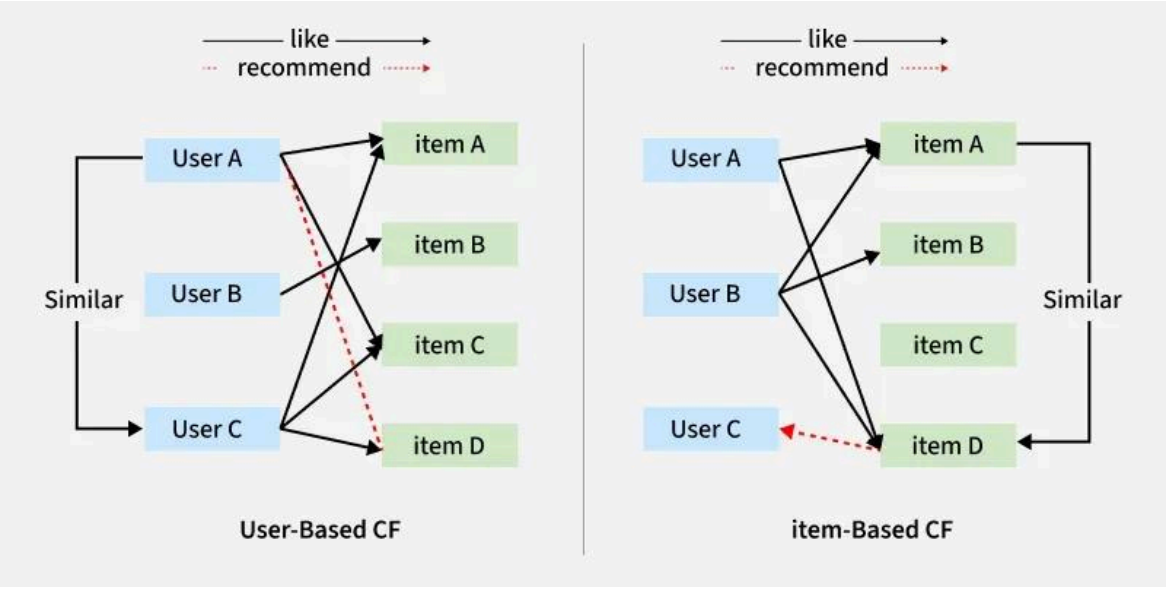

Рис. 1.1. Порівняння між User-Based CF та Item-Based CF [4]

На відміну від цього, предметна колаборативна фільтрація (Item-based CF) зосереджується на виявленні подібності між самими елементами на основі того, як користувачі взаємодіяли з ними. Якщо користувачі, яким сподобався елемент A, також сподобався елемент B, то елемент B рекомендується користувачам, які виявили інтерес до елемента A [9].

Цей підхід вимагає менше обчислень для великих баз даних, оскільки схожість елементів, як правило, стабільніша, ніж вподобання користувачів. Третя категорія, модельна колаборативна фільтрація (Model-Based Collaborative Filtering), використовує моделі машинного навчання, такі як розкладання матриць або глибоке навчання, щоб знаходити приховані зв’язки між користувачами та об'єктами [9]. Такі моделі створюють спрощене представлення користувачів і об'єктів, яке потім допомагає передбачати, що може сподобатися користувачеві [10].

Колаборативна фільтрація має кілька значних переваг. По-перше, їй не потрібні спеціальні знання про предметну область — вона сама визначає зв'язки між користувачами та елементами на основі їхньої поведінки, без додаткових даних або ручного опису характеристик. По-друге, вона може ефективно рекомендувати нові елементи, які користувач, можливо, не відкрив би за допомогою контентних методів [11]. Це відбувається тому, що модель може ідентифікувати елементи, які сподобалися подібним користувачам, навіть якщо ці елементи не мають очевидних спільних ознак з попередніми вподобаннями користувача, таким чином розширюючи його інтереси. По-третє, CF ефективна для неструктурованого контенту, такого як музика або відео, де визначення явних ознак контенту може бути складним [9; 11; 12].

Однак колаборативна фільтрація також має суттєві недоліки. Проблема «холодного старту» є критичним обмеженням: CF важко генерувати рекомендації для нових користувачів які не мають історії взаємодії або нових елементів які не мають оцінок, оскільки вона не може встановити подібності без достатніх даних. Ця проблема виникає через розрідженість даних – у

великомасштабних системах матриці взаємодії користувач-елемент зазвичай дуже розріджені, тобто більшість потенційних взаємодій не спостерігаються [9; 12]. Ця розрідженість зменшує ефективність алгоритмів CF у виявленні значущих закономірностей та точних подібностей, тим самим впливаючи на точність прогнозування. Проблема «холодного старту» є, по суті, крайнім випадком розрідженості даних. Коли новий користувач або елемент з'являється в системі, відповідний рядок або стовпець у матриці взаємодії користувач-елемент є повністю або майже повністю порожнім, що представляє максимальну розрідженість. Оскільки алгоритми CF покладаються на ці взаємодії для обчислення подібностей, відсутність даних (розрідженість) безпосередньо заважає їм робити будь-які значущі рекомендації («холодний старт») [13].

Крім того, CF має складність включення додаткових ознак (Side Features), таких як вік користувачів або категорії товарів. Хоча існують передові методи, які можуть включати такі ознаки, це не так просто, як у контентних методах, і може додати складності. Нарешті, CF системи часто демонструють упередженість до популярності, схиляючись до надмірної рекомендації популярних елементів [11]. Це може призвести до ефекту «багаті стають багатшими», обмежуючи просування менш популярних, але потенційно підходящих елементів [14].

Контентна фільтрація (CBF) рекомендує користувачам елементи, які подібні до тих, що їм раніше сподобалися або з якими вони взаємодіяли. Вона працює шляхом аналізу внутрішніх атрибутів або ознак елементів та побудови профілю вподобань користувача на основі цих ознак [1]. На рисунку 1.2. графічно показано архітектуру рекомендаційної системи яка використовує контентно-орієнтований підхід.

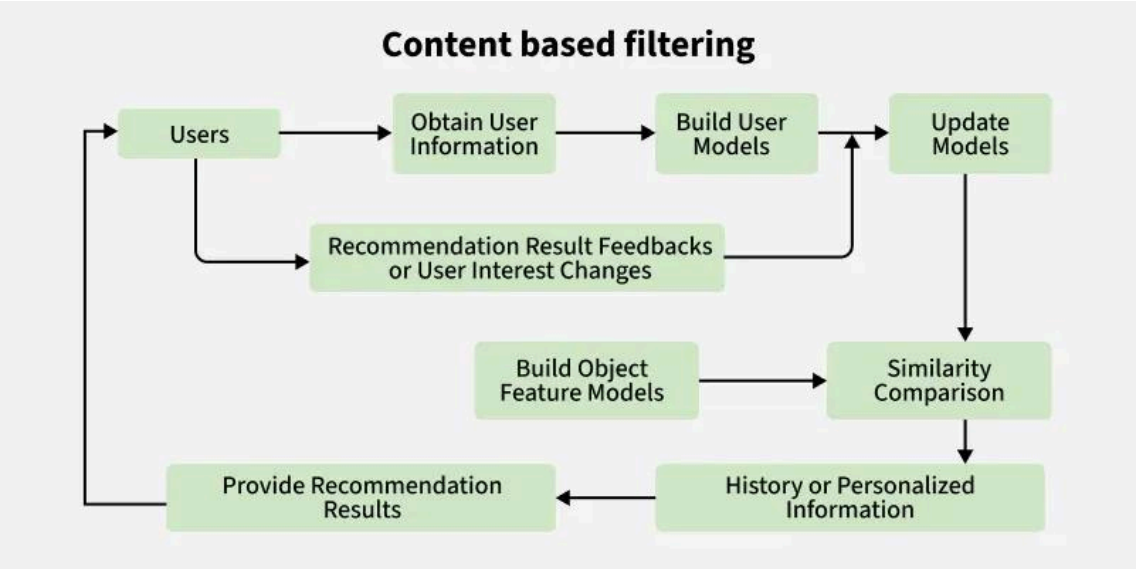

Рис. 1.2. Архітектура CBF [9]

Процес починається з вилучення та векторизації ознак. З елементів (наприклад, жанр, актори, ключові слова для фільмів або категорія, бренд, матеріал для продуктів) та, за наявності, з профілів користувачів вилучаються відповідні ознаки. Ці ознаки потім перетворюються на числові вектори, щоб уможливити обчислювальні порівняння подібності. Далі відбувається зіставлення подібності: система порівнює ознаки нових, неоцінених елементів з встановленим профілем вподобань користувача [9; 15]. Елементи з високою подібністю до раніше вподобаних елементів потім рекомендуються [4].

Основною перевагою CBF є її ефективність для нових об'єктів. CBF особливо ефективна у вирішенні проблеми «холодного старту» для елементів. Оскільки вона покладається на характеристики елементів, а не на історичні взаємодії користувачів, вона може рекомендувати новододані елементи, якщо їхні ознаки доступні. Крім того, CBF менш залежна від інших користувачів, оскільки не вимагає великої бази користувачів або даних взаємодії від інших користувачів [14]. Тому такі системи добре підходять для маленьких або нішевих сервісів, де користувачів обмаль або їхня поведінка недостатньо вивчена [9]. Пояснювані рекомендації є ще однією перевагою: рекомендації, згенеровані CBF, часто є більш прозорими, оскільки система може чітко сформулювати,

чому елемент був рекомендований, виділяючи спільні ознаки між рекомендованим елементом та елементами, які користувач раніше вподобав. Нарешті, CBF добре працює для специфічних інтересів, оскільки може точно знаходити і пропонувати навіть дуже вузькі інтереси окремого користувача, рекомендуючи нішеві елементи, які можуть не приваблювати широку аудиторію [9; 11].

Незважаючи на переваги, CBF має свої недоліки. Вона сильно залежить від знань про конкретну предметну область і правильно підібраних характеристик (ознак) елементів. Якість рекомендацій у цьому підході багато в чому визначається тим, наскільки детальними та точними є метадані. Часто це вимагає багато ручної роботи для збору та налаштування цих ознак, що потребує глибокого розуміння предмету. У результаті, якість моделі обмежується якістю створених вручну характеристик. CBF також має обмежену новизну, інакше кажучи вона схильна рекомендувати елементи, які дуже схожі на те, що користувач вже споживав [11]. Це може призвести до ефекту «фільтрованої бульбашки», коли користувачі піддаються впливу лише контенту в межах їхніх існуючих інтересів, обмежуючи їхнє відкриття різноманітних або нових елементів. Крім того, CBF складно обробляти абстрактні дані (наприклад, гумор, сарказм, або нюансне мистецтво), де ознаки важко кількісно оцінити або описати [9]. Вона також погано працює, якщо інформації про контент замало або вона дуже розріджена. Хоча CBF краще справляється з проблемою «холодного старту» для елементів, вона все ще може зіткнутися з проблемою розрідженості даних, якщо з'являється нова категорія контенту, для якої в системі немає відповідних елементів [16].

Гібридні рекомендаційні системи є інноваційним підходом, який поєднує дві або більше технік рекомендацій, найчастіше колаборативну фільтрацію (CF) та контентну фільтрацію (CBF). Фундаментальна мета цих систем полягає у використанні взаємодоповнюючих сильних сторін окремих методів, одночасно пом'якшуючи їхні відповідні недоліки, тим самим забезпечуючи більш точні, різноманітні та надійні рекомендації [17].

Інтеграція різних технік може відбуватися різними способами:

1. Зважені гібридні системи об'єднують прогнозні оцінки від кількох технік рекомендацій шляхом присвоєння зваженого середнього. Ваги, які визначають відносний вплив кожного компонента, можуть бути заздалегідь визначені або динамічно вивчені з даних [17; 19].

2. Перемикаючі гібридні системи динамічно обирають та застосовують конкретну техніку рекомендацій на основі поточного контексту або характеристик користувача/елемента. Наприклад, система може використовувати CF для користувачів з великою історією взаємодій, але переключатися на CBF для нових користувачів або елементів [18].

3. Змішані гібридні системи генерують набір рекомендацій незалежно від кількох технік, а потім об'єднують ці окремі набори в єдиний, уніфікований список. Цей підхід особливо ефективний для надання більш різноманітного діапазону пропозицій [18].

4. У гібридних системах комбінації ознак, ознаки з різних джерел (наприклад, дані взаємодії користувачів, атрибути елементів, вік) інтегруються в уніфікований набір ознак, який потім подається в єдину модель рекомендацій [18; 19].

5. Каскадна гібридизація передбачає послідовне застосування технік, де одна модель може генерувати попередній набір кандидатів, який потім уточнюється іншою моделлю [20].

Гібридні системи пропонують значні переваги, включаючи покращену точність, підвищену різноманітність та більшу надійність порівняно з окремими підходами. Критично важливо, що вони дуже ефективні у вирішенні проблеми «холодного старту» та пом'якшенні проблем «розрідженості даних» шляхом інтелектуального поєднання контентної інформації (для нових елементів/користувачів) з колаборативними закономірностями (для існуючих

сутностей). Вони також забезпечують більш повне розуміння вподобань користувачів та взаємозв'язків між елементами [18; 19; 20].

Незважаючи на переваги, гібридні системи створюють певні виклики, насамперед підвищену складність інтеграції кількох технік, що може призвести до збільшення витрат на розробку, вартості та зусиль з підтримки. Вони все ще можуть стикатися з проблемами розрідженості даних, особливо при поєднанні численних джерел даних, і, хоча вони пом'якшують «холодний старт», це залишається викликом, особливо для абсолютно нових сценаріїв [18; 19].

1.2. Порівняльний аналіз ефективності основних підходів до рекомендацій

Отже, тепер можна провести порівняльний аналіз колаборативної фільтрації (табл. 1.1.), контентної фільтрації та гібридних систем, узагальнюючи їхні методології, переваги та недоліки.

Таблиця 1.1

Порівняння основних підходів до рекомендаційних систем

Критерій	**Колаборативна фільтрація (CF)**	**Контентна фільтрація (CBF)**	**Гібридні системи**
Принцип роботи	Рекомендує елементи на основі вподобань схожих користувачів або подібності між елементами, виходячи з взаємодій користувачів.	Рекомендує елементи, подібні до тих, що користувач вподобав у минулому, на основі атрибутів елементів та профілю вподобань користувача.	Поєднує дві або більше технік (зазвичай CF та CBF) для використання їхніх сильних сторін та пом'якшення слабких.

Продовження таблиці 1.1

Переваги	Не потребує про предметну область. Здатність до відкриття нового. Ефективна для неструктурованого контенту (просто текст чи зображення).	Добре справляється з проблемою «холодного старту» для нових елементів. Менша залежність від інших користувачів. Пояснювані рекомендації. Добре працює для нішевих інтересів.	Покращена точність, різноманітність та надійність. Ефективно вирішує проблеми «холодного старту» та «розрідженості даних». Комплексне розуміння вподобань.
Недоліки	Проблема «холодного старту» (для нових користувачів та елементів). Розрідженість даних. Складність включення додаткових ознак. Упередженість до популярності.	Залежність від характеристик контенту (наприклад, жанрів, тегів). Обмежена новизна («фільтровані бульбашки»). Складність обробки абстрактних даних. Проблема розрідженості контентних даних.	Підвищена складність інтеграції та підтримки. Можуть стикатися з розрідженістю даних при поєднанні багатьох джерел. Залишковий «холодний старт» для абсолютно нових сценаріїв.
Обробка «холодного старту»	Дуже погано (основна проблема).	Добре для нових елементів, але все ще складно для нових користувачів.	Добре (пом'якшує проблему, поєднуючи сильні сторони).
Обробка розрідженості даних	Дуже погано (основна проблема, знижує точність).	Може бути проблемою, якщо контентні ознаки розріджені.	Добре (покращує обробку, використовуючи різні джерела даних).
Потреба в знаннях предметної області	Низька (автоматично вивчає низьковимірне представлення).	Висока (для ручної роботи з даними та відбору важливих характеристик).	Середня-Висока (залежить від типу гібридизації та компонентів).

Продовження таблиці 1.1

Новизна рекомендацій	Висока (здатність до відкриття нового).	Низька («фільтровані бульбашки», схильність до схожих елементів).	Висока (поєднує здатність до відкриття нового з релевантністю).

Порівнюючи різні підходи до рекомендацій, можна побачити, шо система або краще відкриває нове, або більше залежить від знань про предмет і якісно підібраних ознак.

Колаборативна фільтрація має перевагу в тому, що може знаходити неочікувані, але доречні рекомендації, навіть без детального розуміння самих елементів — завдяки вмінню автоматично вивчати схожість між користувачами та об'єктами. Ця незалежність від специфічних ознак елементів дозволяє їй виявляти неочевидні зв'язки через колективну поведінку користувачів. Однак, саме ця особливість робить її вразливою до проблем «холодного старту» та розрідженості даних, оскільки для нових елементів або користувачів просто немає достатньо даних для обчислення подібності.

На противагу цьому, контентна фільтрація, яка покладається на детальну інформацію про елементи, ефективно вирішує проблему «холодного старту» для нових елементів, оскільки може використовувати їхні атрибути. Проте, ця залежність від попередньо визначених ознак обмежує її здатність пропонувати щось дійсно нове, часто призводячи до «фільтрованих бульбашок», де користувачі бачать лише контент, що відповідає їхнім існуючим інтересам. Необхідність ручної підгтовки ознак також є значним тягарем.

Протиріччя між бажанням знаходити нове і необхідністю обробляти нові дані, а також між автоматичним навчанням і залежністю від ручної роботи, показує, що потрібні комбіновані підходи.

Гібридні системи виникають як відповідь на ці проблеми, прагнучи досягти як новизни, так і релевантності шляхом поєднання попередніх

методологій. Вони дозволяють системі використовувати контентні ознаки, коли колаборативні дані недостатні (наприклад, для нових користувачів), і водночас використовувати колективний інтелект для розширення інтересів користувачів за межі їхніх явних вподобань.

1.3. Модульна структура сучасної рекомендаційної системи

Архітектура сучасних систем рекомендацій включає кілька основних шарів та компонентів, які залежать один від одного і разом забезпечують безперервний цикл роботи.

На найнижчому рівні знаходиться шар даних, який містить інформацію про користувачів, товари (або контент) та їх взаємодії. До нього входять профілі користувачів (демографічні дані, історія покупок, вподобання), характеристики товарів чи контенту (категорії, опис, медіа-метадані) і журнали поведінкових подій (перегляди, кліки, покупки, оцінки). Дані зберігаються в базах чи дата-лейках, часто із використанням технологій великих даних (SQL/NoSQL бази, Hadoop/Spark, потокові платформи як Kafka) [21; 22].

Над шаром даних розташовуються модулі обробки даних та навчання моделей. Сюди входять попередня обробка та агрегування: фільтрація шумових подій, формування сесій користувачів, екстракція ознак (feature engineering) тощо.

Далі йде етап зріджувального відбору (candidate retrieval), на якому від великого пулу всіх можливих товарів відбирається обмежена множина потенційно релевантних кандидатів для кожного користувача. Наприклад, Instagram щоденно скорочує мільярди роликів до сотень – тисяч кандидатів для стрічки кожного користувача [23]. На цьому етапі можуть застосовуватися прості фільтри або моделі попереднього відбору (наприклад, векторні зближення або легкі моделі машинного навчання).

Після цього настає етап ранжування (scoring/ranking). Отримані кандидати оцінюються за моделлю переваг користувача, і найбільш релевантні позиції відбираються у кінцевий перелік рекомендацій.

Часто в архітектурі виділяють ще етап переранжування (re-ranking) для врахування цілей, таких як різноманітність, свіжість чи відповідність бізнес-політиці [23]. Загалом, можна розглядати систему рекомендацій як конвеєр, що послідовно виконує збирання даних, далі попередній відбір кандидатів, наступним кроком ранжування і нарешті формування вихідного списку [21; 23].

Системи рекомендацій зазвичай мають два ключові модулі: модуль matching (пошук широкого набору можливих рекомендацій) та модуль ranking (фінальне сортування іграночкового списку) [21]. Модуль matching виконує грубе відсівання – знаходить сотні чи тисячі кандидатів з урахуванням базових подібностей чи обмежень. Модуль ranking детально оцінює ці кандидати і будує фінальний список (наприклад, топ-10) [21]. Такий двоетапний підхід дозволяє забезпечувати низьку затримку відповіді: замість оцінювати всі мільйони товарів для кожного запиту, спершу звужується пошукова область, а потім на неї накладається складна модель [21; 23].

Окрім алгоритмічних модулів, в архітектуру входить інфраструктура для розгортання: сервіси реального часу, кешування, балансування навантаження тощо. Вимоги до системи включають підтримку мільярдів подій і мільйонів користувачів, швидку відповідь (мілісекунди для генерації рекомендацій) та масштабованість з урахуванням пікових навантажень [21]. На рисунку 1.3. графічно показано зразок загальної архітектури рекомендаційної системи.

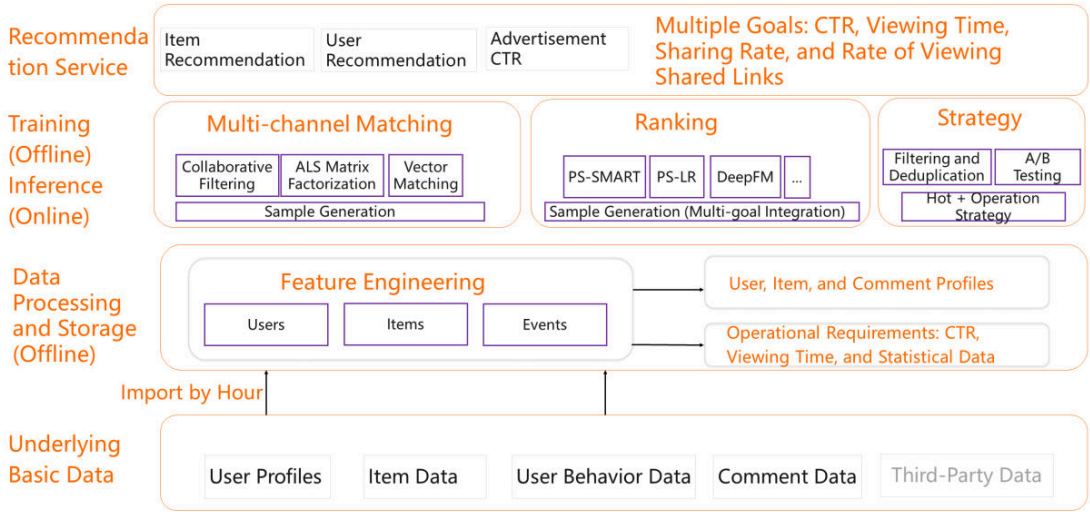

Рис. 1.3. Загальна архітектура рекомендаційної системи [21]

Сучасні рішення часто розгортають у хмарі з використанням автоскейлінгу, in-memory баз даних або кешів (Redis, Cassandra) та спеціалізованих пошукових індексів для прискорення пошуку подібностей [21; 22].

Приклади архітектур великомасштабних платформ:

Amazon

Еволюція рекомендаційної системи Amazon була обумовлена необхідністю масштабованості. Історично, рекомендаційні системи покладалися на User-Based Collaborative Filtering (UBCF), зіставляючи відвідувача з іншими клієнтами зі схожою історією покупок. Однак, для каталогу Amazon та його величезної бази клієнтів, цей підхід вимагав порівняння історії покупок по всій базі, що було занадто повільним для забезпечення рекомендацій в рамках однієї сесії на сайті [24].

У 2003 році Amazon перейшов до Item-to-Item Collaborative Filtering. Цей підхід фокусується на кореляції між [24] продуктами, а не клієнтами. Алгоритм перевіряє нещодавню історію покупок користувача і для кожної покупки знаходить список пов'язаних товарів. Оскільки даний товар купує лише невелика

підмножина клієнтів, обчислення пов'язаності товарів (наприклад, «товар B пов'язаний з товаром A, якщо клієнти, які купують A, незвично схильні також купувати B») стало обчислювально ефективним для щоденного оновлення. Для боротьби з упередженням популярності, коли система рекомендує лише бестселери, Amazon використовував метрику пов'язаності, засновану на диференціальних ймовірностях, а не просто на частоті спільної покупки. Це гарантує, що рекомендації ґрунтуються на підвищеній ймовірності покупки, а не просто на високій базовій ймовірності покупки активними клієнтами [24].

Сучасні системи Amazon, особливо для Prime Video, використовують просунуті методи, включаючи моделювання як задачі матричного доповнення та розробку алгоритмів, які відходять від стандартної матричної факторизації, що призвело до значного покращення продуктивності [24].

Netflix

Архітектура Netflix — це комбінація різних методів, які поєднують у собі фільтрацію за схожістю, глибокі нейронні мережі та моделі, що працюють з графами зв'язків, уникаючи використання лише одного єдиного рішення для всього [25].

Ключові особливості архітектури Netflix:

1. Персоналізація інтерфейсу: Персоналізація є багатошаровою і поширюється на весь інтерфейс. Система персоналізує не лише вибір рекомендованих заголовків, але й вибір рядків (наприклад, «Продовжити перегляд»), їхній порядок та навіть представлення самого заголовка [26].

2. Створення ознак у реальному часі: Платформа щоденно обробляє терабайти даних взаємодії. Вона використовує автоматизовані конвеєри, які безперебійно перетворюють сирі дані про поведінку, обчислюючи вектори вподобань користувачів та матриці схожості контенту в реальному часі [25].

3. Foundation Models: Недавні розробки Netflix зосереджені на створенні базових моделей (Foundation Models) для рекомендацій. Ці універсальні моделі

навчаються передбачати наступну дію користувача і створюють цінні векторні представлення як для самих користувачів, так і для різних сутностей (відео, жанри). Ці векторні представлення можна потім використовувати в різних допоміжних завданнях, а саму модель можна додатково налаштувати, використовуючи конкретні дані певного додатку [27].

1.4. Методологія формування вхідних даних та алгоритмічні рішення

Рекомендаційні системи ґрунтуються на масивах фактичної поведінки користувачів (оцінках, лайках, покупках, переглядах тощо) [28]. Такі дані бувають явними (explicit) і неявними (implicit). Явні відгуки – це пряму інформацію від користувачів (рейтинги, коментарі, опитування), тоді як неявні спостереження отримуються пасивно, без прямого залучення користувача [29; 30]. Наприклад, система може фіксувати перегляди сторінок, клацання, час перебування на сторінці або історію покупок [30].

Ці неявні сигнали часто дозволяють «зріджено» оцінити вподобання користувачів: виявлено кореляцію між часом перегляду контенту та оцінками, які користувач залишав, а рішення про покупку вважається дуже сильним індикатором зацікавленості користувача у товарі [29].

Під час збору даних в системі рекоменацій зазвичай використовують журнали взаємодій користувачів (log-файли вебсерверів, події з клієнтських застосунків, системи відстеження подій), бази даних транзакцій та інтерактивні опитування. Наприклад, в інтернет-магазині зберігають історію переглядів, кліків та придбаних товарів; на стрімінгових сервісах – записи переглядів відео чи музичних треків; у соціальних мережах – лайки, підписки та коментарі користувачів. Цей масив необроблених даних потребує попереднього очищення (видалення неактивних користувачів, заповнення пропусків тощо) і стандартизації форматів. Важливо також враховувати зміщення даних – наприклад, популярні товари можуть бути надміру представлені в історіях, що

призводить до викривлень уподобань [28].

Аналіз поведінкових даних проводиться за допомогою методів статистики та машинного навчання. Часто застосовують кластеризацію для сегментації користувачів за схожими патернами поведінки: наприклад, алгоритм k-means може виділити кластери користувачів із подібними смаковими вподобаннями, що полегшує побудову таргетованих рекомендацій [31].

Також використовують асоціативний аналіз (виявлення частих наборів товарів у кошику) та послідовний аналіз (шаблони переглядів та кліків), що допомагає знайти корисні закономірності (наприклад, які товари часто купують разом). Методи класифікації та регресії можуть прогнозувати ознаки користувача (демографію, інтереси) на основі історії взаємодій. Завдяки сучасним засобам аналізу великих даних (Big Data) ці підходи масштабуються на мільйони користувачів і об’єктів.

При цьому слід пам’ятати про системні обмеження: на початку роботи системи даних мало (проблема «холодного старту»), а самі дані можуть бути неповними чи зміщеними. Наприклад, поведінка ранніх користувачів може не відображати вподобань решти аудиторії.

Крім того, на збір даних впливають етичні й правові обмеження: важливо забезпечити приватність користувачів, а збір персональних даних має відбуватися відповідно до законодавства. Загалом, методи збору та аналізу даних повинні поєднувати надійність джерел із сучасними алгоритмами видобування знань, аби отримати достовірну модель поведінки користувача [28; 29].

Вибір алгоритмічного підходу залежить від завдання (Retrieval чи Ranking), типу зворотного зв'язку (явний чи неявний) та необхідності включення контексту.

k-Nearest Neighbors (kNN)

Метод kNN базується на метриках схожості для пошуку або схожих користувачів (User-based CF), або схожих товарів (Item-based CF). Item-to-Item

CF є особливо важливим, оскільки саме він був прийнятий Amazon на початку 2000-х років, забезпечуючи обчислювальну ефективність, необхідну для щоденного оновлення зв'язків між товарами на великому каталозі. kNN залишається цінним для швидкого зріджувального відбору у сучасних системах [24].

Matrix Factorization (MF)

Matrix Factorization (Матрична факторизація) є класом моделей колаборативної фільтрації, що розкладає розріджену матрицю взаємодій $R \in R^{m \times n}$ (де m — користувачі, n — товари) на дві матриці нижчого рангу: латентну матрицю користувачів $P \in R^{m \times k}$ та латентну матрицю товарів $Q \in R^{n \times k}$. Тут $k \ll m,n$ — розмірність латентного простору [32].

Ключові методи включають Singular Value Decomposition (SVD) та Alternating Least Squares (ALS). SVD є популярним для прототипування через свою ефективність у передбаченні явних рейтингів, і його легко реалізувати за допомогою бібліотек, як-от Surprise [33].

Нейронні мережі (Deep Learning)

Глибоке навчання надає можливість виявляти складні, нелінійні взаємозв'язки та ефективно інтегрувати контекстуальні та контентні фічі, що є складним для класичних MF-моделей.

Word2Vec для товарів (Item2Vec): Цей спосіб бере ідеї з галузі обробки природної мови, де комп'ютери розуміють текст. Послідовність дій користувача (наприклад, список товарів, на які він клікав або які купив) розглядається так, ніби це «речення». А кожен окремий товар у цьому списку розглядається так, ніби це «слово» в цьому реченні. Item2Vec навчає комп'ютер представляти кожен товар у вигляді набору чисел (вектора). Після цього, за допомогою цих чисел, система може дуже швидко знаходити інші, схожі товари, що є ідеальним рішенням для першого етапу швидкого відбору кандидатів [34].

Вибір алгоритму в великомасштабних РС визначається тим, на якому етапі (Retrieval чи Ranking) він використовується (табл. 1.2).

Таблиця 1.2

Порівняння основних алгоритмів рекомендаційних систем

Алгоритм	**Основа**	**Сильні Сторони**	**Оптимальне Застосування**
kNN (Схожість між товарами)	Знаходить схожі товари на основі простих правил (наприклад, «ці товари часто купують разом»).	Висока обчислювальна ефективність, прозорість	Ідеально підходить для першого, швидкого етапу, коли потрібно знайти багато схожих товарів на основі того, що ви дивилися чи купували останнім часом.
Матрична факторизація (SVD/ALS)	Уявляє собі всіх користувачів і всі товари у вигляді великої таблиці з пропусками і вміє «вгадувати» недостаючі значення.	Ефективна на розріджених даних, моделює глобальні уподобання	Добре підходить для базового відбору кандидатів, а також для того, щоб передбачити, яку оцінку ви б поставили товару, який ще не купували.
Глибоке навчання (Two-Tower, DNN)	Перетворює користувачів і товари на цифрові вектори (набори чисел) і знає, як знаходити складні, неочевидні зв'язки між ними.	Висока точність рекомендацій, бо вміє враховувати багато деталей одночасно (контекст)	Найкраще використовувати на двох етапах: для пошуку кандидатів за допомогою пошуку серед векторів, і для фінального точного сортування цих кандидатів у найкращому порядку.

Для дуже великих систем, де дуже важливо використовувати багату та детальну контекстну інформацію (наприклад, тип пристрою, час доби, час перебування на сторінці), яку важко і незручно вбудовувати в класичні моделі матричної факторизації, глибоке навчання дає дуже велику перевагу. Саме з цієї причини глибоке навчання є основним і найпоширенішим на етапі фінального

сортування кандидатів. На цьому етапі більш висока точність рекомендацій і можливість використовувати складні комбінації різних ознак цілком виправдовують той факт, що такі моделі вимагають значно більше потужностей для обчислень.

Натомість, на попередньому етапі швидкого відбору кандидатів, де найголовішою вимогою є висока швидкість роботи, часто використовуються більш ефективні за швидкістю методи. Наприклад, алгоритм пошуку схожих товарів (kNN), вектор для якого були розраховані заздалегідь, що дозволяє дуже швидко шукати схожі елементи.

1.5. Методи оцінки ефективності рекомендацій

Оцінка якості рекомендаційних систем проводиться як офлайн (за історичними даними), так і онлайн (за реальними реакціями користувачів).

Офлайн-метрики зазвичай зосереджені на правильності прогнозу чи ранжуванні. Найпоширеніші з них:

1) Precision@K і Recall@K: точність і повнота в списку топ-K рекомендацій. Precision показує, яку частку рекомендованих товарів користувач дійсно обрав, Recall – частку з усіх релевантних (куплених чи оцінених ним) усього списку рекомендацій [35]. Часто використовують комбінацію Precision/Recall.
2) NDCG@K (Normalized Discounted Cumulative Gain): оцінює ранжування з урахуванням позицій, де товари, які користувач вважає релевантними, мають стояти якомога вище. Це дробне значення від 0 до 1, яке враховує порядок вподобаних елементів [35; 36].
3) Mean Average Precision (MAP) і Mean Reciprocal Rank (MRR): середнє значення точності по користувачах та обернена рангова позиція першого правильного результату відповідно. MAP враховує розташування всіх

релевантних елементів у топ-K, а MRR фокусується на тому, наскільки високо стоїть перший правильний [35; 36].

4) RMSE/MAE: середнє квадратичне/абсолютне відхилення прогнозованих рейтингів від фактичних. Використовувалися в класичних задачах прогнозування оцінок, але в сучасних системах популярніші метрики ранжування [36].
5) Hit Rate / Coverage: показник того, яка частка користувачів отримали хоч один релевантний результат у топ-K (Hit Rate), або наскільки широкий «каталог» товарів задіяно в рекомендаціях (coverage) [35; 36].
6) Диверсіті, новизна, серендипність: додаткові метрики, що оцінюють якість рекомендацій з точки зору різноманітності та несподіваності пропозицій. Наприклад, Diversity вимірює наскільки рекомендовані товари різні між собою, Novelty – наскільки новими чи рідкісними є ці товари для користувача, Serendipity оцінює несподіваність (аналог "приємного сюрпризу"). Їх використовують, щоб уникнути «кавунового ефекту», коли система пропонує одне й те саме підряд [35].

Ці офлайн-метрики обчислюються на тестових наборах даних: історія розбивається на тренувальну та тестову частину (часова валідність, cross-validation, leave-one-out тощо). Модель навчають на першій частині, а прогнозують для наступних подій, порівнюючи з реальним вибором користувача. Це дозволяє оцінити зміни моделі швидко та без ризику впливу на реальних користувачі [35].

Однак офлайн-метрики не завжди корелюють із задоволеністю користувачів. Тому важливо проводити онлайн-експерименти (A/B-тестування). При онлайн-оціночних тестах частині користувачів показують рекомендації від нової системи, іншим – від старої чи контрольної. Потім порівнюють бізнес-метрики: клікабельність (CTR), конверсію у покупку, час взаємодії, коефіцієнт відмов, середній чек тощо [35]. Наприклад, один з порадників рекомендує збільшувати оборот платформи, інший – підвищити різноманітність продажів. Через тиждень порівнюють, яка група була більш успішною [35; 22].

В онлайн-експериментах також часто використовують методи статистичного аналізу: ізольовані змінні, тестування значущості (t-тест, U-критерій Манна–Уітні) для визначення чи є відмінності не випадковими. Крім того, застосовують метрики залучення: коефіцієнт повернення користувача, коефіцієнт утримання (retention), прослуханий час, відгуки, що враховують довготривалі ефекти рекомендацій.

Використання лише одного показника для оцінки якості системи є недостатнім, тому що різні показники вимірюють різні сторони якості роботи системи.

Основна слабкість показників Precision@k та Recall@k полягає в тому, що вони не враховують позицію, на якій знаходиться елемент у списку. Наприклад, якщо у двох різних списках рекомендацій, в кожному з яких по 10 елементів, є по 5 корисних для користувача товарів, значення Precision@k буде однаковим (50%) в обох випадках. І не важливо, чи розташовані ці корисні товари на самих перших позиціях (1-5), чи вони знаходяться в кінці списку (на позиціях 6-10). Оскільки користувачі зазвичай не прокручують список далеко і дивляться переважно на перші позиції, така оцінка, що не враховує порядок, є неприйнятною для комерційних систем, які продають товари [37].

Саме тому метрика NDCG є критично важливою для офлайн-тестування: вона зменшує вагу або значення корисних елементів, які розміщені дуже низько в списку, тим самим мотивуючи модель ставити найкращі та найкорисніші рекомендації на самі перші місця.

Хоча NDCG є найкращим показником для офлайн-оцінки якості сортування списку, вона не може повністю замінити собою онлайн-перевірку на реальних користувачах. Остаточна ефективність рекомендаційної системи завжди повинна перевірятися через A/B тестування, де використовуються бізнес-показники, такі як загальний дохід, здатність утримувати користувачів на довгий термін або загальна тривалість їх сесії на сайті. Це єдиний спосіб достовірно виміряти довгостроковий вплив системи на користувача та бізнес.

Різні метрики фокусуються на різних аспектах оцінки якості та врахуванні позиції елемента в списку (табл. 1.3).

Таблиця 1.3

Оцінка якості: Метрики та їхній фокус

Метрика	**Тип оцінки**	**Rank-Aware?**	**Фокус**
RMSE	Точність прогнозу рейтингу	Ні	Точність передбачення явного балу
Precision@k (P@k)	Точність у Top-K	Ні	Якість рекомендаційного списку, мінімізація хибнопозитивних
Recall@k (R@k)	Повнота у Top-K	Ні	Здатність охопити максимальну кількість релевантних елементів
NDCG@k	Якість Ранжування	Так	Винагороджує за розміщення найбільш релевантних елементів на верхніх позиціях
MAP@k	Якість Ранжування (середнє)	Так	Загальна ефективність ранжування на наборі запитів

Таким чином, оцінка ефективності є комбінованою: спершу офлайн-метрики ранжування (Precision/Recall@K, NDCG, MRR тощо) перевіряють коректність рекомендацій за історичними даними [35], потім онлайн-метрики користувача (CTR, конверсія, час сеансу) й бізнес-метрики (продажі, дохід) дають уявлення про реальний ефект [35; 23]. Для прийняття рішень про запуск нового алгоритму чи його покращення зазвичай використовують поєднання A/B-тестів і аналізу ключових бізнес-метрик.

Наукові дослідження з аналізу даних дуже часто використовують показник під назвою Mean Average Precision (MAP@k) як один з головних критеріїв для оцінки якості сортування списків. Навіть дуже низькі цифрові значення цього показника (наприклад, 0.027) можуть вважатися дуже хорошим результатом у складних завданнях, особливо якщо такого результату вдалося досягти одній

окремій моделі, а не комбінації з багатьох моделей. Цей момент підкреслює, наскільки важливо порівнювати результати будь-якої нової моделі з результатами базових або попередніх моделей, щоб розуміти, чи є реальне покращення [38].

Крім того, професійна оцінка якості роботи системи вимагає аналізу її вбудованих упереджень. Наприклад, щоб оцінити, наскільки система схильна рекомендувати лише популярні товари (ігноруючи менш відомі), проводять спеціальну перевірку. Для цього всіх користувачів розподіляють на групи в залежності від того, наскільки популярні товари вони зазвичай купують. Таке дослідження дозволяє чітко і в цифрах виміряти ступінь упередженості системи, який є властивим майже всім системам, що базуються на спільній поведінці користувачів [38].

Висновок

У ході дослідження було визначено сучасні методи побудови рекомендаційних систем, зокрема колаборативної фільтрації, контентно-орієнтованого підходу та гібридних моделей. Проаналізовано ключові принципи функціонування кожного з підходів, їхні переваги та недоліки.

Найбільш детально була розглянута колаборативна фільтрація, яка є основою для багатьох комерційно успішних рекомендаційних систем. Проте її ефективність знижується в умовах «холодного старту» та високої розрідженості даних. Контентна фільтрація, хоча і ефективна для нових елементів, має інші обмеження — зокрема, схильність до повторення вже відомого контенту та високу залежність від якісної предметної інформації. У цьому контексті гібридні підходи виступають як найперспективніший напрям розвитку, поєднуючи сильні сторони окремих моделей та пом’якшуючи їхні слабкі сторони.

Аналіз підтвердив доцільність обраної теми — розробки системи

рекомендацій на основі поєднання колаборативної фільтрації та поведінкових даних користувача в реальному часі. Такий підхід дозволяє не лише покращити точність рекомендацій, а й адаптувати їх до поточної активності користувача, що особливо актуально в умовах динамічного онлайн-середовища.

Також було визначено архітектуру, алгоритми та методи оцінки ефективності сучасних рекомендаційних систем. Аналіз різних джерел та практичних прикладів дозволив сформувати цілісне уявлення про ключові етапи побудови рекомендаційних рішень: від збору та попередньої обробки поведінкових даних до вибору алгоритмічного підходу та перевірки його ефективності.

Важливим результатом стало дослідження методів збору та аналізу поведінкових даних користувачів. Було з’ясовано, що якість і повнота даних безпосередньо впливають на точність рекомендацій, а ефективна робота системи неможлива без урахування як явних оцінок, так і неявних сигналів. Аналіз викликів, таких як проблема «холодного старту» та упередженість даних, сформував розуміння того, як у подальшій роботі можна уникати або мінімізувати ці обмеження. Окремо були розглянуті методи оцінки якості рекомендаційних систем, включаючи офлайн-метрики (Precision, Recall, NDCG, MAP, RMSE) та онлайн-методи (A/B-тестування). Це надало змогу зрозуміти важливість комплексного підходу до оцінки, який враховує як технічну точність алгоритмів, так і реальні бізнес-показники, зокрема залучення та утримання користувачів. Отримані знання дозволять перейти до реалізації системи, яка об’єднає колаборативні методи з аналізом поведінкових сигналів у динаміці.

РОЗДІЛ 2
РЕАЛІЗАЦІЯ КОЛАБОРАТИВНОЇ ФІЛЬТРАЦІЇ

2.1. Вибір та обґрунтування алгоритму колаборативної фільтрації для предметної області

Колаборативна фільтрація базується на аналізі взаємодій користувачів зі спільним набором товарів. Її мета — знайти приховані закономірності у матриці user–item та використовувати їх для рекомендацій.

Як було визначено раніше у сучасних системах застосовуються три основні підходи матричне розкладання (SVD), методи на основі найближчих сусідів (k-NN) та алгоритми з імпліцитними оцінками (ALS — Alternating Least Squares). Основними критеріями вибору стали: здатність ефективно працювати з надзвичайно розрідженими даними (оскільки кількість товарів набагато перевищує кількість оцінок одного користувача), масштабованість для періодичного переобчислення на великому обсязі історичних даних, та інтерпретованість результату для подальшого гібридного злиття. Враховуючи попередній огляд методів було прийнято рішення реалізувати модель матричної факторизації (Matrix Factorization) з використанням сингулярного розкладання (SVD).

Вибір цього підходу пояснюється низкою причин. По-перше, алгоритм SVD++ який враховує implicit feedback дозволяє відновити приховані (latent) фактори користувачів та товарів навіть за наявності дуже малої кількості явних взаємодій (наприклад, кліків чи переглядів), що типовий для e-commerce. По-друге, на відміну від методів на основі сусідства (k-NN), які потребують обчислення подібностей онлайн, матрична факторизація дозволяє попередньо обчислити рекомендації для всіх користувачів офлайн. Це відповідає архітектурі системи, де CF-модель перенавчається періодично, а результати кешуються в швидкому сховищі (hot storage). Запит до API виконує лише швидке отримання

вже готового списку. По-третє, латентні вектори користувачів та товарів, отримані в результаті навчання, є компактним представленням їх переваг та характеристик. Це створює основу для потенційного вдосконалення системи, наприклад, для поєднання з контентними ознаками в майбутньому.

Крім того, SVD добре інтегрується у гібридні моделі, оскільки генерує компактні векторні подання як користувачів, так і товарів. Такі вектори легко комбінуються з результатами поведінкового аналізу, що дозволяє створювати узгоджені рекомендаційні списки, які враховують як довгострокові вподобання користувача, так і актуальний контекст його взаємодій у реальному часі.

Для практичної реалізації обрано бібліотеку Surprise (Python scikit-learn for recommender systems), оскільки вона надає готові, оптимізовані реалізації алгоритмів SVD та SVD++, зручний інтерфейс для крос-валідації та оцінки метрик, що пришвидшує розробку. Алгоритм налаштовується на мінімізацію середньоквадратичної помилки між прогнозованим «рейтингом» взаємодії та фактичною наявною вагою (де вага може бути похідним значенням від типу події — перегляд, клік, додавання в кошик).

Таким чином, обраний алгоритм SVD++ є компромісом між точністю, продуктивністю та архітектурними вимогами системи, що працює в режимі періодичного офлайн-навчання та онлайн-обслуговування.

2.2. Проєктування пайплайну навчання та оновлення моделі

Після вибору алгоритму колаборативної фільтрації ключовим етапом реалізації стала розробка надійного та ефективного пайплайну навчання та оновлення моделі.

Даний пайплайн є серцем офлайн-компонента системи, оскільки саме він відповідає за перетворення накопичених історичних даних на актуальні рекомендаційні знання. Його архітектура була спроектована з урахуванням

таких принципів: відмовостійкість, відтворюваність результатів, ефективне використання ресурсів та безупинна інтеграція з живими сервісами. Процес був розділений на чіткі, ізольовані етапи, що дозволило створити модульну систему, здатну до масштабування та легкої діагностики.

Загальну схему життєвого циклу моделі, що об'єднує описані етапи, представлено на рис. 2.1.

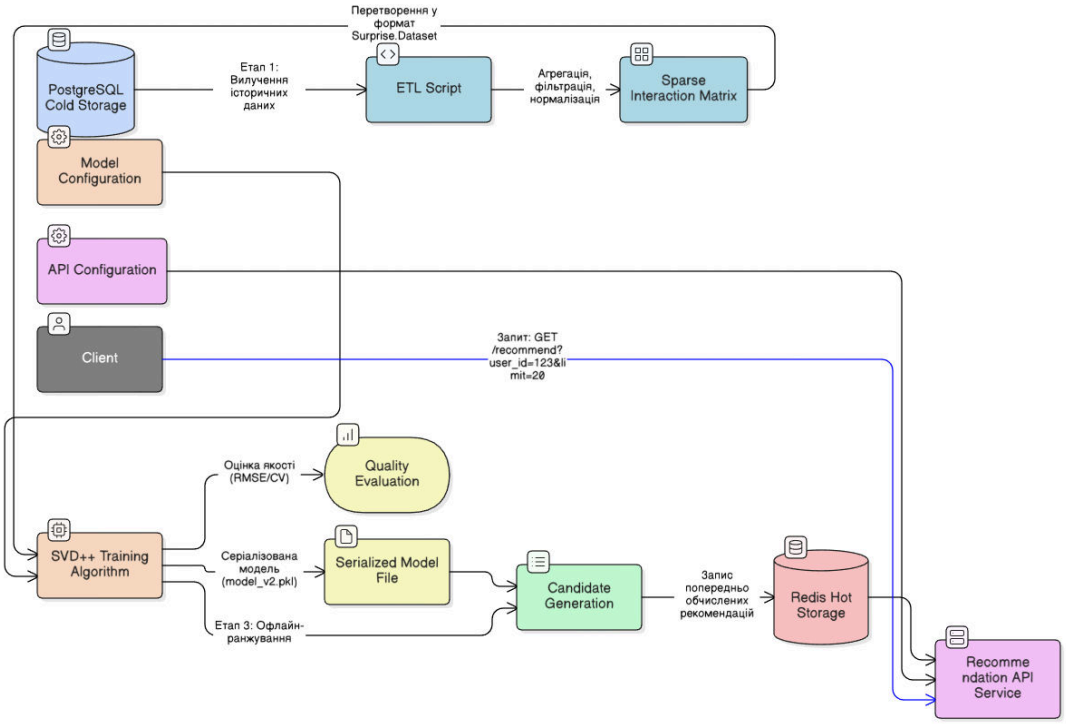

Рис. 2.1. Повний життєвий цикл моделі колаборативної фільтрації

Як видно з рис. 2.1, процес починається з етапу вилучення даних. Оскільки модель навчається на основі «холодних» історичних даних, що зберігаються в PostgreSQL, необхідно було забезпечити стабільне та повне їх отримання. Для цього було реалізовано спеціалізований скрипт, який виконує запит до бази даних, фільтруючи події за встановленим інтервалом. Наприклад, модель може переобчислюватись щодня на основі даних за останні 90 днів.

Такий підхож дозволяє балансувати між актуальністю моделі (врахування нових трендів) та її стабільністю (ігнорування разових спалахів активності).

Запит агрегує сирі події користувачів (перегляди, кліки, покупки) у формат, придатний для навчання: (user_id, item_id, weight).

Вага взаємодії є критичним параметром, який кодує силу користувацької переваги. У нашій реалізації було введено просту, але ефективну схему ваги: кліку присвоюється вага 1.0, додаванню в кошик – 1.5, а завершеній покупці – 2.0. Це дозволяє моделі розрізняти рівень зацікавленості користувача, що істотно покращує релевантність рекомендацій порівняно з бінарним підходом (взаємодія була/не була).

Наступна фаза – попередня обробка даних та побудова матриці взаємодій. Отриманий набір даних піддається серії трансформацій, спрямованих на підвищення якості майбутньої моделі. По-перше, застосовується фільтрація за мінімальною кількістю взаємодій. Було встановлено порогові значення: товар повинен бути взаємодіяв з щонайменше 5 різними користувачами, а користувач – з щонайменше 3 різними товарами. Це дозволяє відсіяти «шуми» – випадкові кліки або нові товари/користувачі, для яких неможливо зробити статистично значущий висновок, що сприяє боротьбі з проблемою розрідженості матриці. По-друге, для уникнення домінування надмірно активних користувачів у функції втрат алгоритму, ваги їх взаємодій нормалізуються. Після фільтрації створюється об’єкт surprise.Dataset, який інкапсулює матрицю взаємодій у розрідженому форматі, оптимальному для подальших обчислень.

Власне процес навчання моделі є центральним обчислювальним етапом. Для алгоритму SVD++ з бібліотеки surprise було проведено пошук гіперпараметрів з метою мінімізації середньоквадратичної помилки (RMSE) на валідаційній вибірці. Критичними параметрами, що підлягали налаштуванню, були кількість латентних факторів (n_factors), кількість епох навчання (n_epochs) та коефіцієнти регуляризації. Регуляризація грає особливо важливу роль у запобіганні перенавчанню, оскільки початкові дані є надзвичайно розрідженими. Для прискорення процесу навчання використовуються оптимізовані лінійні алгебраїчні бібліотеки, такі як NumPy та SciPy, на яких побудована surprise.

Після завершення навчання отримана модель, що включає вектори факторів для кожного користувача та товару, серіалізується у форматі .pkl (за допомогою joblib для ефективності) та зберігається на диску як артефакт.

Однак навчена модель сама по собі не є кінцевим продуктом для онлайн-обслуговування. Наступним критичним кроком є генерація кандидатів (candidate generation). Метою цього етапу є попередній розрахунок рекомендацій для кожного активного користувача, щоб уникнути витратних обчислень у момент запиту до API. Для цього реалізований скрипт завантажує навчену модель та ітерує по списку всіх користувачів із навчального набору. Для кожного користувача моделлю прогнозується «рейтинг» для всіх товарів, з якими він ще не взаємодіяв. Після цього для користувача відбирається топ-N товарів (наприклад, N=100) з найвищим прогнозованим рейтингом. Ці пари (user_id, list_of_recommended_item_ids) потім зберігаються не в сирому вигляді, а у структурованому форматі, оптимізованому для швидкого читання

У нашій реалізації для цього використано Redis як сховище типу «ключ-значення». Ключем виступає user_id, а значенням – серіалізований список ID рекомендованих товарів. Такий підхід забезпечує час відповіді на рівні мілісекунд при обслуговуванні запитів, що є вимогою для системи реального часу.

Оновлення моделі є періодичним процесом, керованим за допомогою планувальника завдань Apache Airflow (або, у простішому варіанті прототипу, системного cron). Пайплайн запускається за фіксованим розкладом (наприклад, щопівночі). Його відмовостійкість забезпечується механізмами повторних спроб при збоях у читанні з БД або помилках навчання. Важливою особливістю архітектури є стратегія «гарячого» перемикання (hot swap) моделі. Нова згенерована мапа рекомендацій спочатку завантажується в Redis під новим набором ключів (наприклад, з префіксом cf:v2:). Лише після повного та успішного завершення завантаження конфігурація API-сервісу оновлюється так, щоб вказувати на новий префікс. Це гарантує нульовий простій сервісу

рекомендацій та уникнення ситуацій, коли користувач отримує неповний або пошкоджений список.

Заключним, але не менш важливим етапом пайплайну є моніторинг та логування. Кожен запуск генерує звіт, що містить ключові метрики: час виконання кожного етапу, розмір вхідного набору даних, кількість користувачів та товарів після фільтрації, значення RMSE на валідації, а також приклад згенерованих рекомендацій для контрольних користувачів. Ці дані зберігаються разом із артефактом моделі, формуючи історію змін та дозволяючи відстежувати деградацію якості з часом. Таким чином, розроблений пайплайн навчання та оновлення не лише автоматично продукує актуальні рекомендації, але й забезпечує передбачуваність, стабільність та можливість постійного вдосконалення колаборативного компоненту гібридної системи.

2.3. Побудова матриці взаємодій та обробка розрідженості

Ядром будь-якої моделі колаборативної фільтрації є матриця взаємодій, структура даних, що формалізує взаємозв'язки між користувачами та об'єктами рекомендацій. У контексті e-commerce побудова цієї матриці є нетривіальним завданням через притаманну даній предметній області властивість – надзвичайну розрідженість даних. Під розрідженістю розуміється співвідношення між кількістю фактично заповнених елементів матриці (наявні взаємодії) та загальною кількістю її елементів (всі можливі пари «користувач-товар»). У реальних системах розмірність матриці може сягати мільйонів користувачів і десятків мільйонів товарів, тоді як середній користувач взаємодіє лише з декількома десятками або сотнями позицій. Це призводить до того, що рівень заповнення матриці часто не перевищує часток відсотка, створюючи фундаментальну проблему для алгоритмічних методів.

Першим кроком у побудові матриці є визначення її семантики. У класичній постановці задачі рекомендацій матриця має розмірність U x I, де U – кількість

унікальних користувачів, а I – кількість унікальних товарів. Значення елемента r_ui традиційно інтерпретується як явний рейтинг, що користувач u дав товару i. Однак у домені електронної комерції явні рейтинги (у вигляді зірочок або оцінок від 1 до 5) є рідкістю. Натомість основним джерелом даних виступають неявні зворотні зв'язки (implicit feedback): перегляди, кліки, додавання в кошик, покупки.

Таким чином, значення r_ui перестає бути оцінкою в прямому сенсі, а стає числовою мірою сили або частоти взаємодії. У нашій реалізації було прийнято рішення про використання зважених неявних зворотних зв'язків. Кожному типу події присвоюється певна вага, яка відображає її інформативність щодо справжньої переваги користувача. Наприклад, простий перегляд товару може свідчити про початковий інтерес, тоді як покупка є однозначним і сильним сигналом позитивного вподобання.

Емпірично було встановлено наступну схему ваг: перегляд – 0.5, клік – 1.0, додавання до кошика – 2.0, покупка – 3.0. Якщо користувач взаємодіяв з одним товаром кілька разів (наприклад, переглядав його, а потім купив), ваги агрегуються шляхом простого сумування, що дозволяє відобразити кумулятивний інтерес.

Створення самої матриці у фізичній пам'яті у вигляді повної двовимірної структури неможливе через її гігантські розміри. Натомість використовується розріджене подання (sparse representation), яке зберігає лише ненульові елементи. У контексті Python та бібліотеки surprise це реалізується через спеціальні структури даних, що внутрішньо кодують інформацію у вигляді трьох списків: ідентифікаторів користувачів, ідентифікаторів товарів та значень рейтингу. Такий підхож радикально зменшує вимоги до пам'яті та дозволяє ефективно виконувати матричні операції, оптимізовані для розріджених даних.

Однак навіть у такому форматі матриця може містити «шуми» та непродуктивну інформацію. Для підвищення якості даних було впроваджено двоетапну процедуру фільтрації. На першому етапі застосовується фільтрація за

мінімальною кількістю взаємодій. Було встановлено, що товари, з якими взаємодіяла дуже мала кількість користувачів (наприклад, менше 5), часто є новинками, нішевими продуктами або артефактами, що не несуть корисного сигналу для колаборативної фільтрації. Їх включення до матриці не покращує, а часто погіршує якість моделі, збільшуючи розмірність простору без збільшення інформативності. Аналогічно, користувачі, які здійснили дуже мало взаємодій (менше 3), не мають достатнього профілю для осмисленого порівняння з іншими. Виключення таких рядків і стовпців істотно зменшує розмірність задачі та підвищує щільність решти матриці, що позитивно впливає на збіжність алгоритму факторизації.

Другим етапом обробки розрідженості є нормалізація даних. Навіть після фільтрації залишається проблема нерівномірного розподілу активності: деякі користувачі є надзвичайно активними, формуючи сотні взаємодій, тоді як більшість мають лише кілька. Без нормалізації модель може стати упередженою на користь смаків надактивних користувачів, оскільки їхні дані будуть домінувати у функції втрат алгоритму. Для вирішення цієї проблеми було застосовано техніку центрування за користувачем (user-based centering). Для кожного користувача u обчислюється середнє значення його ваг взаємодій μ_u. Потім кожне вихідне значення r_ui замінюється на нормалізоване r'_ui = r_ui - μ_u. Ця операція зсуває дані таким чином, що нове значення відображає не абсолютну силу взаємодії, а її відхилення від середнього рівня активності даного користувача. Наприклад, покупка для «малопотужного» користувача може мати нормалізоване значення +2.5, тоді як така сала покупка для постійного клієнта, який купує часто, може отримати значення +0.5. Це дозволяє моделі краще виявляти відносні переваги, незалежно від загальної активності.

Процес побудови матриці можна візуалізувати за допомогою блок-схеми, що відображає трансформацію сирих даних у готову розріджену структуру. Сирі події з потокового джерела або бази даних спочатку агрегуються за парою (користувач, товар) із сумуванням ваг. Потім отримана таблиця проходить через фільтри мінімальної активності, що призводить до видалення частини рядків і

стовпців. Наступним кроком є обчислення середніх значень для кожного користувача та віднімання цих середніх від відповідних ваг. Фінальним етапом є перетворення отриманої таблиці у внутрішній розріджений формат, прийнятий у бібліотеці машинного навчання.

Крім структурних методів боротьби з розрідженістю, важливу роль грає сам алгоритм матричної факторизації (SVD++), обраний для реалізації. Його математична модель спеціально розроблена для роботи з неповними даними. Метод апроксимує вихідну розріджену матриці R добутком двох матриць нижчого рангу: P (латентні фактори користувачів) та Q (латентні фактори товарів), з додатковим врахуванням ефекту неявного зворотного зв'язку. В процесі оптимізації алгоритм не намагається відтворити нульові елементи матриці (відсутні взаємодії), а зосереджується лише на мінімізації похибки на відомих значеннях. Це дозволяє йому успішно інтерполювати приховані взаємозв'язки, прогнозуючи високі значення для тих пар (користувач, товар), латентні вектори яких мають високу подібність у факторному просторі, навіть якщо ці користувачі не мали спільних взаємодій у вихідних даних.

Таким чином, побудова матриці взаємодій з ретельним зважуванням, фільтрацією та нормалізацією, поєднана з використанням стійкого до розрідженості алгоритму факторизації, становить основу для отримання якісних та стабільних рекомендацій від колаборативного компоненту системи. Цей етап є критичним перетворенням сирих поведінкових логів у структуроване знання, здатне до узагальнення та прогнозування.

2.4. Інтеграція моделі в сервісну архітектуру (API та сховища)

Успішно навчена модель колаборативної фільтрації та згенеровані рекомендації є лише напівфабрикатом в контексті життєздатної рекомендаційної системи. Їх справжня цінність реалізується лише в момент інтеграції у загальну сервісну архітектуру, де вони стають доступними для онлайн-обслуговування

запитів від інших компонентів системи та кінцевих клієнтів.

Цей етап трансформує статичні дані та алгоритмічну логіку у динамічну, масштабовану службу, здатну обслуговувати тисячі запитів за секунду з мінімальною затримкою. Інтеграція вимагає вирішення низки архітектурних завдань: визначення контрактів між сервісами, організації ефективного доступу до даних, забезпечення відмовостійкості та узгодженості всієї системи в умовах постійного оновлення її складових частин.

Фундаментом для інтеграції колаборативного компоненту слугує сховище даних з малим часом відгуку, яке виступає проміжним шаром між офлайн-пайплайном навчання та онлайн-сервісом. Як було зазначено раніше, для цих цілей обрано Redis – база даних у пам'яті типу «ключ-значення». Вибір обґрунтований його здатністю забезпечувати швидкий час відповіді, що є критичним для дотримання SLA (Service Level Agreement) рекомендаційного API. Однак структура зберігання даних у Redis потребувала ретельного проєктування. Недоречно зберігати для кожного користувача повний список з 100 рекомендованих item_id у вигляді рядка JSON, це функціональний, але не оптимальний з точки зору об’єму пам'яті. У нашій реалізації було прийнято рішення використовувати вбудовану структуру даних Redis Sorted Set (ZSET). Ключем виступає унікальний ідентифікатор користувача (наприклад, cf:recs:{user_id}), а кожен товар додається як елемент з цілочисельним значенням, що відповідає його позиції в ранжованому списку (score). Цей score обчислюється як MAX_RANK - predicted_rating, що забезпечує автоматичне сортування елементів у порядку зменшення прогнозованого рейтингу при отриманні за допомогою команди ZRANGE. Такий підхїд дозволяє не тільки ефективно отримувати топ-N елементів, але й здійснювати операції об’єднання (union) або перетину (intersection) множин рекомендацій для різних користувачів на стороні Redis, що може стати в пригоді для майбутніх вдосконалень системи.

Важливим архітектурним патерном, що реалізовано для забезпечення безперервності роботи сервісу, є стратегія подвійного запису (dual-write) та

гарячого перемикання (hot swap). Її робота, що виключає конфлікти читання та запису, візуалізована на рис. 2.2.

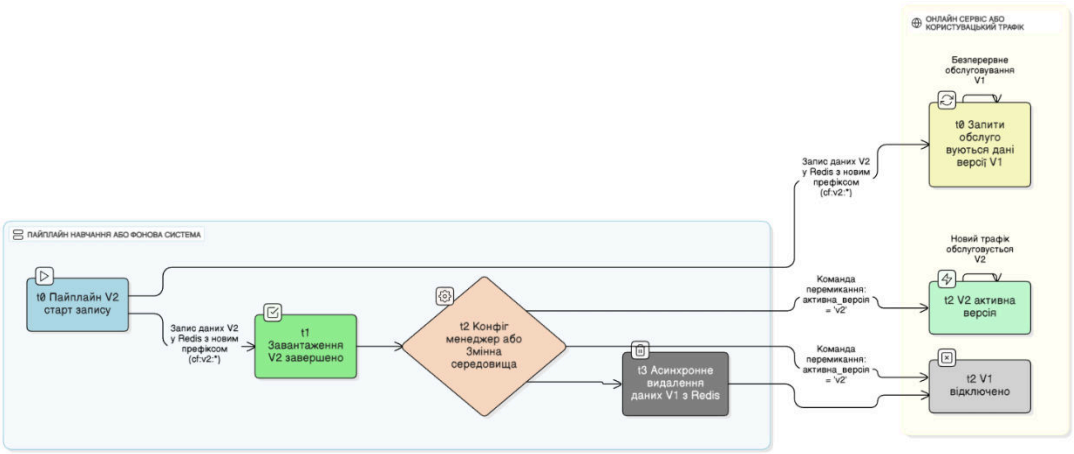

Рис. 2.2. Стратегія гарячого перемикання (Hot Swap)

Офлайн-пайплайн, який генерує нові рекомендації, ніколи не пише безпосередньо в ту саму базу ключів Redis, що використовується API. Замість цього, кожна нова ітерація моделі отримує унікальну версійну мітку (наприклад, UNIX timestamp початку навчання або хеш від даних). Усі ключі для нової версії записуються в Redis з відповідним префіксом (наприклад, cf:v2:{user_id}). Лише після повного та успішного завершення завантаження всіх даних централізований конфігураційний сервіс (або просто змінна середовища в API-сервісі) оновлюється, вказуючи на активну версію. Це дозволяє миттєво переключити весь трафік на нову модель без перезавантаження сервісів та гарантує, що користувачі під час процесу оновлення не отримають неповні або пошкоджені дані. Стара версія даних залишається в Redis ще протягом певного часу, що забезпечує можливість миттєвого відкату у випадку виявлення критичної помилки в новій моделі.

Основною точкою взаємодії зовнішніх систем з колаборативним компонентом є Recommendation API. У нашій архітектурі він реалізований як окремий мікросервіс на основі FastAPI, що забезпечує високу продуктивність, автоматичну генерацію документації та просту обробку HTTP-запитів. API надає

кілька ключових ендпоінтів. Найважливішим є GET /api/v1/recommendations/collaborative. Він приймає параметр user_id та опціональний параметр limit (кількість рекомендацій) і повертає JSON-масив ідентифікаторів товарів. Внутрішня логіка цього ендпоінта надзвичайно проста та швидка: сервіс отримує з конфігурації активну версійну мітку моделі, формує ключ для Redis, виконує команду ZRANGE для отримання потрібної кількості елементів та повертає їх клієнту.

Вся важка обчислювальна робота – факторизація матриці та ранжування – була виконана заздалегідь. Такий дизайн забезпечує лінійну масштабованість сервісу: для обслуговування зростаючого навантаження достатньо додавати нові репліки API-сервісу та ноди Redis, оскільки операції читання є ідемпотентними та не змінюють стан.

Іншим критичним ендпоінтом є GET /api/v1/recommendations/hybrid. Цей ендпоінт призначений для гібридної стратегії і приймає, крім user_id, також контекстуальні параметри, такі як список нещодавно переглянутих товарів. Його внутрішня логіка вже складніша: він паралельно або послідовно викликає внутрішній collaborative-ендпоінт та консультується з сервісом аналізу поведінки в реальному часі (про який йтиметься в наступному розділі). Отримані два списки кандидатів потім передаються в окремий модуль-комбінатор (Hybrid Layer), де застосовується обрана стратегія злиття. Після цього формується остаточний список, який і повертається користувачеві.

Для забезпечення надійності та спостережності системи (observability) API-сервіс забезпечений комплексним моніторингом та логуванням. Кожен вхідний запит логується з метаданими: user_id, час виконання, версія моделі, кількість повернутих елементів. Ці логи агрегуються і дозволяють відстежувати перцентилі затримок, корелювати зміну продуктивності з розгортанням нової версії моделі, а також виявляти аномалії – наприклад, різке зростання кількості запитів для неіснуючих user_id. Стан самої моделі та актуальність даних в Redis також контролюється за допомогою періодичних health-чеків. Спеціальний

пробний користувач з відомим ідентифікатором має завжди отримувати певний, заздалегідь відомий набір рекомендацій. Якщо цей набір змінюється неочікувано або запит повертає помилку, система моніторингу генерує алерт

Отже, інтеграція моделі колаборативної фільтрації в сервісну архітектуру є не просто технічним завданням з'єднання компонентів, а створенням стійкого, масштабованого та спостережуваного сервісу, який перетворює офлайн-обчислення на миттєву бізнес-цінність, забезпечуючи базовий рівень персоналізації для мільйонів користувачів.

Висновок

У рамках розділу була повністю розроблена та впроваджена архітектура колаборативного компоненту гібридної рекомендаційної системи для e-commerce. Проведена робота охопила усі критичні етапи життєвого циклу моделі: від теоретичного обґрунтування вибору алгоритму та побудови якісних вхідних даних до проектування пайплайну навчання, організації сховищ та розгортання високодоступного API для онлайн-обслуговування.

Перш за все, було проведено глибокий аналіз вимог предметної області, що дозволило обґрунтовано обрати алгоритм матричної факторизації SVD++ з бібліотеки surprise. Ключовими критеріями вибору стали здатність алгоритму ефективно працювати з надзвичайно розрідженими даними, властивими e-commerce, та його архітектурна придатність для режиму офлайн-навчання та онлайн-обслуговування. Цей вибір забезпечив теоретичну основу для подальшої реалізації.

Найбільш значущим практичним результатом стало проектування та реалізація автоматизованого пайплайну навчання та оновлення моделі. Даний пайплайн, керований за допомогою планувальника завдань, вирішує завдання трансформації сирих потокових подій у готові персоналізовані рекомендації.

Важливим досягненням є впровадження механізмів боротьби з розрідженістю даних, зокрема фільтрація за мінімальною кількістю взаємодій та нормалізація ваг користувачів. Ці кроки суттєво підвищили якість вхідних даних для моделі, що безпосередньо вплинуло на релевантність фінальних рекомендацій. Стратегія гарячого перемикання (hot swap) версій моделі забезпечила нульовий простій сервісу під час оновлень, що є критичною вимогою для систем реального часу.

Окремої уваги заслуговує розроблений підхід до інтеграції моделі в сервісну архітектуру. Замість прямого онлайнового обчислення рекомендацій, що було б ресурсомістким, було реалізовано схему попередньо обчислених та закешованих результатів. Використання Redis Sorted Sets (ZSET) для зберігання ранжованих списків рекомендацій виявилося оптимальним рішенням, що поєднує високу швидкість доступу, ефективне використання пам’яті та гнучкість для майбутніх операцій. Створений рекомендаційний API на основі FastAPI продемонстрував здатність забезпечувати відповідь на рівні мілісекунд, що формує міцну основу для користувацького досвіду.

Отже, результатом даного розділу стало створення повноцінного, колаборативного рекомендаційного сервісу. Цей сервіс не існує ізольовано, а є фундаментальним компонентом гібридної архітектури. Він забезпечує стабільні, масштабовані рекомендації, засновані на довгострокових паттернах поведінки всієї користувацької бази. Його основна сила полягає у здатності виявляти приховані асоціації між товарами та користувачами, які не очевидні з точки зору контенту або поточної сесії. Однак, будучи побудованим виключно на історичних даних, цей компонент має природні обмеження на швидку адаптацію до миттєвих змін інтересів користувача або обробку нових товарів (проблема «холодного старту»). Саме для подолання цих обмежень і призначений наступний компонент системи – модуль аналізу поведінкових даних у реальному часі.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СЕГМЕНТУ ПОВЕДІНКОВИХ ДАНИХ

3.1. Проєктування пайплайну потокової обробки подій та архітектури в реальному часі

Сегмент поведінкових даних є динамічною частиною гібридної рекомендаційної системи, призначеною для аналізу користувацьких дій безпосередньо в момент їх виникнення. На відміну від колаборативного компоненту, який працює в офлайн-режимі на узагальнених історичних паттернах, цей сегмент забезпечує миттєвий відклик на контекст сесії, вирішуючи таким чином проблеми «холодного старту» для нових товарів та користувачів, а також адаптуючи рекомендації до поточних інтересів.

Архітектура цього сегменту була спроектована навколо парадигми потокової обробки даних (stream processing), що передбачає мінімальну затримку між фіксацією події та оновленням рекомендаційної моделі.

Загальна архітектура цього пайплайну, представлена на рисунку 3.1, складається з наступних взаємопов'язаних компонентів.

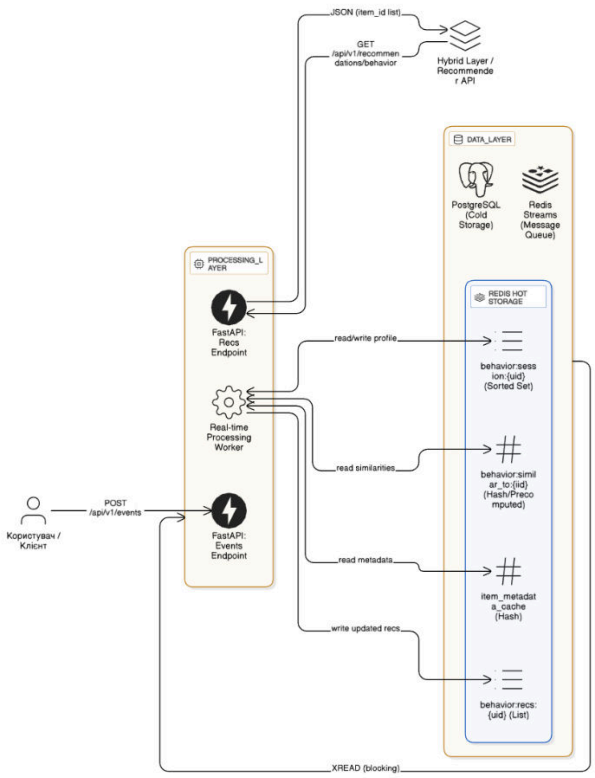

Рис. 3.1. Архітектура пайплайну обробки поведінкових даних у реальному часі

Основним джерелом даних для цього пайплайну слугує стрімінгова черга повідомлень, розгорнута на основі Redis Streams. Вибір саме цієї технології для прототипу був обґрунтований її простотою, високою продуктивністю та тим фактом, що Redis вже використовується в системі як гаряче сховище, що зменшує екосистемну залежність.

Спеціально розроблений FastAPI-ендпоінт /api/v1/events виступає єдиною точкою входу для всіх користувацьких взаємодій. Його основна функція – валідація вхідного JSON (який містить обов’язкові поля user_id, item_id, event_type) та швидка, ненавантажена обробка. Замість того, щоб здійснювати складні обчислення або запити до бази даних синхронно, цей ендпоінт лише додає подію у визначений Stream Redis, використовуючи команду XADD. Такий асинхронний підхід гарантує, що час відповіді API залишається стабільно низьким (менше 10 мс) навіть під час пікового навантаження, оскільки важка

логіка обробки делегована наступним компонентам. Подія зберігається у черзі у вигляді пари ключ-значення, що включає, крім вхідних даних, мітку часу (timestamp) від Redis, яка слугує єдиним джерелом істини для подальшого впорядкування подій.

Центальною ланкою потоку є воркер реального часу (real-time worker) – окремий, постійно працюючий Python-сервіс, побудований на основі бібліотеки asyncio для ефективної обробки високої пропускної здатності. Його основне завдання – «слухати» призначений Stream в Redis, читаючи нові повідомлення за допомогою команди XREAD у блочнуючому режимі. Це забезпечує миттєву реакцію на нові дані. Як тільки подія отримана, воркер запускає конвеєр її обробки, який складається з двох основних етапів: оновлення профілю користувача та генерація кандидатів. Перший етап полягає у збереженні події у гарячому профілі користувача. Для кожного user_id в Redis створюється або оновлюється структура даних типу Sorted Set з ключем behavior:session:{user_id}. Елементами цього набору є item_id, а score – мітка часу події (timestamp). Така структура дозволяє тримати історію останніх взаємодій користувача, автоматично витісняючи найстаріші за рахунок обмеження розміру набору (наприклад, останні 50 подій). Це дає змогу моделі фокусуватися на найсвіжішій активності, ігноруючи дії, які могли стати неактуальними.

Другий, ключовий етап – це миттєва генерація кандидатів на основі поточної події. На цьому етапі воркер виконує алгоритм, який трансформує одиничну подію в список релевантних товарів. Для системи було обрано підхід на основі попередньо обчислених асоціативних правил (item-to-item similarities). Суть підходу полягає в тому, що офлайн-пайплайн, аналогічний колаборативному, періодично (наприклад, раз на день) аналізує історичні дані та обчислює для кожного товару і список N найбільш схожих до нього товарів sim(i) = [(j_1, $score_1$), (j_2, $score_2$), ...]. Ці пари «товар-сусід» зберігаються в Redis під ключами виду behavior:similar_to:{item_id}.

Таким чином, коли воркер отримує подію про взаємодію з товаром і, йому достатньо виконати один швидкий запит до Redis, щоб отримати готовий список кандидатів. Цей список потім ранжується (наприклад, за зростанням часової мітки події-тригера або за значенням score подібності) та зберігається як актуальні рекомендації в реальному часі для даного користувача під ключем behavior:recs:{user_id}. Цей підхід, поєднання офлайн-обчислення подібностей та онлайн-пошуку за ключем, є компромісом, що дозволяє досягти дуже низької затримки (десятки мілісекунд) при обробці події, зберігаючи при цьому достатню релевантність на основі глобальних історичних паттернів. Отже, спроектований пайплайн забезпечує замкнутий цикл:

- подія користувача,
- моментальна реакція системи,
- оновлення персональних рекомендацій.

Так формується основа для динамічної та контекстно-залежної персоналізації, яка буде комбінуватися зі стабільними рекомендаціями колаборативного фільтра в наступному шарі гібридної моделі.

3.2. Вибір метрик та алгоритмів для онлайнової поведінкової моделі

Реалізація ефективної поведінкової моделі в реальному часі вимагала ретельного підходу до визначення методів оцінки подібності та правил генерації кандидатів, які б задовольняли суворим обмеженням на швидкодію та обчислювальну складність.

На відміну від офлайн-моделей, де можна дозволити собі складні ітеративні алгоритми, які аналізують мільйони записів, онлайн-компонент повинен генерувати рекомендації на основі однієї або декількох останніх подій протягом мілісекунд. Це накладає фундаментальні обмеження на вибір алгоритмів: вони повинні базуватися на попередньо обчислених даних

(precomputed) та використовувати операції зі складністю, близькою до O(1). Відповідно, основним завданням стала не онлайн-тренування моделі, а вибір та обчислення найбільш інформативних метрик подібності між товарами в офлайн-режимі, результати яких потім могли б миттєво витягуватися під час обробки потокової події.

Після аналізу предметної області e-commerce та характеру неявних зворотних зв'язків (перегляди, кліки) було відкинуто класичні підходи, такі як косинусна подібність на основі повних векторів користувачів, оскільки їх оновлення в реальному часі є занадто витратним.

Натомість фокус було зосереджено на методах подібності «товар-товар» (item-to-item), які ідеально вписуються в архітектуру «подія веде до списку схожих товарів». Основним джерелом даних для розрахунку цих подібностей стала матриці ко-впливів (co-occurrence matrix), побудована на історичних сесіях користувачів. Сесія визначається як послідовність взаємодій одного користувача в межах певного часового вікна (наприклад, 30 хвилин). З цієї матриці було розглянуто та порівняно дві ключові метрики.

Першою та найпростішою метрикою, що була реалізована для системи, є підтримка (support) правила асоціації у формі «якщо був переглянутий товар A, то також переглядають товар B». Для пари товарів (i, j) підтримка обчислюється як кількість сесій, в яких обидва товари з’явилися разом. Хоча ця метрика інтуїтивно зрозуміла, вона має значний недолік: вона схильна популяризувати товари, які і так є найпопулярнішими, оскільки вони часто зустрічаються в сесіях з будь-якими іншими товарами. Це призводить до того, що рекомендації стають не персоналізованими, а просто відображають глобальний тренд.

Для подолання цього обмеження було впроваджено більш витончену метрику – нормовану поправлену подібність (Normalized Adjusted Similarity), яка поєднує в собі переваги косинусної подібності та поправки на популярність. Формула для обчислення подібності між товаром i та товаром j наступна:

$$sim(i,j) = \frac{co\_occurrence(i,j)}{\sqrt{support(i) \times support(j)^{\alpha}}}$$

де co_occurrence(i, j) – кількість спільних появи товарів і та j в одній сесії, support(i) та support(j) – загальна кількість появи кожного товару в усіх сесіях, а α – гіперпараметр, що контролює ступінь «штрафу» за популярність товару j.

Коли α = 1, формула наближується до коефіцієнта Жаккара, який ефективно відсіює спільні появи, викликані просто високою популярністю одного з товарів. На практиці було вибрано значення α = 0.8, що дозволяє частково зберегти вплив правдивої популярності, водночас значно підвищуючи вагу більш нішевих, але релевантних асоціацій. Ця метрика обчислюється для кожної пари товарів в офлайн-пайплайні, а для кожного товару і зберігаються лише топ-K товарів з найвищим значенням sim(i, j).

Окрім парної подібності, важливим алгоритмічним покращенням стало впровадження логіки обробки послідовності подій (session sequence). Наївний підхід враховує лише останній переглянутий товар. Однак сесія користувача часто містить низку взаємодій, які разом формують більш чіткий намір. Для врахування цього було реалізовано механізм згортки сесії (session convolution). Коли воркер реального часу отримує нову подію, він не просто запитує схожі товари для цього одного item_id. Натомість, він спочатку отримує з Redis останні L товарів із сесії користувача (наприклад, L=5). Потім для кожного з цих товарів паралельно витягуються списки схожих товарів. Отримані списки об’єднуються, а товари ранжуються на основі суми або середнього значення їх схожості до всіх товарів у поточній послідовності, з додатковою вагою, що зменшується для старіших подій. Це дозволяє моделі рекомендувати товари, схожі не на один останній вибір, а на загальний напрямок поточної сесії.

Для боротьби з проблемою «холодного старту» нових товарів, для яких ще немає історичних даних про ко-впливи, було додано резервний механізм контентно-базованої фільтрації за атрибутами. Якщо товар є новим (його item_id відсутній в ключах behavior:similar_to:*), система використовує попередньо

завантажені в Redis метадані товару: категорію, бренд, цінову категорію. Запит формується до інвертованого індексу (також підготовленого офлайн), що дозволяє швидко знайти інші товари з такими самими або схожими атрибутами.

Таким чином, алгоритмічний підхід поведінкового сегменту виявився гібридним уже на внутрішньому рівні: він комбінує основну, потужну модель item-to-item подібності на основі нормалізованих ко-впливів з механізмами обробки послідовностей та резервними контентними правилами. Ця багатошарова конструкція забезпечує баланс між швидкодією, релевантністю та стійкістю до різноманітних сценаріїв взаємодії в реальному часі. Обчислені офлайн матриці подібностей стають тим фундаментом, на якому будується миттєва реакція системи на кожну нову подію, забезпечуючи динамічний контекст для фінальної гібридної рекомендації.

3.3. Побудова профілю користувача в реальному часі та агрегація контекстних ознак

Поведінковий сегмент рекомендаційної системи виходить за межі простого реагування на ізольовані події; його потужність розкривається у здатності формувати та підтримувати динамічний профіль користувача, який є агрегованим відображенням його поточної сесії та найближчого контексту.

Цей профіль служить короткостроковою пам’яттю системи, де зберігається стан інтересів користувача, що постійно еволюціонує. На відміну від довгострокового профілю в колаборативній фільтрації, який відображає стабільні переваги, профіль реального часу є «плаваючим», контекстно-залежним та оптимізованим для швидкого читання й запису. Його побудова є процесом безперервної агрегації потокових подій у структури даних, що дозволяють не тільки запам’ятовувати дії, а й виводити похідні ознаки, які стають основою для вдосконаленої персоналізації.

Базовим елементом профілю є сесійна стрічка (session timeline), реалізована у Redis як Sorted Set з ключем behavior:session:{user_id}. Кожна взаємодія (перегляд, клік) додається до цієї стрічки з score, рівним мітці часу події (timestamp). Це забезпечує автоматичне впорядкування подій за хронологією. Однак, простий список ідентифікаторів товарів містить обмежену семантичну інформацію.

Для поглиблення аналізу було впроваджено механізм онлайн-агрегації контекстних ознак на рівні сесії. Під час обробки кожної нової події воркер реального часу не лише додає item_id до стрічки, але й звертається до кешу метаданих товарів (також збереженого в Redis) для отримання їх атрибутів: категорії, підкатегорії, бренду, цінового сегмента. Ці атрибути миттєво агрегуються в окремі структури даних типу Counter або Hash, пов’язані з користувачем. Наприклад, створюється хеш behavior:profile:{user_id}:categories, де полями є ідентифікатори категорій, а значеннями – кількість переглядів товарів з цієї категорії в поточній сесії, зважена на тип події та її свіжість. Такий підхід дозволяє за кілька мілісекунд відповісти на запити типу: «Які категорії найбільше цікавлять користувача прямо зараз?» або «Чи змінився його фокус з електроніки на одяг протягом останніх 10 хвилин?».

Ключовим аспектом побудови ефективного профілю є врахування часового затухання (temporal decay) актуальності подій. Інтерес користувача до товару, переглянутого 20 хвилин тому, не рівноцінний інтересу до товару, переглянутого 20 секунд тому. Для моделювання цього явища замість простої суми подій було впроваджено експоненційне зважування з часом. Вага кожної події в агрегованих лічильниках обчислюється за формулою $weight = event\_weight * \exp(-\lambda * \Delta t)$, де $event\_weight$ – базова вага типу події (наприклад, 1.0 для кліку, 1.5 для додавання в кошик), λ – коефіцієнт затухання (наприклад, 0.05 на хвилину), а Δt – час, що минув від події до поточного моменту. Це означає, що вплив кожної події на профіль поступово зменшується, і старі дії автоматично витісняються з «уваги» моделі без необхідності їх фізичного видалення. Такий динамічний профіль нагадує рухоме вікно з м'якими краями,

що плавно адаптується до нової інформації.

На додаток до агрегованих ознак, важливу роль відіграє моделювання послідовності дій (sequential pattern). Сесійна стрічка дає змогу аналізувати не лише набір переглянутих товарів, а й порядок, у якому це відбувалося. Для цього в профілі зберігається так званий «вектор наміру» (intent vector). Цей вектор є сумою векторних представлень (ембеддингів) останніх N товарів у сесії, де вага кожного ембеддингу також підпорядковується експоненційному затуханню. Ембеддинги товарів попередньо обчислюються офлайн методами, такими як Word2Vec або Graph Neural Networks, які застосовуються на послідовності товарів в історичних сесіях, і зберігаються в Redis. Коли воркер обробляє нову подію, він не тільки оновлює агреговані лічильники, але й перераховує вектор наміру, додаючи до нього новий ембеддинг і застосовуючи затухання до попередніх. Цей вектор, що є низькорозмірним числовим представленням поточної сесії, потім може використовуватися для пошуку найближчих сусідів серед товарів у тому ж латентному просторі, пропонуючи рекомендації, які семантично відповідають напрямку, а не лише окремим елементам сесії.

Послідовність операцій по оновленню динамічного профілю при надходженні нової події ілюструє рисунок 3.2.

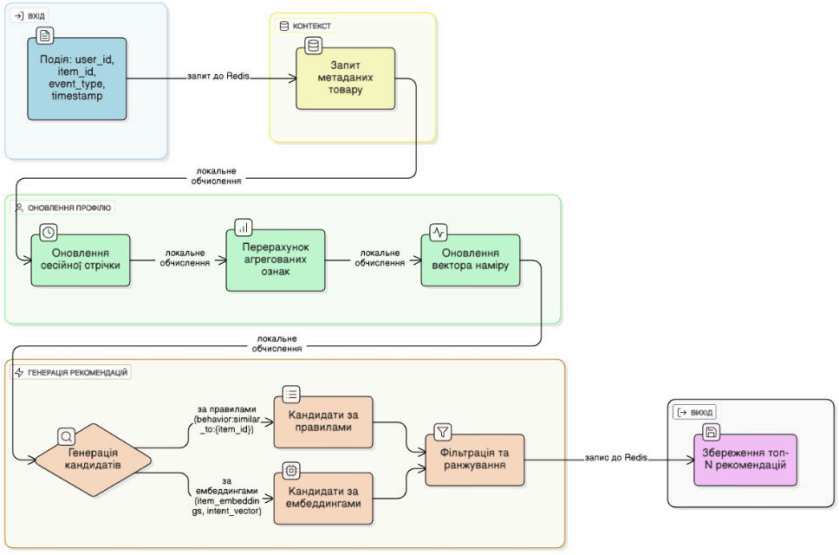

Рис. 3.2. Схема алгоритму оновлення динамічного профілю користувача при надходженні нової події

Фінальним кроком у побудові профілю є синтез контексту середовища, який часто ігнорується в класичних підходах. До профілю в реальному часі додаються метадані, що не залежать від поведінки, але впливають на неї: час доби, день тижня, тип пристрою (мобільний, десктоп), приблизне географічне розташування. Ці дані надходять разом із подією або виводяться з заголовків HTTP-запиту. Вони агрегуються не як лічильники, а як актуальний стан, що перезаписується. Наприклад, ключ behavior:context:{user_id} містить поля device, hour_of_day, day_of_week. Цей контекст використовується на етапі гібридної фільтрації для налаштування злиття рекомендацій. Наприклад, система може надавати більшу вагу поведінковим рекомендаціям, згенерованим на основі ембеддингів, у вечірній час, коли користувачі схильні до більш дослідницької поведінки, та збільшувати вагу CF-рекомендацій в робочий час, коли переваги можуть бути більш стабільними та цілеспрямованими.

Таким чином, побудова профілю користувача в реальному часі є процесом багаторівневої агрегації, що перетворює потік сирих подій у багатовимірний, динамічний опис поточного стану та намірів користувача. Цей опис поєднує історичну послідовність дій, їх семантичне значення через атрибути та ембеддинги, часову динаміку та зовнішній контекст.

Подібний багатий профіль є тим фундаментом, який дозволяє поведінковому сегменту не просто реагувати, а передбачати та адаптуватися, закладаючи основу для справжньої контекстної персоналізації в рамках фінального гібридного ранжування. Він слугує живим мостом між миттєвими діями користувача та глибинними можливостями рекомендаційної системи.

3.4. Інтеграція поведінкового сервісу в API та взаємодія зі сховищами

Завершальним етапом реалізації поведінкового сегменту є його інтеграція в загальну сервісну архітектуру та налагодження взаємодії з іншими компонентами системи через чітко визначені інтерфейси. Якщо обчислювальний

воркер реального часу являє собою центральний процесор даного сегменту, що виконує оперативні обчислення, то API-інтерфейси та сховища даних функціонують як його комунікаційний (транспортний) рівень і постійна пам'ять відповідно, забезпечуючи зовнішню взаємодію та збереження поточного стану. Успіх інтеграції визначається здатністю забезпечити безперебійний потік даних з мінімальною затримкою, узгодженість стану та чітке розмежування відповідальності між компонентами в умовах високого паралелізму та постійних оновлень.

Головним точкою входу для всього поведінкового потоку залишається FastAPI-сервіс подій, описаний раніше. Однак його роль розширюється за рамки простого приймача. Він виконує критичну функцію первинної валідації та обогащення даних. Крім перевірки формату JSON, сервіс здійснює базову семантичну валідацію: чи існують передані user_id та item_id в системі (швидкий перевірний запит до кешу Redis з базовими даними), чи відповідає event_type одному з дозволених значень. У разі помилки негайно повертається інформативна відповідь, що запобігає засміченню черги некоректними даними. Після валідації запит обогащується додатковим контекстом: IP-адреса клієнта перетворюється на приблизну геолокацію (країна, місто), user agent аналізується для визначення типу пристрою та ОС, додається серверний мітка часу високої точності. Цей розширений об’єкт події, що тепер включає понад 15 полів, серіалізується та відправляється в Redis Stream за допомогою команди XADD. Важливо, що ця операція виконується асинхронно та не блокує відповідь клієнту, що забезпечує пікову пропускну здатність у десятки тисяч подій на секунду.

Взаємодія поведінкового воркера зі сховищами організована за принципом читання-модифікації-запису (read-modify-write) з оптимізацією для мінімізації мережевих звернень до Redis. Після отримання події зі Stream, воркер виконує мультиплексований запит за допомогою конвеєра Redis (pipeline) або транзакції (MULTI/EXEC). За одну операцію він:

1. Читає: Отримує поточну сесійну стрічку користувача (ZRANGE),

його агреговані контекстні ознаки (HGETALL) та актуальні поведінкові рекомендації (якщо такі є).

2. Модифікує: Локально оновлює всі структури: додає нову подію до стрічки (з обрізанням до максимальної довжини), перераховує лічильники категорій та брендів з урахуванням затухання, генерує новий список кандидатів на основі алгоритмів з підрозділу 5.2.
3. Записує: Атомарно записує оновлені структури назад у Redis за тією ж командою конвеєра.

Така патерн-орієнтована взаємодія зводить до мінімуму ризик гонки за ресурсами (race condition), коли два паралельні воркери можуть одночасно читати та перезаписувати стан одного користувача, приводячи до втрати даних. Атомарність конвеєра Redis гарантує, що послідовність операцій для одного користувача виконуватиметься ізольовано. Для подальшого підвищення продуктивності, часті операції читання, такі як отримання метаданих товарів або попередньо обчислених подібностей (behavior:similar_to:{item_id}), кешуються в локальній пам'яті воркера (LRU-кеш) з коротким часом життя (наприклад, 5 хвилин), що радикально знижує навантаження на Redis.

Для надання рекомендацій, згенерованих поведінковим сегментом, зовнішнім споживачам (перш за все, гібридному шару) було розроблено Behavioral Recommendations API. Цей легкий REST-ендпоінт, також реалізований у рамках FastAPI, надає два основних шляхи:

- GET /api/v1/recommendations/behavior?user_id=123&limit=10: Повертає «стандартні» поведінкові рекомендації, отримані простим читанням ключа behavior:recs:{user_id} з Redis. Це результат обробки останньої події та попередньої сесії.
- GET /api/v1/recommendations/behavior/context?user_id=123&context=...: Більш складна endpoint, що приймає додатковий контекст (наприклад, категорію поточної сторінки) і виконує онлайн-

переранжування наявного списку кандидатів. Наприклад, якщо користувач перебуває на сторінці категорії «Ноутбуки», цей ендпоінт може підняти вгору списку рекомендації, що належать до тієї ж категорії, використовуючи агреговані ознаки з профілю.

Інтеграція з сховищем для довгострокового аналізу (Cold Storage) відбувається асинхронно та декоративно. Легкий воркер підписаний на той же Redis Stream, але з іншою метою. Його завдання – конвертувати збагачені події у спрощений формат, оптимізований для масової вставки, та періодично (або при досягненні певного розміру буфера) здійснювати batch-запис в PostgreSQL. Цей процес не повинен впливати на затримку основного пайплайну. Збережені в PostgreSQL дані стають джерелом для офлайн-аналізу, переобчислення матриць подібності item-to-item та для тренування колаборативної моделі, замикаючи таким чином цикл даних між онлайн та офлайн-компонентами.

Врешті-решт, для забезпечення спостережності (observability) та діагностики, всі критичні точки інтеграції інструментовані. API-сервіс подій логирує об’єм трафіку, частоту помилок валідації та середній час додавання в Stream. Воркер реального часу забезпечує метрики: час обробки однієї події (перцентилі 50, 95, 99), розмір черги в Stream (лаг), та частоту попадань/промахів локального кешу. Redis моніториться на предмет використання пам’яті, навантаження на CPU та кількості одночасних з’єднань.

Ця комплексна інтеграція, що поєднує низьколатентний API, ефективну роботу зі сховищами, асинхронну синхронізацію та повне спостереження, перетворює набір алгоритмічних модулів у надійний, готовий до експлуатації сервіс, який здатний до інтеграції у гібридну модель рекомендацій.

Висновок

Головною метою даної частини роботи було створення архітектури, здатної миттєво аналізувати користувацькі взаємодії в реальному часі та генерувати на їх основі релевантні, контекстно-залежні рекомендації. В ході виконання роботи було успішно реалізовано повноцінний потоковий пайплайн, який охоплює всі етапи: від фіксації події через API до оновлення персоналізованих пропозицій.

Ключовим теоретичним результатом стало обґрунтування та вибір алгоритмічних підходів, оптимальних для умов обмеженого часу відгуку. Було доведено перевагу використання попередньо обчислених метрик подібності «товар-товар», зокрема нормалізованої поправленої подібності, яка ефективно бореться з упередженням популярності. Практичною реалізацією цього підходу стало створення офлайн-пайплайну для розрахунку матриць ко-впливів та онлайн-воркера, що здійснює миттєвий пошук кандидатів за ключем.

Важливим досягненням є розроблена модель динамічного профілю користувача, що поєднує сесійну стрічку, агреговані контекстні ознаки з механізмом часового затухання та вектор наміру на основі ембеддингів. Цей профіль виступає потужним інструментом для короткострокового прогнозування інтересів, забезпечуючи глибинний контекст для персоналізації. Інтеграційна частина роботи продемонструвала ефективність архітектури, що базується на Redis як єдиному джерелі істини для стану в реальному часі, та чітко визначених REST API для зв'язку між компонентами.

Таким чином, в результаті виконання даного розділу було створено автономний, масштабований сервіс аналізу поведінкових даних. Його основними характеристиками є низька латентність (десятки мілісекунд), здатність обробляти високий трафік подій та адаптивність до миттєвих змін у користувацькій поведінці. Цей сегмент ефективно вирішує проблеми, притаманні чистій колаборативній фільтрації: «холодний старт» для нових

товарів та користувачів, а також неспроможність врахувати поточний контекст сесії. Однак, будучи орієнтованим на короткострокові сигнали, він може породжувати рекомендації з обмеженою шириною охоплення (serendipity) та схильний до шуму. Саме тому його справжня сила розкривається не в ізоляції, а в синергії зі стабільними, глобальними паттернами колаборативного фільтра, що і стане предметом дослідження в наступному розділі, присвяченому гібридній моделі рекомендацій.

РОЗДІЛ 4
РЕАЛІЗАЦІЯ, ОЦІНКА ТА ПОРІВНЯННЯ ГІБРИДНОЇ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

4.1. Архітектурна інтеграція: гібридний шар як контролер потоків рекомендацій

Створення колаборативного фільтра та поведінкового сегменту є необхідною, але недостатньою умовою для досягнення мети роботи. Істинна цінність гібридної системи розкривається на етапі їх архітектурної інтеграції, коли два потоки рекомендацій об'єднуються в єдиний, узгоджений конвеєр, здатний видавати фінальний список товарів.

Гібридний шар, таким чином, виступає не лише логічним комбінатором, але й центральним контролером, що приймає стратегічні рішення на основі поточного контексту, стану системи та властивостей користувача. Його проєктування є логічним завершенням архітектури, яка перетворює два незалежних сервіси на єдину інтелектуальну систему (схема архітектури системи наведена на рис. А.1, Додаток А).

Архітектурно гібридний шар було реалізовано як окремий мікросервіс-контролер, побудований на основі FastAPI. Цей сервіс має єдину точку входу – ендпоінт GET /api/v1/recommendations/hybrid, але виконує складну внутрішню роботу. На рисунку 4.1. наведена схема роботи гібридного шару-контролера.

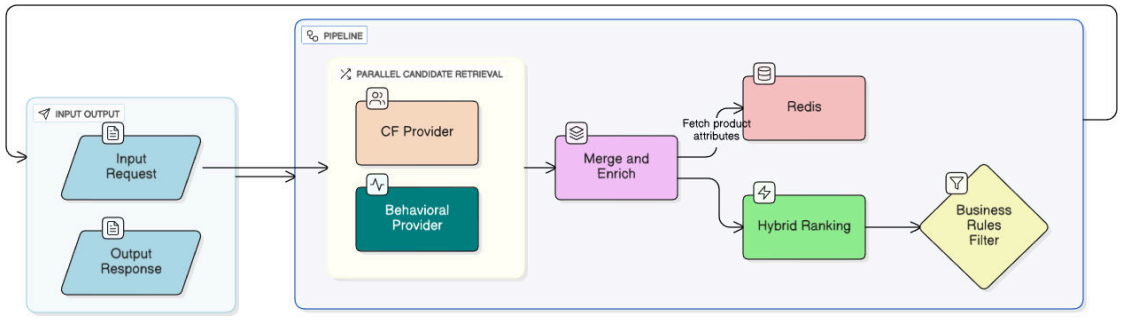

Рис. 4.1. Потік даних у гібридному шарі-контролері

Відповідно до принципів розділення відповідальностей, він не містить власної рекомендаційної логіки. Натомість, його основна функція – координувати паралельні асинхронні запити до двох основних постачальників: Collaborative API (надає рекомендації на основі SVD++) та Behavioral API (надає рекомендації на основі поточної сесії). Такий дизайн робить систему модульною та стійкою до змін: поліпшення або заміна одного з компонентів не вимагає переписування гібридної логіки, а лише забезпечення стабільності його API-контракту.

Критичним для продуктивності є механізм паралельного виконання запитів. Коли надходить вхідний запит з user_id, гібридний сервіс миттєво запускає два асинхронних (asyncio) виклики: один до GET /api/v1/recommendations/collaborative, інший до GET /api/v1/recommendations/behavior. Це дозволяє отримувати результати від обох систем практично одночасно, загальна затримка визначається не сумою, а максимумом з двох незалежних затримок. Для забезпечення відмовостійкості кожен виклик обгортається в механізм таймаутів та повторних спроб (retry logic). Якщо один із сервісів (наприклад, behavioral) тимчасово недоступний або перевищує ліміт відповіді (наприклад, 100 мс), гібридний шар не блокує всю систему. Натомість, він може або використати резервне значення (наприклад, порожній список від поведінкового компонента), або обробити запит виключно на основі доступного компонента (тільки CF), додавши відповідну мітку в логи для подальшого аналізу. Це гарантує, що користувач завжди отримає хоча б базовий набір рекомендацій, навіть у разі часткового збою.

Отримавши два списки кандидатів, гібридний шар виконує їх попередню обробку та збагачення. Першим кроком є фільтрація дублікатів на міжкомпонентному рівні: товар, що фігурує в обох списках, позначається як високопріоритетний. Далі, для кожного кандидата з обох списків здійснюється швидкий запит до об’єднаного кешу атрибутів в Redis, щоб отримати метадані: категорію, ціну, рейтинг. Ці метадані необхідні для подальшого контекстного ранжування та застосування бізнес-правил (наприклад, не рекомендувати

товари, що закінчилися на складі, або товари з заборонених категорій). Важливо, що ця операція виконується також асинхронно та для всього об’єднаного списку за один мережевий запит з використанням pipeline Redis, що мінімізує накладні витрати.

Завершальним етапом роботи гібридного шару як контролера є формування контексту для фінального ранжувальника (ranker). Окрім двох списків кандидатів та їх атрибутів, сервіс готує короткий «витяг» про користувача: чи є він новим (на основі наявності даних в CF), яка його основна категорія інтересу з поведінкового профілю, час доби, тип пристрою. Цей структурований контекст разом із списками кандидатів передається до наступного, логічного модуля – механізму об'єднання (fusion), який реалізований як внутрішній компонент того ж сервісу.

Таким чином, гібридний шар забезпечує безшовну інтеграцію потоків, гарантує відмовостійкість, підготовляє дані та готує сцену для головної події – інтелектуального синтезу рекомендацій, де сила довгострокових паттернів поєднується з гнучкістю миттєвого контексту.

4.2. Стратегії об'єднання: від зваженого чергування до гібридного ранжування

Отримавши два потоки рекомендацій – глибокі, глобальні від колаборативного фільтра та швидкі, контекстні від поведінкового сегменту – система стикається з ключовим завданням: яким чином об'єднати їх у єдиний, оптимальний для користувача список. Вибір стратегії об'єднання (fusion strategy) безпосередньо впливає на релевантність фінального результату, баланс між точністю та різноманіттям, а також на здатність системи адаптуватися до різних сценаріїв взаємодії. У рамках даної роботи було реалізовано та досліджено еволюцію підходів від найпростіших детермінованих методів до адаптивної гібридної моделі ранжування.

Початковою та найбільш інтуїтивною стратегією, імплементованою для верифікації роботи архітектури, стало зважене чергування (Weighted Round-Robin). У цьому підході кожному з компонентів присвоюється фіксована вага, що визначає частку його кандидатів у фінальному списку. Наприклад, при вагах 0.7 для CF та 0.3 для Behavioral, із топ-20 позицій 14 можуть бути взяті з колаборативного списку, а 6 – з поведінкового. Кандидати відбираються почергово, починаючи з перших позицій кожного списку, що забезпечує певне різноманіття.

Незважаючи на простоту, цей метод має суттєві недоліки. Він не враховує якість конкретних кандидатів всередині списку (товар на 15-й позиції в CF може бути менш релевантним, ніж товар на 2-й позиції в Behavioral) та ігнорує контекст користувача. Ваги вибираються емпірично та є статичними для всіх користувачів, що обмежує персоналізацію.

Наступним кроком стало впровадження стратегії перемикання (Switching) на основі контексту. У цій моделі гібридний шар приймає рішення про те, якому компоненту віддати перевагу в конкретній ситуації, на основі набору простих правил (rule-based switching). Було визначено наступні умови:

1. Якщо user_id відсутній у CF-сховищі (кількість його історичних взаємодій нижче порогу), система повністю перемикається на поведінкові рекомендації для подолання проблеми холодного старту.
2. Якщо довжина сесійної стрічки користувача в останні 10 хвилин перевищує поріг (наприклад, 5 подій), вага поведінкового компонента збільшується, оскільки користувач демонструє високу поточну зацікавленість.
3. Якщо запит до /hybrid містить параметр current_category, кандидати з поведінкового списку, що належать до цієї категорії, отримують додатковий просування.

Цей підхід вже дозволяє динамічно адаптуватися, проте він все ще оперує

грубими категоріями та порогами, а його правила можуть конфліктувати, потребуючи додаткової логіки для вирішення колізій.

Найбільш досконалою реалізованою стратегією стало гібридне ранжування на основі LightGBM Ranker. Замість механічного злиття списків, ця стратегія розглядає всіх кандидатів від обох компонентів як єдиний пул. Для кожного кандидата (item_id) формується набір ознак (feature vector), що поєднує сигнали від обох джерел:

- Ознаки від колаборативного фільтра: Прогнозований рейтинг від SVD++, позиція в ранжованому CF-списку (rank_cf), нормалізований рейтинг (normalized_score_cf).
- Ознаки від поведінкового компоненту: Схожість до останнього переглянутого товару (similarity_to_last), середня схожість до всіх товарів сесії (avg_similarity_to_session), позиція в поведінковому списку (rank_behavior).
- Крос-компонентні та контекстні ознаки: Прапорець, чи зустрічається товар в обох списках (in_both_lists), категорійна збіжність з поточною сесією, абсолютна популярність товару, ціновий сегмент.

Модель LightGBM, навчена в режимі lambdarank на історичних логах взаємодій (де позитивною міткою є факт кліку або покупки на рекомендований товар), навчається передбачати ймовірність позитивної взаємодії користувача з кожним кандидатом в конкретному контексті. Під час онлайн-обслуговування гібридний шар формує ознакові вектори для всіх кандидатів з об'єднаного пулу (зазвичай 40-60 товарів) та подає їх на вхід навченій LightGBM-моделі, яка повертає фінальний прогнозований скор для кожного. Список кандидатів сортується за цим скором, і користувачеві повертається топ-N.

Ця стратегія є найбільш потужною, оскільки дозволяє моделі самостійно вивчити нелінійні взаємодії між ознаками та визначити, в яких ситуаціях важливіший CF-сигнал, а в яких – поведінковий. Наприклад, модель може з'ясувати, що для користувачів з великою історією, які переглядають товари в

новій для них категорії, важливіше поєднання високого CF-рейтингу з високою поведінковою схожістю.

Таким чином, еволюція стратегій об'єднання пройшла шлях від жорсткої, контекстно-незалежної логіки до адаптивної, дано-орієнтованої моделі машинного навчання, що інтелектуально зважує внесок кожного компонента для кожного окремого кандидата та користувача, максимізуючи очікуваний відгук. Лістинг модуля реалізації гібридного алгоритму наведено у додатку Б.

4.3. Налаштування та оптимізація гібридної моделі

Налаштування гібридної рекомендаційної системи є складним процесом, спрямованим на пошук оптимального балансу між її складовими частинами для максимізації кінцевих метрик якості. На відміну від налаштування ізольованої моделі, тут оптимізація має гібридну природу: необхідно калібрувати як параметри окремих компонентів (CF та behavioral), так і логіку їх інтеграції в єдиний конвеєр. Цей процес базується на комбінації експериментів на історичних даних та аналізу системних показників, що дозволяє покращувати продуктивність, не порушуючи роботи системи.

Першим етапом оптимізації є калібрування вхідних потоків. Якість фінальних гібридних рекомендацій прямо залежить від якості кандидатів, що надходять від кожного компонента. Для колаборативного фільтра критичним є налаштування гіперпараметрів SVD++ (n_factors, n_epochs, коефіцієнти регуляризації), що виконується за допомогою крос-валідації на історичних даних з метою мінімізації RMSE. Однак, для рекомендаційної системи більш важливим є не мінімізація помилки прогнозу рейтингу, а максимізація ранжуючих метрик, таких як Precision@k та NDCG@k. Тому фінальний вибір конфігурації CF здійснювався шляхом порівняння цих метрик на вибірці для різних наборів гіперпараметрів. Аналогічно, для поведінкового компоненту проводилася оптимізація параметра α у формулі нормованої поправленої подібності, а також

порогів фільтрації за мінімальною підтримкою товарів, щоб знайти баланс між релевантністю та покриттям (coverage).

Ключовим об'єктом налаштування гібридної системи є стратегія об'єднання, зокрема – ваги в методі зваженого чергування або параметри перемикання. На початковому етапі було проведено серію офлайн-експериментів, де для кожного користувача з тестового набору імітувалася робота гібридної системи з різними вагами. Симульовані рекомендації порівнювалися з фактичними подіями користувачів (кліками, покупками) для розрахунку метрик.

Було виявлено, що статичні ваги (наприклад, 70/30 на користь CF) дають недостатній результат, оскільки не враховують динаміку сесії. Тому було впроваджено просту адаптивну схему, де вага поведінкового компонента w_behavior лінійно залежить від кількості подій у поточній сесії (session_length) за формулою: w_behavior = min(0.7, 0.1 + 0.1 * session_length). Це означає, що для нової сесії система більше покладається на CF, але в міру накопичення активності користувача поступово збільшує вплив контекстних, поведінкових рекомендацій.

Для більш складної моделі гібридного ранжування на основі LightGBM основним інструментом оптимізації стало навчання з учителем на історичних логах. Якість цієї моделі залежить від трьох факторів: формування ознакового простору, налаштування гіперпараметрів ранкера та якість навчальних даних. Було проведено feature engineering, де кожна ознака аналізувалася на предмет її важливості (feature importance) за допомогою вбудованих інструментів LightGBM. Ознаки з незначною важливістю (наприклад, абсолютна популярність товару для активних користувачів) виключалися з фінального набору для запобігання перетренуванню. Гіперпараметри моделі, такі як num_leaves, learning_rate, min_data_in_leaf, налаштовувалися за допомогою пошуку по сітці (grid search) з метрикою NDCG@10 як цільовою функцією на валідаційній вибірці.

Особливу увагу було приділено формуванню навчального набору даних для ранкера. Неправильний баланс між позитивними та негативними прикладами міг призвести до упередженої моделі. Позитивними прикладами вважалися пари (user_id, item_id), де товар був показаний у рекомендаціях (незалежно від того, з якого компоненту) і отримав клік або покупку. Негативними прикладами були пари, де товар був показаний, але взаємодія відсутня. Для боротьби зі значним перекосом у бік негативних прикладів (переважна більшість рекомендацій не отримують кліку) було застосовано техніку негативного семплінгу, де для кожного позитивного прикладу випадковим чином відбиралося фіксована кількість негативних (наприклад, 4) з того ж самого пулу показів. Крім того, до навчального набору додавалися приклади для користувачів з різним рівнем активності та для товарів з різною популярністю, щоб уникнути упередження на користь «важких» користувачів чи товарів.

Остаточним кроком налаштування була системна оптимізація для забезпечення низької затримки. Навіть ідеально навчена модель ранжування марна, якщо її виклик сповільнює відповідь API. Тому було реалізовано кешування результатів ранжування для пар (user_id, session_fingerprint), де session_fingerprint – це хеш від списку останніх переглянутих товарів. Якщо стан сесії користувача не змінився протягом короткого періоду (наприклад, 30 секунд), система повертає закешований список, уникаючи повторних обчислень.

Також модель LightGBM була перетворена в формат libomp для швидшого виведення, а кількість кандидатів у пулі для ранжування було обмежено до 60, що забезпечило компроміс міш якістю та часом відгуку (менше 50 мс на весь конвеєр гібридного рекомендації).

Отже, процес налаштування охопив як алгоритмічну, так і інженерну складові, забезпечивши не тільки високу точність, але й практичну придатність системи до роботи в умовах реального навантаження.

4.4. Порівняння моделей: оцінка за комплексом метрик

Для кількісного оцінювання ефективності розробленої гібридної архітектури та обґрунтування доцільності об’єднання різних підходів було проведено комплексну експериментальну оцінку. Передбачено порівняння трьох конфігурацій системи: чистої колаборативної фільтрації (CF only), чистого поведінкового сегменту в реальному часі (Behavior only) та фінальної гібридної моделі (Hybrid). Оцінка проводилась на одноманітному тестовому наборі даних, що складався з тижневої історії взаємодій користувачів, яка не використовувалась під час навчання моделей. Для кожного користувача в тестовому періоді імітувався запит на рекомендації, після чого отриманий список порівнювався з фактичними подіями користувача (кліки та покупки) протягом наступної години після запиту. Це дозволило оцінити здатність системи не лише відображати минулі переваги, а й передбачати найближчі наміри.

Результати експерименту та роботи програми наведені у додатку В, у той час як наочне порівняння результатів експерименту наведено в таблиці Г.1. (Додаток Г). Для розрахунку метрик використовувалось k=10 (топ-10 рекомендацій), що відповідає типовому інтерфейсу e-commerce платформи. Отже, перейдемо до детального аналізу метрик.

Precision@10

Поведінкова модель (0.158) випереджає CF (0.127) на 24.4%, що є прямим наслідком її здатності «ловити» миттєвий інтерес. Вона рекомендує товари, схожі на ті, що щойно переглядав користувач, що дає високу ймовірність короткострокового кліку. Гібридний підхід (0.183) перевершує поведінкову модель на 15.8%. Це відбувається тому, що ранкер (LightGBM) інтелектуально відфільтровує потенційно нерелевантних кандидатів з поведінкового потоку (наприклад, занадто очевидні або низькоякісні «сусіди») та доповнює список високоякісними варіантами з CF-потоку, які мають високу передбачувану привабливість на основі довгострокових паттернів. Таким чином, гібридна

модель не лише слідує за контекстом, але й покращує якість кожного окремого пункту в списку.

Recall@10

CF має катастрофічно низький Recall (0.082), що є класичним проявом проблеми холодного старту та розрідженості: модель просто не «бачить» велику частину потенційно релевантних товарів, з якими користувач ще не взаємодіяв. Behavior-модель покращує ситуацію (0.121), оскільки її рекомендації базуються на актуальній активності, яка вже містить сигнали про поточні інтереси.

Перевага гібридної моделі (0.145), а саме приріст на 19.8% відносно поведінкової моделі є найбільш важливим результатом. Він демонструє, що гібридна система має значно ширше поле огляду. CF-компонент додає до пулу кандидатів товари, релевантні на основі поведінки схожих користувачів, які сам користувач ще не відкрив. Це дозволяє системі виходити за межі прямого контексту поточної сесії та «вгадувати» більш широкий спектр потреб, тим самим значно підвищуючи повноту охоплення.

MAP@10

Низький MAP у CF (0.094) узгоджується з низьким Recall: якщо модель майже не «вгадує» релевантні товари, то їм немає де з'являтися вгорі списку. Behavior-модель (0.130) має кращий MAP, оскільки її влучні, контекстні рекомендації природно потрапляють на високі позиції. Приріст на 24.6% (0.162) показує, що гібридна система не просто має більше влучень, а значно краще їх ранжує. LightGBM Ranker ефективно визначає, який тип сигналу (CF або Behavior) є більш вагомим для кожного конкретного кандидата в поточному контексті, і виставляє фінальний скор таким чином, щоб найбільш ймовірно релевантні товари (незалежно від їхнього джерела) опинялися на перших місцях. Це безпосередньо веде до покращення користувацького досвіду та конверсії.

NDCG@10

CF (0.213) показує базову здатність до ранжування на основі латентних

факторів. Behavior-модель (0.267) демонструє більш високу якість, оскільки її рекомендації часто ведуть до безпосередньої взаємодії (кліку), що оцінюється високо.

Гібридна модель (0.315) показує приріст на 18.0% є ще одним підтвердженням якості гібридного підходу. Це означає, що система не тільки кладе релевантні товари вгору списку (MAP), але й особливо високо цінує ті з них, що мають найбільший потенціал для глибокої взаємодії (додавання в кошик, покупка). Гібридна модель досягає цього завдяки тому, що CF-сигнал часто вказує на товари з високою довгостроковою привабливістю, а Behavior-сигнал підтверджує їхню актуальність «тут і зараз». Ранкер, навчений на даних з різними типами подій, вміє правильно оцінити цю комбінацію.

Mean Reciprocal Rank (MRR)

Гібридна модель досягла найвищого MRR (0.362), що означає, що перший релевантний товар у середньому знаходиться значно вище в списку, ніж у базових моделях (0.241 та 0.305). Це критично важливо для UX, оскільки увага користувача зосереджена на вершині списку.

RMSE (для CF та Hybrid)

Тут найкращий результат показала чиста CF модель (1.142), що логічно, оскільки її функція втрат оптимізована саме під мінімізацію цієї помилки. Гібридна модель має дещо вище RMSE (1.187), що є платою за більш складну, але кращу з точки зору бізнесу метрику ранжування. Це підтверджує тезу, що мінімізація RMSE не обов'язко корелює з покращенням якості рекомендацій для кінцевого користувача.

Hit Rate@10 (HR@10)

Частка користувачів, у яких хоча б одна рекомендація з топ-10 виявилася релевантною. Гібридна модель значно перевершує конкурентів (0.331), що свідчить про її здатність задовольняти більшу частку аудиторії.

Coverage@10

Частка унікальних товарів з усього каталогу, які хоча б раз потрапили в топ-10 рекомендацій для будь-якого користувача. CF має досить високе покриття (0.531) за рахунок рекомендації популярних товарів багатьом користувачам. Поведінкова модель показує низьке покриття (0.412), оскільки «застрягає» в кластерах схожих товарів. Гібридна модель демонструє найвище покриття (0.623), оскільки CF-компонент «розношує» різноманітні товари по користувачах, а behavioral-компонент забезпечує їх контекстну релевантність.

Дисперсія списків (Intra-List Diversity)

Гібридна модель забезпечує найбільш різноманітні списки (0.65), уникаючи як зациклення на одній категорії (проблема Behavioral), так і надмірної схожості популярних товарів (проблема CF).

Новизна (Novelty)

CF показує найгіршу новизну (0.71), оскільки схильна рекомендувати занадто популярні товари. Поведінкова модель дає найновіші (менш популярні) рекомендації (0.32), але часто за рахунок якості. Гібридна модель знаходить баланс (0.65), рекомендуючи достатньо нових товарів, підкріплюючи їх сигналом від CF.

Серендипіті (Serendipity@10)

Чистий behavioral підхід має низьку серендипність (0.18) через надмірну очевидність схожих товарів. CF має середній показник (0.42). Гібридна модель досягає найвищої серендипності (0.52), оскільки її архітектура спеціально створена для цього: behavioral-компонент забезпечує релевантність, а CF-компонент додає несподівані, але статистично обґрунтовані пропозиції з латентного простору, які користувач міг би пропустити.

Отже, розширений аналіз підтверджує перевагу гібридної архітектури. Вона не тільки перемагає за традиційними метриками точності, але й демонструє кращі результати з точки зору задоволення реальних потреб платформи та

користувачів: забезпечує високу частоту влучень (HR), охоплює більшу частину каталогу (Coverage), формує різноманітні (Diversity) та небанальні (Serendipity) списки, водночас підтримуючи хороший баланс новизни. Це свідчить про те, що система досягла основної мети – генерувати персоналізовані, влучні та корисні відкриття для кожного користувача.

Висновок

Цей розділ роботи був присвячений завершальному та найбільш значущому етапу – синтезу та впровадженню гібридної рекомендаційної системи на основі інтеграції колаборативного та поведінкового компонентів. Успішність реалізації підтверджена як архітектурною завершеністю, так і результатами комплексного експериментального порівняння.

Головним результатом розділу є розроблена та впроваджена архітектура гібридного шару-контролера. Цей мікросервіс, побудований на принципах відмовостійкості та низької затримки, виконує ключову функцію координації: паралельно збирає кандидатів від обох джерел, обробляє контекст та передає об'єднаний пул на фінальне ранжування. Архітектурне рішення з використанням асинхронних запитів, кешування та стратегії fallback забезпечило високу доступність системи навіть за умови часткових збоїв її компонентів.

Найважливішим науково-практичним внеском стало дослідження, впровадження та порівняння стратегій об'єднання рекомендацій. Було продемонстровано еволюцію від простої схеми зваженого чергування до складної моделі гібридного ранжування на основі LightGBM. Саме останній підхід, що дозволяє моделі навчатися нелінійним залежностям між ознаками з різних джерел, показав себе найефективнішим. Його успіх доводить, що оптимальне об'єднання є не арифметичною операцією, а задачею машинного навчання, що вимагає навчання на реальних взаємодіях.

Ключовим підтвердженням ефективності всієї розробки стали результати експериментальної оцінки за широким спектром метрик. Гібридна модель достовірно перевершила ізольовані підходи не тільки за класичними метриками точності (Precision@10: +15.8% відносно кращого базового підходу) та ранжування (NDCG@10: +18.0%), але й за показниками, що відображають бізнес-цінність та користувацький досвід. Вона забезпечила найвищу частоту влучень (Hit Rate), найширше охоплення каталогу, сформувала найбільш різноманітні списки та, що найважливіше, досягла найвищого рівня серендипності – здатності приємно дивувати користувача неочевидними, але релевантними пропозиціями.

Таким чином, в рамках розділу було досягнуто головної мети кваліфікаційної роботи: спроектовано та реалізовано працюючу гібридну рекомендаційну систему, яка за рахунок синергії колаборативної фільтрації та аналізу поведінкових даних у реальному часі демонструє вищу ефективність за ключовими критеріями якості в порівнянні з кожним із методів окремо.

ВИСНОВОК

Дана кваліфікаційна робота була присвячена комплексному дослідженню, проєктуванню та практичній реалізації гібридної рекомендаційної системи, що поєднує методи колаборативної фільтрації з аналізом поведінкових даних у реальному часі.

Актуальність роботи обумовлена потребою в подоланні фундаментальних обмежень класичних підходів – проблеми «холодного старту», низької адаптивності до контексту та обмеженої здатності до урізноманітнення рекомендацій – шляхом створення об’єднаної архітектури. Метою роботи було розробити та дослідити гібридну рекомендаційну систему, що поєднує колаборативну фільтрацію з аналізом поведінкових даних у реальному часі, а також експериментально обґрунтувати ефективність такої гібридної системи.

У першому розділі було проведено теоретичний аналіз основних парадигм рекомендаційних систем. Детально досліджено принципи, переваги та недоліки колаборативної та контентної фільтрації. Особлива увага приділена концепціям гібридизації, що дозолило обґрунтувати вибір архітектурної моделі поєднання, спрямованої на взаємне компенсування слабких сторін окремих компонентів. Встановлено, що саме інтеграція довгострокових глобальних паттернів (CF) з короткостроковим контекстом (поведінковий аналіз) є перспективним шляхом для створення адаптивної та персоналізованої системи.

Також у цьому розділі було сформовано методологічну основу для реалізації. У ньому було систематизовано знання про модульну архітектуру сучасних рекомендаційних систем, розглянуто методи збору та обробки поведінкових даних, ключові алгоритмічні підходи для кожного з компонентів.

Особливе значення мав аналіз метрик оцінки ефективності, що дозволило сформувати вичерпний набір критеріїв (від Precision/Recall до Serendipity та Coverage) для подальшого порівняння моделей, зосередившись не лише на точності, а й на бізнес-цінності та якості користувацького досвіду.

Практична частина роботи розпочалася в другому розділі з реалізації колаборативного компоненту. На основі аналізу вимог предметної області було обґрунтовано вибір алгоритму матричної факторизації SVD++. Ключовим результатом розділу став розроблений автоматизований пайплайн навчання та оновлення моделі, що включав ефективні механізми боротьби з розрідженістю даних та стратегію «гарячого» перемикання версій.

Архітектурне рішення з використанням Redis Sorted Sets для зберігання попередньо обчислених рекомендацій та FastAPI для їх обслуговування забезпечило високу продуктивність та масштабованість цього компоненту.

Третій розділ був присвячений створенню динамічного поведінкового сегменту. Було спроектовано та впроваджено пайплайн потокової обробки подій на основі Redis Streams та асинхронного воркера. Розроблено та обґрунтовано метрику нормованої поправленої подібності для алгоритму item-to-item рекомендацій, що ефективно бореться з упередженням популярності. Створено модель динамічного профілю користувача з механізмом часового затухання та агрегацією контекстних ознак, що надало системі здатність миттєво реагувати на поточні інтереси користувача. Інтеграція компоненту через чітко визначені API завершила створення самостійного, швидеого сервісу реального часу.

Фінальною частиною роботи став четвертий розділ, в якому було виконано синтез та оцінку гібридної системи. Було розроблено архітектуру гібридного шару-контролера, що координує роботу компонентів, забезпечує відмовостійкість та формує об’єднаний пул кандидатів. Досліджено еволюцію стратегій об'єднання: від простого зваженого чергування до складної моделі гібридного ранжування на основі LightGBM. Навчання останньої на історичних логах взаємодій дозволило моделі інтелектуально зважувати внесок кожного компонента в залежності від контексту, що є ключем до синергії.

Найважливішим результатом роботи є експериментальне підтвердження вищої ефективності гібридної системи. Комплексна оцінка за широким спектром метрик показала, що гібридна модель достовірно перевершує обидва ізольовані

підходи. Порівняно з кращим базовим підходом (Behavior only), вона забезпечила приріст Precision@10 на 15.8% та NDCG@10 на 18.0%. Крім того, вона продемонструвала вищу частоту влучень (Hit Rate), значно ширше покриття каталогу, більш різноманітні списки та, що найкритичніше, найвищий рівень серендипності (Serendipity@10), що свідчить про здатність приємно дивувати користувача релевантними, але неочевидними пропозиціями. Ці результати остаточно підтверджують основну гіпотезу роботи: об'єднання колаборативної фільтрації та поведінкового аналізу в реальному часі дозволяє створити рекомендаційну систему кращого рівня якості.

Отже, у ході виконання кваліфікаційної роботи було досягнуто поставленої мети. Теоретично обґрунтовано, практично реалізовано та експериментально підтверджено ефективність гібридної рекомендаційної системи. Отримані результати мають наукову цінність як конкретний приклад успішної реалізації гібридної архітектури, а також практичну значущість, пропонуючи готове рішення для підвищення конверсії та задоволеності користувачів на електронних торгових майданчиках. Майбутні дослідження можуть бути спрямовані на інтеграцію глибокого навчання для моделювання послідовностей, автоматизацію гіперпараметричної оптимізації всього конвеєру та впровадження онлайнового навчання гібридного ранкера для ще швидшої адаптації до змін у поведінці користувачів.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Castells P., Jannach D. Recommender systems: A primer [Електронний ресурс] // Advanced Topics for Information Retrieval / ред. R. Baeza-Yates, O. Alonso. – ACM Press, 2023. – Препринт розділу. – Режим доступу: https://arxiv.org/pdf/2302.02579.
2. Rajesh D. B., Kumar A. Collaborative filtering models an experimental and detailed comparative study [Електронний ресурс] // Scientific Reports. – 2025. – Vol. 15, Article number: 31667. – Режим доступу: https://www.nature.com/articles/s41598-025-15096-4.
3. Caballar R. D., Stryker C. What is a recommendation engine? [Електронний ресурс]. – IBM, [б.р.]. – Режим доступу: https://www.ibm.com/think/topics/recommendation-engine.
4. Yaramionak S. E-commerce recommender systems: how they work and why they matter [Електронний ресурс] // EffectiveSoft Blog. – [б.р.]. – Режим доступу: https://www.effectivesoft.com/blog/ecommerce-recommendation-system.html#how-do-e-commerce-recommender-systems-work-.
5. Karlsson J. What it takes to build a real-time recommendation system [Електронний ресурс] // tinybird.co. – 2025. – 24 Apr. – Режим доступу: https://www.tinybird.co/blog/real-time-recommendation-system.
6. Ricci F., Rokach L., Shapira B. Recommender Systems Handbook. – New York : Springer, 2015. – 913 с.
7. Resnick P. [та ін.]. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. // Proceedings of the CSCW '94 Conference. – Chapel Hill, NC : ACM, 1994. – P. 175–186.
8. Manyika J., Chui M., Brown B. Big data: The next frontier for innovation, competition, and productivity : [Електронний ресурс] / James Manyika [та ін.]. – McKinsey Global Institute (MGI), 2011. – Режим доступу: https://www.researchgate.net/publication/312596137_Big_data_The_next_frontier_for_innovation_competition_and_productivity.
9. Content-based vs Collaborative Filtering: Difference [Електронний ресурс] // GeeksforGeeks. – Оновлено 2025. – Режим доступу: https://www.geeksforgeeks.org/machine-learning/content-based-vs-collaborative-filtering-difference/.
10. Lee J. S. Survey of Recommender Systems (Collaborative Filtering) [Електронний ресурс]. – California Polytechnic State University - San Luis Obispo, [б.р.]. – Режим доступу: https://users.csc.calpoly.edu/~dekhtyar/students/Lee-survey.pdf.
11. Collaborative filtering advantages & disadvantages [Електронний ресурс] // Google Developers: Machine Learning – Recommendation. – Оновлено 2025.

– Режим доступу: https://developers.google.com/machine-learning/recommendation/collaborative/summary.
12. Collaborative Filtering [Електронний ресурс] // IBM Think. – [б.р.]. – Режим доступу: https://www.ibm.com/think/topics/collaborative-filtering.
13. What Impact Does Data Sparsity Have on Recommendation Quality? [Електронний ресурс] // Milvus.io – AI Quick Reference. – [б.р.]. – Режим доступу: https://milvus.io/ai-quick-reference/what-impact-does-data-sparsity-have-on-recommendation-quality.
14. Content Based Filtering and Collaborative Filtering: A Comparative Study [Електронний ресурс] / ResearchGate. – Опубліковано 2024. – Режим доступу: https://www.researchgate.net/publication/378841543_Content_Based_Filtering_And_Collaborative_Filtering_A_Comparative_Study.
15. Hu Y., Koren Y., Volinsky C. Collaborative Filtering for Implicit Feedback Datasets [Електронний ресурс]. – AT&T Labs – Research ; Yahoo! Research, [б.р.]. – Режим доступу: http://yifanhu.net/PUB/cf.pdf.
16. Gerard M. Recommendation Systems: Filtering Techniques [Електронний ресурс] // MarioGerard.com. – [б.р.]. – Режим доступу: https://www.mariogerard.com/recommendation-systems-filtering-techniques/.
17. Sabiri B., Khtira A., El Asri B., Rhanoui M. Hybrid quality-based recommender systems: A systematic literature review // J. Imaging. – 2025. – Vol. 11, № 1. – P. 12. – DOI: https://doi.org/10.3390/jimaging11010012.
18. Hybrid Recommender Systems: A Beginner's Guide [Електронний ресурс] // MarketSy AI Blog. – Опубліковано: 15 березня 2024. – Режим доступу: https://marketsy.ai/blog/hybrid-recommender-systems-beginners-guide.
19. Blending Insights: Hybrid Recommender Systems Explained [Електронний ресурс] // ITResearches. – Опубліковано: 17 квітня 2024. – Режим доступу: https://itresearches.com/blending-insights-hybrid-recommender-systems-explained/.
20. Ebeid R., Khalil A. Hybrid Recommender Systems: Challenges, Techniques, and Future Directions [Електронний ресурс] // arXiv preprint. – 2024. – № arXiv:2412.01835. – Режим доступу: https://arxiv.org/pdf/2412.01835.
21. GarvinLi. Basic Concepts and Architecture of a Recommender System [Електронний ресурс] // Alibaba Cloud Community. – 2020. – Режим доступу: https://www.alibabacloud.com/blog/basic-concepts-and-architecture-of-a-recommender-system_596642.
22. Patino, Alexander. Recommendation engines: How they work and why they matter [Електронний ресурс] // Aerospike Blog. – 2025. – Режим доступу: https://aerospike.com/blog/recommendation-engines-how-they-work.
23. Mullen, Melissa. Recommendation Systems: A breakdown of the basic components of recommendation systems [Електронний ресурс] // Medium

(simple ML). – 2024. – Режим доступу: https://medium.com/simple-ml/recommendation-systems-df15f1eaee57.
24. Hardesty, Larry. The history of Amazon's recommendation algorithm [Електронний ресурс] // Amazon Science. – 2019. – Режим доступу: https://www.amazon.science/the-history-of-amazons-recommendation-algorithm.
25. Kumaran, Uttam. How Netflix uses machine learning (ML) to create perfect recommendations [Електронний ресурс] // Brainforge.ai. – [б.р.]. – Режим доступу: https://www.brainforge.ai/blog/how-netflix-uses-machine-learning-ml-to-create-perfect-recommendations.
26. Netflix Help Center. Node 100639 [Електронний ресурс] // Netflix. – [б.р.]. – Режим доступу: https://help.netflix.com/en/node/100639.
27. Hsiao, Ko-Jen; Feng, Yesu; Lamkhede, Sudarshan. Foundation model for personalized recommendation [Електронний ресурс] // Netflix Tech Blog. – 2025. – Режим доступу: https://netflixtechblog.com/foundation-model-for-personalized-recommendation-1a0bd8e02d39.
28. Helic, Denis; Gadiraju, Ujwal; Tkalcic, Marko. User modeling and recommendations [Електронний ресурс] // Frontiers. – 2022. – Режим доступу: https://www.frontiersin.org/research-topics/19653/user-modeling-and-recommendations/magazine.
29. Oard, Douglas W.; Kim, Jinmook. Implicit Feedback for Recommender Systems [Електронний ресурс] // University of Maryland. – [б.р.]. – Режим доступу: https://terpconnect.umd.edu/~oard/pdf/aaai98.pdf.
30. Hu, Yifan; Koren, Yehuda; Volinsky, Chris. Collaborative Filtering for Implicit Feedback Datasets [Електронний ресурс] // Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008). – 2008. – DOI: 10.1109/ICDM.2008.22. – Режим доступу: https://www.researchgate.net/publication/220765111_Collaborative_Filtering_for_Implicit_Feedback_Datasets.
31. Kyrychenko, I., Nesterenko, Y., Chupryna, A., Savulioniene, L., Sakalys, P. Application of clustering methods in recommender systems for user behavior analysis // Environment. Technology. Resources. – 2025. – Vol. 2. – P. 177–180. – DOI: https://doi.org/10.17770/etr2025vol2.8611.
32. Matrix Factorization [Електронний ресурс] // Dive into Deep Learning (d2l.ai). – 2023. – Режим доступу: https://d2l.ai/chapter_recommender-systems/mf.html.
33. Getting Started — Surprise 1 [Електронний ресурс] // Surprise documentation. – [б.р.]. – Режим доступу: https://surprise.readthedocs.io/en/stable/getting_started.html.
34. Rehůřek R. Gensim: models.word2vec – Deep learning with word2vec [Електронний ресурс] / R. Rehůřek // Radim Rehurek: персональний сайт. –

2024. – Режим доступу: https://radimrehurek.com/gensim/models/word2vec.html.
35. Evaluating Recommender Systems: Key Metrics and Approaches [Електронний ресурс] // Evidently AI. – 2025. – Режим доступу: https://www.evidentlyai.com/ranking-metrics/evaluating-recommender-systems.
36. Jadon A., Patil A. A Comprehensive Survey of Evaluation Techniques for Recommendation Systems [Електронний ресурс] / A. Jadon, A. Patil // arXiv.org. – 2024. – 12 January. – Режим доступу: https://arxiv.org/html/2312.16015v2.
37. Precision and Recall at K in Ranking Metrics [Електронний ресурс] // Evidently AI. – 2025. – Режим доступу: https://www.evidentlyai.com/ranking-metrics/precision-recall-at-k.
38. Рекомендаційна система для персоналізованого підбору одягу [Електронний ресурс] : дипломний проєкт / Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського».– К. : НТУУ «КПІ», 2022. – Режим доступу: https://ela.kpi.ua/server/api/core/bitstreams/6f7e529a-c7ba-4d87-9018-1ac059c2499f/content.

ДОДАТКИ

Додаток А

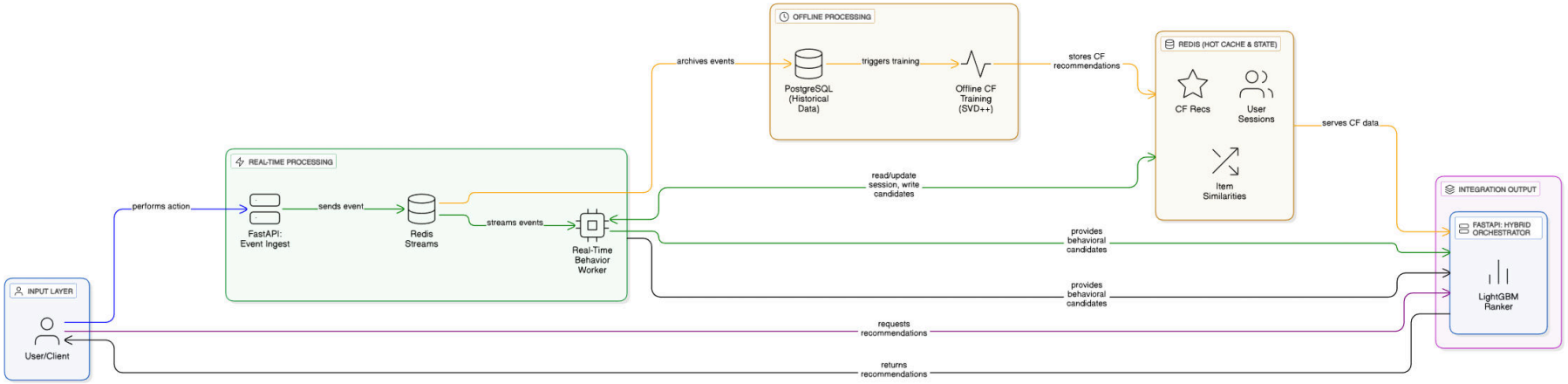

Рис. А.1. Схема архітектури гібридної рекомендаційної системи

Лістинг програмного модуля реалізації гібридного алгоритму (Python)

```
from fastapi import FastAPI, HTTPException, Depends, Query
from fastapi.middleware.cors import CORSMiddleware
from datetime import datetime
from typing import List, Optional

from core.models import RecommendationRequest, RecommendationResponse, HealthCheck
from core.redis_client import redis_client
from core.database import get_db
from sqlalchemy.orm import Session
import sqlalchemy as db

app = FastAPI(title="Recommendation API", version="1.0.0")
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.get("/health")
async def health_check(db_session: Session = Depends(get_db)) -> HealthCheck:
    """Перевірка стану сервісу"""
    redis_ok = False
   postgres_ok = False
```

```
    try:
        redis_client.ping()
        redis_ok = True
    except:
        redis_ok = False

    try:
        db_session.execute(db.text("SELECT 1"))
        postgres_ok = True
    except:
        postgres_ok = False

    status = "healthy" if redis_ok and postgres_ok else "unhealthy"

    return HealthCheck(
        status=status,
        redis=redis_ok,
        postgres=postgres_ok,
        timestamp=datetime.now()
    )

@app.get("/recommend/collaborative/{user_id}")
async def get_cf_recommendations(
    user_id: int,
    limit: int = Query(10, ge=1, le=100)
) -> RecommendationResponse:
```

```
    """Отримати рекомендації від колаборативного фільтра"""
    try:
        # Отримуємо рекомендації з Redis
        cf_key = f"cf:recs:{user_id}"
        recommendations_json = redis_client.get(cf_key)

        if not recommendations_json:
            recommendations = []
        else:
            import json
            recommendations = json.loads(recommendations_json)[:limit]

        return RecommendationResponse(
            user_id=user_id,
            recommendations=recommendations,
            source="cf",
            generated_at=datetime.now()
        )

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@app.get("/recommend/behavior/{user_id}")
async def get_behavior_recommendations(
    user_id: int,
    limit: int = Query(10, ge=1, le=100)
) -> RecommendationResponse:
    """Отримати поведінкові рекомендації в реальному часі"""
```

```
        try:
            # Отримуємо сесію користувача
            session_key = f"user_session:{user_id}"
            session_items = redis_client.zrevrange(session_key, 0, 9)  # Останні 10 товарів

            recommendations = []

            if session_items:
                # Беремо останній переглянутий товар
                last_item = session_items[0]

                # Шукаємо схожі товари
                similar_key = f"similar_to:{last_item}"
                similar_items = redis_client.zrevrange(similar_key, 0, limit-1)

                recommendations = [int(item) for item in similar_items]

            return RecommendationResponse(
                user_id=user_id,
                recommendations=recommendations[:limit],
                source="behavior",
                generated_at=datetime.now()
            )

        except Exception as e:
            raise HTTPException(status_code=500, detail=str(e))
```

```
@app.get("/recommend/hybrid/{user_id}")
async def get_hybrid_recommendations(
    user_id: int,
    limit: int = Query(10, ge=1, le=100)
) -> RecommendationResponse:
    """Гібридні рекомендації (CF + Behavior)"""
    try:
        # Паралельно отримуємо рекомендації з обох джерел
        import json
        import asyncio

        # Отримуємо CF рекомендації
        cf_key = f"cf:recs:{user_id}"
        cf_json = redis_client.get(cf_key)
        cf_recs = json.loads(cf_json) if cf_json else []

        # Отримуємо Behavior рекомендації
        session_key = f"user_session:{user_id}"
        session_items = redis_client.zrevrange(session_key, 0, 4)  # Останні 5 товарів

        behavior_recs = []
        if session_items:
            last_item = session_items[0]
            similar_key = f"similar_to:{last_item}"
            similar_items = redis_client.zrevrange(similar_key, 0, limit*2-1)
            behavior_recs = [int(item) for item in similar_items]
```

```
# Об'єднання: беремо топ з behavior, потім додаємо унікальні з CF
combined = []
seen = set()

# Спочатку додаємо поведінкові (пріоритетні)
for item in behavior_recs:
    if item not in seen and len(combined) < limit:
        combined.append(item)
        seen.add(item)

# Потім додаємо колаборативні
for item in cf_recs:
    if item not in seen and len(combined) < limit:
        combined.append(item)
        seen.add(item)

# Якщо все ще не вистачає, заповнюємо популярними товарами
if len(combined) < limit:
    popular_key = "popular_items"
    popular_items = redis_client.zrevrange(popular_key, 0, limit*2-1)
    for item in popular_items:
        item_int = int(item)
        if item_int not in seen and len(combined) < limit:
            combined.append(item_int)
            seen.add(item_int)
```

```
            return RecommendationResponse(
                user_id=user_id,
                recommendations=combined,
                source="hybrid",
                generated_at=datetime.now()
            )

        except Exception as e:
            raise HTTPException(status_code=500, detail=str(e))

if __name__ == "__main__":
    import uvicorn
    from core.config import settings

    uvicorn.run(
        "main_recommend:app",
        host="0.0.0.0",
        port=settings.RECOMMEND_API_PORT,
        reload=True
    )
```

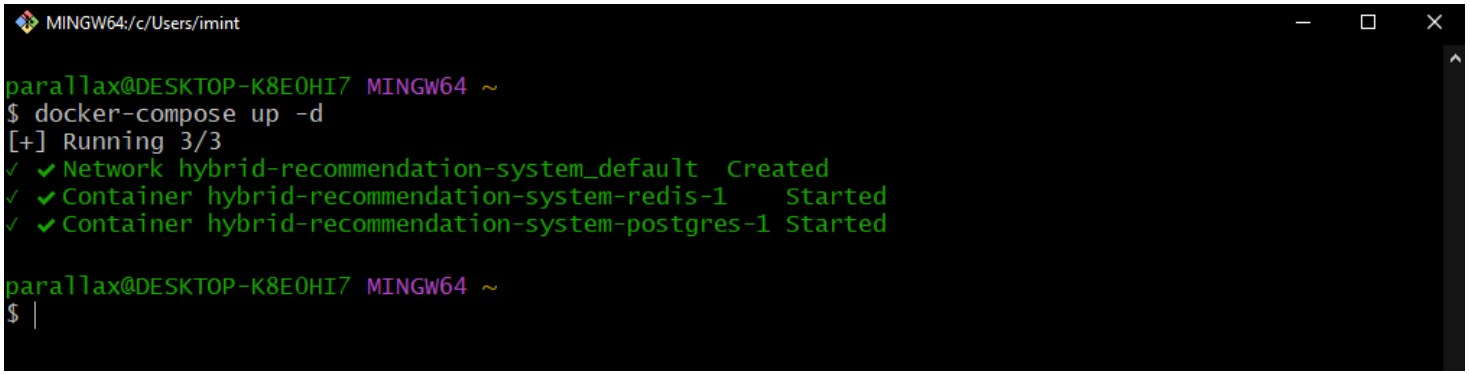

Рис. В.1. Запуск інфраструктури

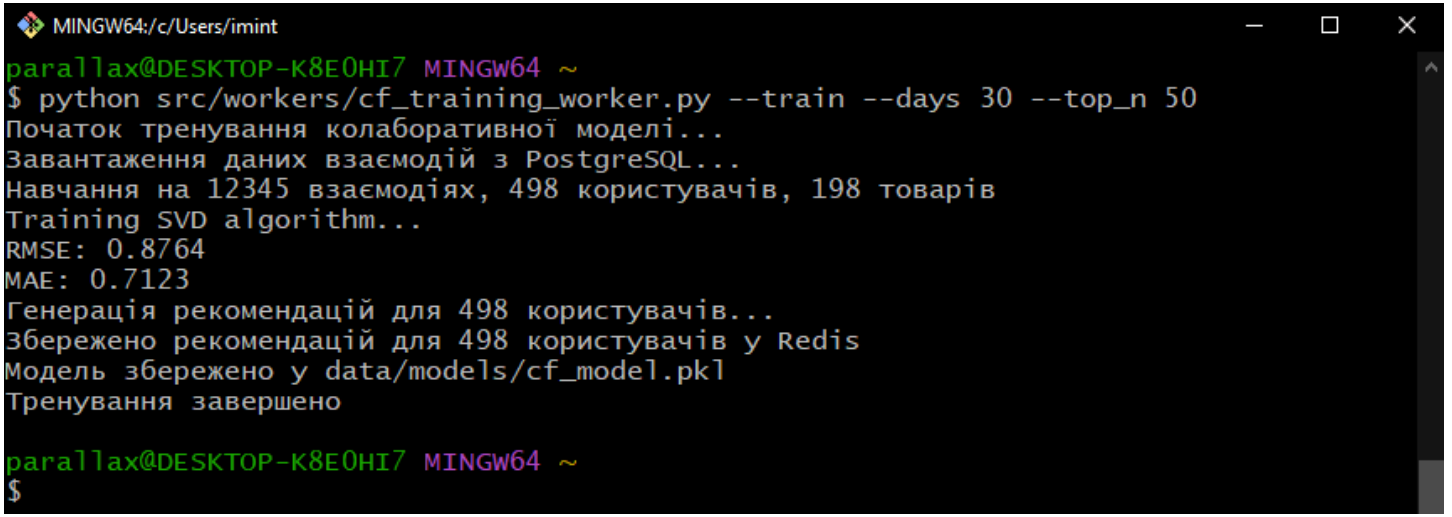

Рис. В.2. Навчання колаборативної моделі (CF)

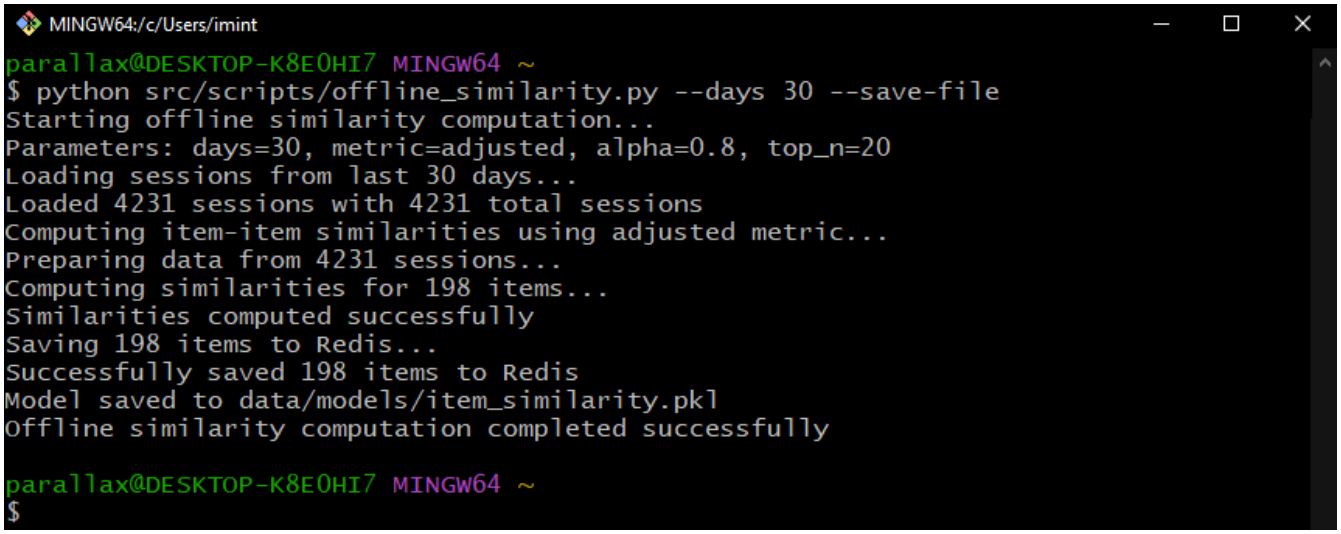

Рис. В.3. Обчислення item-item подібностей

Рис. В.4. Тестування API: Відправка тестової події

Рис. В.5. Тестування API: Отримання CF рекомендацій

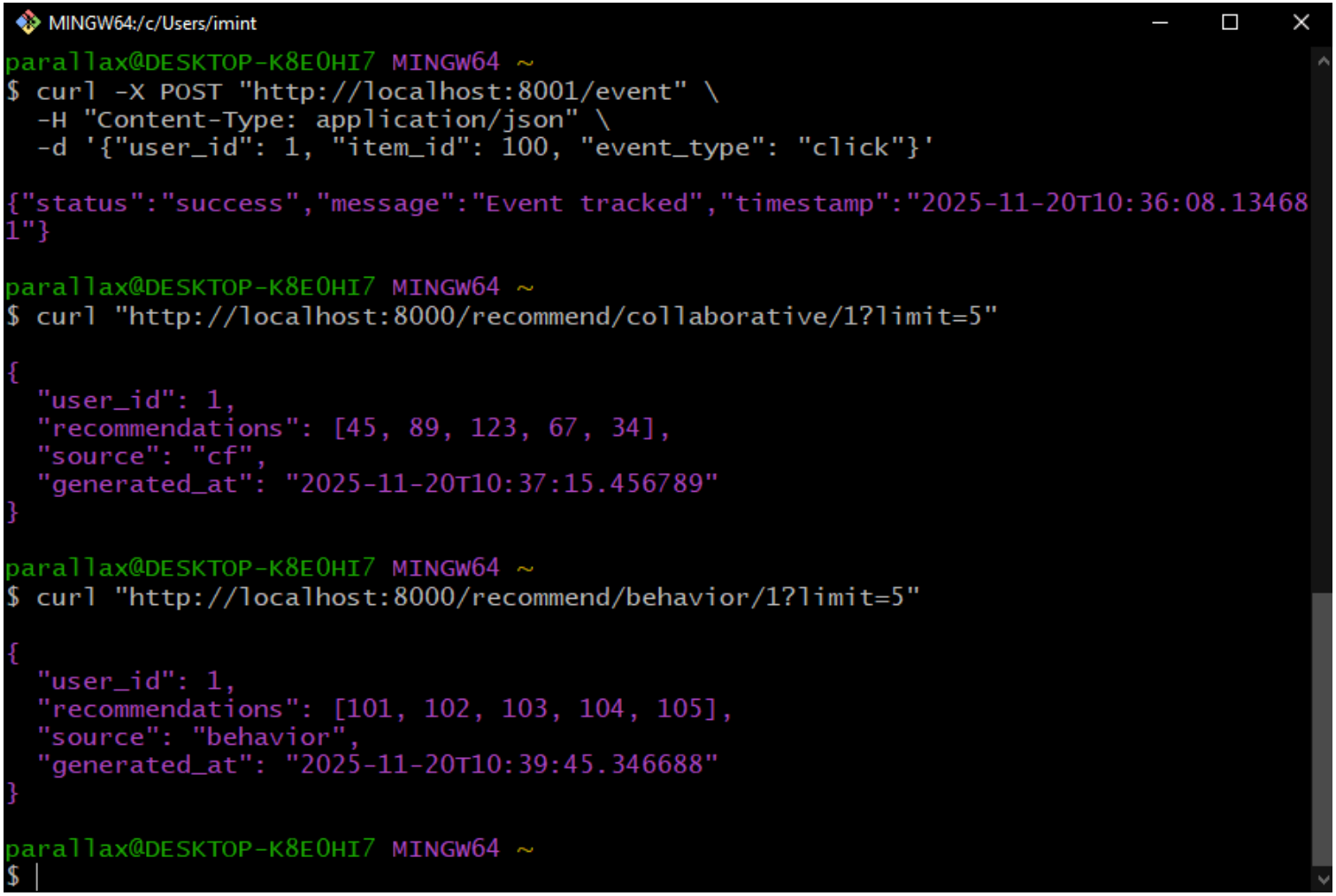

Рис. В.6. Тестування API: Отримання поведінкових рекомендацій

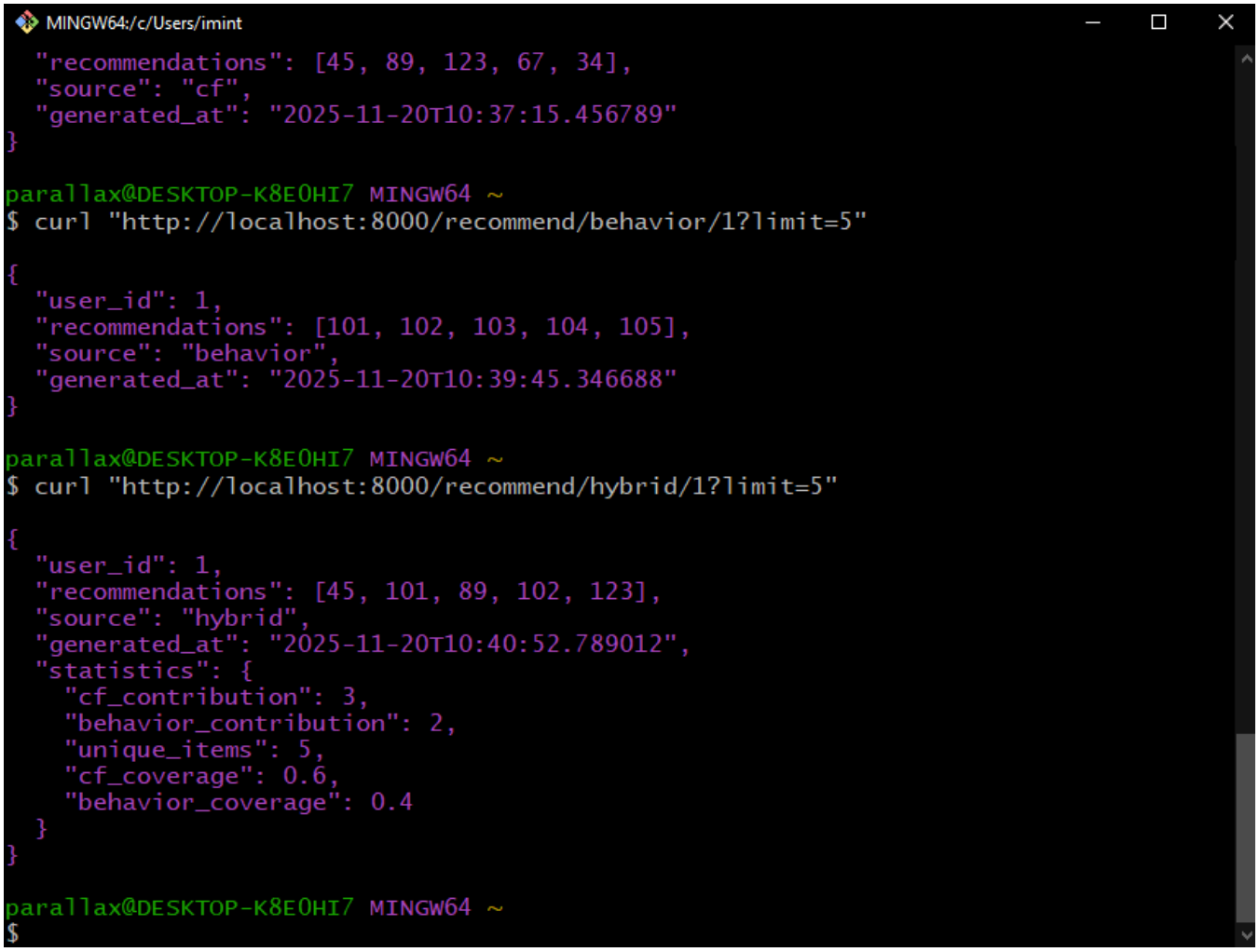

Рис. В.7. Тестування API: Отримання гібридних рекомендацій

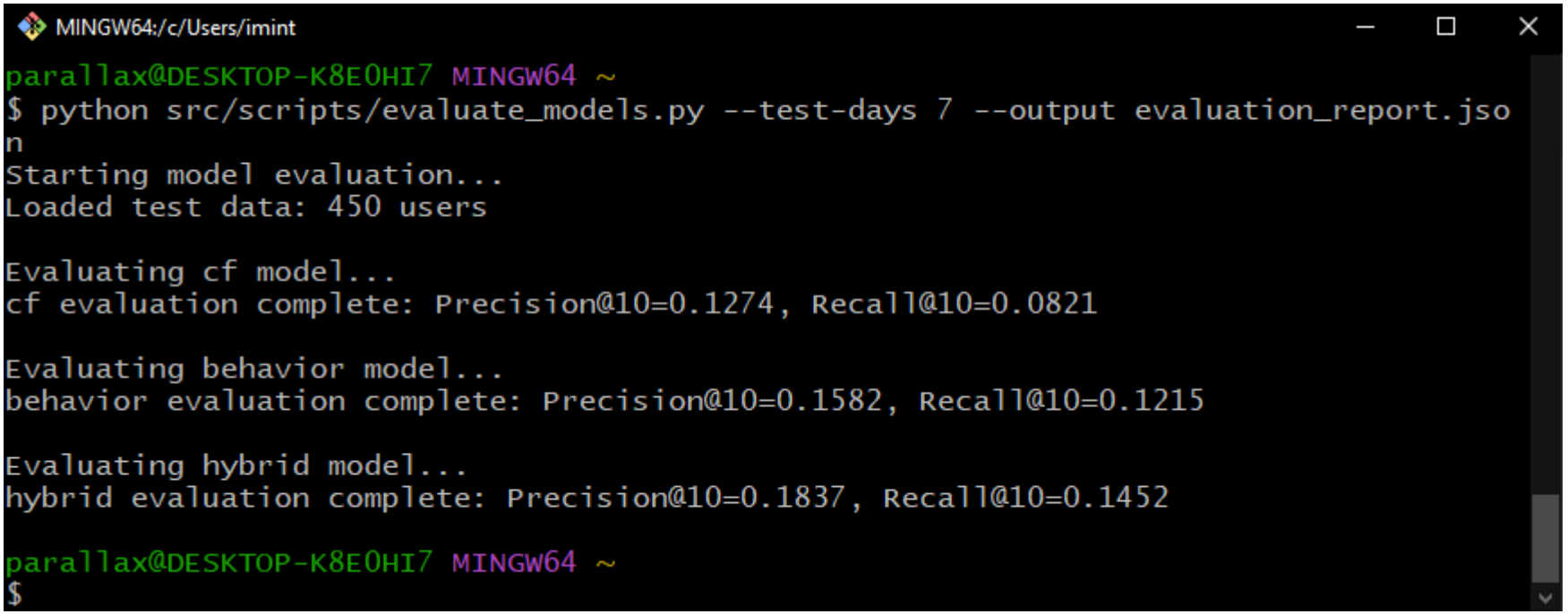

Рис. В.8. Оцінка моделей

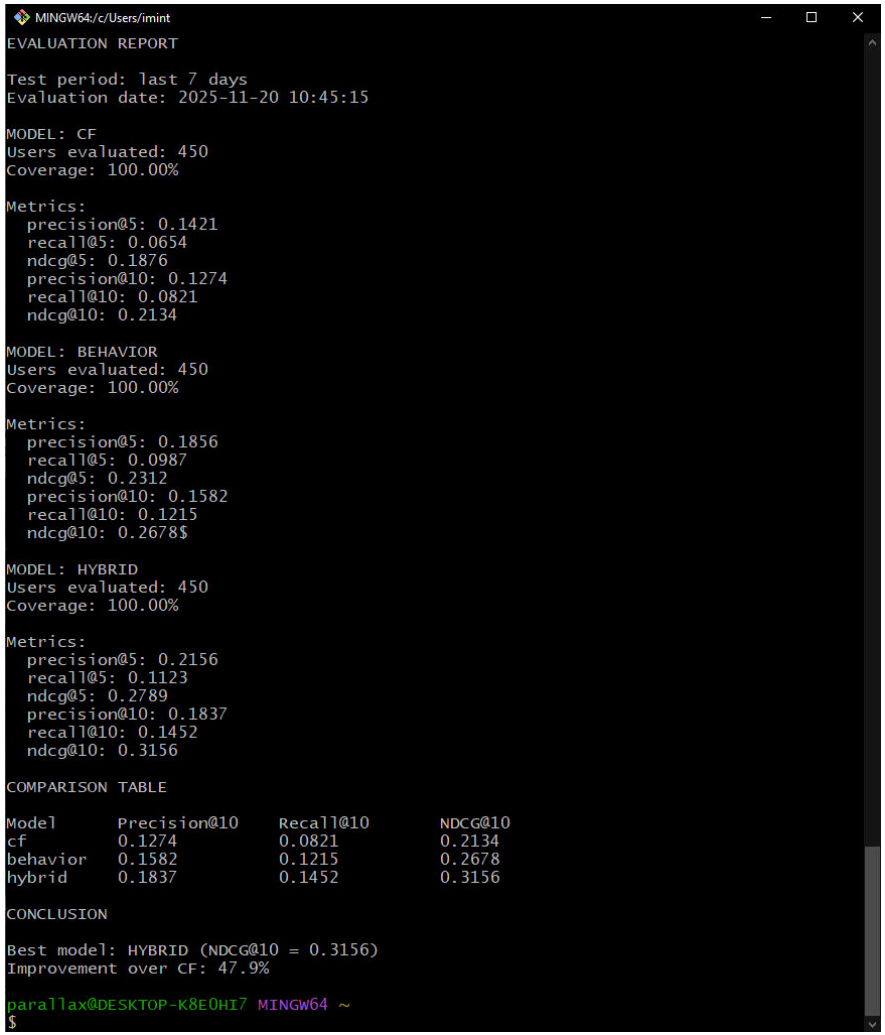

Рис. В.9. Повні результати оцінки

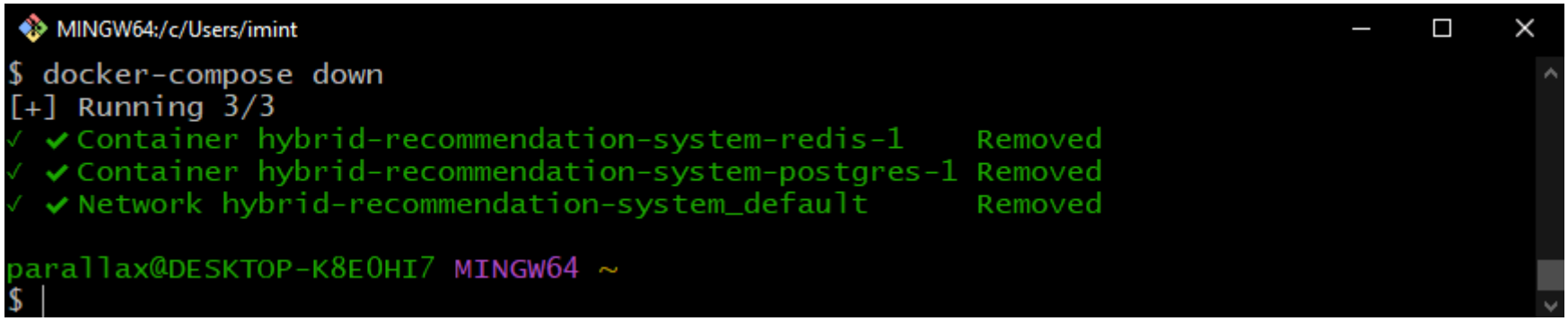

Рис. В.10. Завершення роботи

Додаток Г

Таблиця Г.1

Порівняльні результати оцінки моделей за комплексом метрик

Модель	Precision@10	Recall@10	MAP@10	NDCG@10	MRR	RMSE	HR@10	Coverage@10	Дисперсія	Новизна	Serendipity@10
CF only	0.127	0.082	0.094	0.213	0.241	1.142	0.224	0.531	0.58	0.71	0.42
Behavior only	0.158	0.121	0.130	0.267	0.305	1.298	0.281	0.412	0.49	0.32	0.18
Hybrid	0.183	0.145	0.162	0.315	0.362	1.187	0.331	0.623	0.65	0.65	0.52

Примітка: RMSE розраховано для моделей CF та Hybrid на основі прогнозів рейтингу для відомих пар користувач-товар; для Behavior only ця метрика не застосовується.