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

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2026 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Розробка рекомендаційної системи для магазину комп'ютерних ігор»

Виконав:
студент IV курсу, групи КА-21
Олефіренко Мирослав Олександрович _____

Керівник:
доцент кафедри ММСА, к.ф.-м.н., ст.н.с.
Яковлева Алла Петрівна _____

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

Консультант з нормоконтролю:
Доцент кафедри ММСА, к.ф.-м.н.
Статкевич Віталій Михайлович _____

Рецензент:
доцент кафедри системного проектування НН ІПСА, к.т.н.
Безносик Олександр Юрійович _____

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

Київ – 2026 року

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

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітня програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2026 р.

ЗАВДАННЯ

на дипломну роботу студенту

Олефіренку Мирославу Олександровичу

1. Тема роботи «Розробка рекомендаційної системи для магазину комп'ютерних ігор», керівник роботи Яковлева Алла Петрівна, доцент кафедри ММСА, кандидат фізико-математичних наук, старший науковий співробітник, затверджені наказом по університету від «27» травня 2026 р. №1944-с.
2. Термін подання студентом роботи «10» червня 2026 р.
3. Вихідні дані до роботи: основні теоретичні відомості у галузях машинного навчання та рекомендаційних систем; датасет Kaggle.
4. Зміст роботи: аналіз предметної області та існуючих рішень; методи та алгоритми розв'язання задачі; проєктування та розробка програмної системи; функціонально-вартісний аналіз програмного продукту.
5. Перелік ілюстративного матеріалу: презентація, таблиці та графіки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Функціонально-вартісний аналіз	Рощина Н.В., доц., к.е.н.		

7. Дата видачі завдання 18.02.2026

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Вивчення літератури за темою роботи	21.04.2026	Виконано
2	Аналіз предметної області, підготовка першого розділу	28.04.2026	Виконано
3	Опис методів та алгоритмів, підготовка другого розділу	05.05.2026	Виконано
4	Реалізація системи та проведення експериментів	12.05.2026	Виконано
5	Аналіз результатів, підготовка третього розділу	26.05.2026	Виконано
6	Підготовка економічної частини	31.05.2026	Виконано
7	Оформлення роботи відповідно до нормоконтролю	01.06.2026	Виконано
8	Підготовка презентації доповіді	05.06.2026	Виконано

Студент Мирослав ОЛЕФІРЕНКО

Керівник Алла ЯКОВЛЕВА

РЕФЕРАТ

Дипломна робота: 104 с., 13 рис., 15 табл., 2 додатки, 23 джерела.

РЕКОМЕНДАЦІЙНА СИСТЕМА, КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, СИНГУЛЯРНИЙ РОЗКЛАД МАТРИЦІ, МАШИННЕ НАВЧАННЯ, ГРАДІЄНТНИЙ БУСТИНГ, РАНЖУВАННЯ, FEATURE ENGINEERING, STEAM.

Об’єкт дослідження — процес автоматичного формування персоналізованих рекомендацій відеоігор на основі поведінкових та контентних даних користувачів.

Предмет дослідження — методи колаборативної фільтрації, матричної факторизації та класифікації в задачі побудови рекомендаційних систем для ігрових платформ.

Мета роботи — розробка та дослідження гібридної рекомендаційної системи для магазину комп’ютерних ігор на основі методів колаборативної фільтрації та машинного навчання.

Методи дослідження — колаборативна фільтрація на основі косинусної подібності, алгоритми машинного навчання.

Актуальність — потреба цифрових платформ дистрибуції відеоігор (зокрема Steam) у якісних персоналізованих рекомендаціях в умовах стрімкого зростання каталогів.

Результати роботи — побудовано гібридну рекомендаційну систему та проведено її дослідження. Показано, що розширення набору ознак контентними характеристиками ігор підвищує якість класифікації відгуків.

Шляхи подальшого розвитку предмету дослідження — використання текстів відгуків, застосування нейромережевої колаборативної фільтрації та розгортання вебсервісу для надання рекомендацій.

ABSTRACT

Diploma thesis: 108 pp., 13 figures, 15 tables, 2 appendices, 23 references.
RECOMMENDER SYSTEM, COLLABORATIVE FILTERING, SINGULAR VALUE DECOMPOSITION, MACHINE LEARNING, GRADIENT BOOSTING, RANKING, FEATURE ENGINEERING, STEAM.

The object of research — the process of automatic generation of personalized video game recommendations based on the behavioral and content data of users.

The subject of research — the set of methods of collaborative filtering, matrix factorization, and classification in the task of building recommender systems for gaming platforms.

Objective of the work — to develop and study a hybrid recommender system for a computer game store based on collaborative filtering and machine learning methods.

Research methods — collaborative filtering based on cosine similarity, machine learning algorithms.

Relevance — the need of digital video game distribution platforms (in particular, Steam) for high-quality personalized recommendations under the rapid growth of catalogs.

Results of the work — a hybrid recommender system was built and studied. It was shown that extending the feature set with content characteristics of games improves the review classification quality.

Future development paths — incorporating review texts, applying neural collaborative filtering, and deploying a web service for providing recommendations.

ЗМІСТ

ВСТУП

Стрімкий розвиток цифрової ігрової індустрії призводить до значного збільшення кількості доступних відеоігор на ринку. Платформа Steam, що є одним із найбільших у світі онлайн-магазинів комп'ютерних ігор, налічує понад 50 000 найменувань і щорічно поповнюється тисячами нових продуктів [1]. За таких умов користувач стикається з проблемою інформаційного перевантаження: самостійний пошук релевантного контенту стає дедалі складнішим і часозатратним завданням. Ефективним інструментом вирішення цієї проблеми є рекомендаційні системи, які автоматично аналізують поведінку користувача та формують персоналізований перелік пропозицій [2].

Рекомендаційні системи широко застосовуються в різних предметних областях — від електронної комерції до потокового відео, — однак їх адаптація до специфіки ігрової індустрії залишається актуальним напрямом досліджень. Особливість ігрового контенту полягає у значній варіативності жанрів, цінових категорій та тривалості ігрового процесу, що потребує комплексного підходу до побудови моделі рекомендацій [3]. Додатковим викликом є масштаб даних сучасних платформ: обсяг взаємодій вимірюється десятками мільйонів записів, що висуває жорсткі вимоги до обчислювальної ефективності алгоритмів.

У цій роботі досліджується гібридна рекомендаційна система, що поєднує колаборативну фільтрацію на основі сингулярного розкладу матриці з методами машинного навчання для повторного ранжування кандидатів. Основою для дослідження слугує датасет Steam, що містить понад 41 мільйон відгуків користувачів [4]. Використання даних такого масштабу дозволяє побудувати та оцінити ефективність моделей в умовах, наближених до реального продуктивного середовища.

Актуальність теми зумовлена як практичною потребою ігрових платформ у якісних персоналізованих рекомендаціях, що безпосередньо

впливають на залученість користувачів і доходи, так і науковим інтересом до порівняння класичних і сучасних методів машинного навчання в умовах розрідженості та дисбалансу реальних даних.

Метою роботи є розробка та дослідження гібридної рекомендаційної системи для магазину комп'ютерних ігор на основі методів колаборативної фільтрації та машинного навчання.

Для досягнення поставленої мети необхідно вирішити такі завдання:

— провести огляд існуючих підходів до побудови рекомендаційних систем та обґрунтувати вибір методів;

— виконати попередню обробку та дослідницький аналіз даних датасету Steam;

— реалізувати двоетапний пайплайн рекомендацій: отримання кандидатів засобами матричної факторизації (SVD) та їх повторне ранжування класифікаторами машинного навчання;

— розширити набір ознак контентними характеристиками ігор та дослідити сучасні градієнтні бустинги (XGBoost, LightGBM);

— оцінити якість рекомендаційної системи за стандартними метриками Precision@K, HitRate@K та NDCG@K і порівняти ефективність різних класифікаторів.

Об'єктом дослідження є процес автоматичного формування персоналізованих рекомендацій відеоігор на основі поведінкових та контентних даних користувачів.

Предметом дослідження є методи колаборативної фільтрації, матричної факторизації та класифікації в задачі побудови рекомендаційних систем для ігрових платформ.

Методи дослідження: колаборативна фільтрація на основі косинусної подібності, метод сингулярного розкладу матриці (SVD), логістична регресія, дерево рішень, випадковий ліс, градієнтний бустинг, протокол leave-one-out для оцінювання якості моделей.

Практичне значення отриманих результатів полягає у створенні відтворюваного програмного пайплайну рекомендаційної системи, придатного для інтеграції у платформи цифрової дистрибуції відеоігор, а також у кількісному обґрунтуванні впливу контентних ознак на якість рекомендацій.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Поняття та класифікація рекомендаційних систем

1.1.1 Загальне поняття та призначення

Рекомендаційна система (recommendation system, RecSys) — це клас інформаційних систем, призначених для автоматичного прогнозування вподобань або інтересів користувача на основі накопичених даних про його взаємодію з контентом [2]. Основна задача рекомендаційної системи полягає у ранжуванні множини доступних об’єктів (товарів, ігор, фільмів тощо) відповідно до прогнозованої релевантності для конкретного користувача та поданні найбільш відповідних із них у вигляді персоналізованого списку [3].

Історично перші рекомендаційні системи з’явилися на початку 1990-х років (система Tapestry для фільтрації електронної пошти, проєкт GroupLens для новинних груп). Потужним поштовхом до розвитку галузі став конкурс Netflix Prize (2006–2009), у межах якого методи матричної факторизації продемонстрували суттєву перевагу над класичними підходами та стали стандартом колаборативної фільтрації [8]. Сьогодні рекомендаційні системи є невід’ємною частиною цифрових сервісів: великі платформи електронної комерції, потокового відео та музики, а також магазини застосунків значною мірою покладаються на рекомендації для утримання користувачів. За оцінками дослідників, значна частка взаємодій користувачів з контентом на таких платформах ініціюється саме рекомендаційними алгоритмами [2].

Залежно від типу даних, що використовуються для формування рекомендацій, виділяють три основні підходи: контентну фільтрацію, колаборативну фільтрацію та гібридні методи [2, 5]. Розглянемо кожен із них детальніше.

1.1.2 Контентна фільтрація

Контентна фільтрація (content-based filtering) формує рекомендації на основі характеристик самих об’єктів та профілю вподобань користувача. Для кожного об’єкта будується профіль (вектор атрибутів — жанр, теги, опис, ціна), а для користувача — узагальнений профіль на основі об’єктів, які він оцінив позитивно. Рекомендуються об’єкти, профіль яких найбільше схожий на профіль користувача. Для текстових атрибутів широко застосовують зважування TF-IDF та косинусну подібність векторів.

Перевагою контентної фільтрації є здатність рекомендувати нові об’єкти (для яких відомі атрибути, але ще немає взаємодій) та незалежність від інших користувачів. Основним недоліком є проблема «фільтраційного міхура» — надмірної зосередженості системи на вже відомих користувачеві типах контенту, що обмежує різноманіття та новизну рекомендацій, а також обмеженість якістю наявних атрибутів об’єктів.

1.1.3 Колаборативна фільтрація

Колаборативна фільтрація (collaborative filtering) формує рекомендації на основі поведінки групи користувачів зі схожими вподобаннями, без явного аналізу характеристик об’єктів [6]. Логіка підходу спирається на «мудрість натовпу»: якщо два користувачі мали схожу поведінку в минулому, вони, ймовірно, матимуть схожі вподобання й надалі. Вхідними даними є матриця

взаємодій «користувач × об’єкт», а завданням — прогнозування відсутніх елементів цієї матриці.

Перевагою колаборативної фільтрації є здатність виявляти неочевидні, латентні зв’язки між об’єктами різної природи без потреби в описі їхніх атрибутів. Недоліком є проблема «холодного старту» (cold start) — неможливість формування якісних рекомендацій для нового користувача або об’єкта, про які ще немає достатньо даних [3], а також чутливість до розрідженості матриці взаємодій.

1.1.4 Гібридні підходи

Гібридні підходи (hybrid approaches) поєднують переваги контентної та колаборативної фільтрації з метою подолання їхніх окремих недоліків [5]. Розрізняють кілька стратегій комбінування: зважене поєднання оцінок (score) окремих моделей (weighted), перемикання між методами залежно від ситуації (switching), об’єднання ознак різної природи в одній моделі (feature combination) та каскадне уточнення, за якого один метод відбирає кандидатів, а інший їх переранжовує (cascade).

Гібридні системи, що є предметом дослідження в цій роботі, дозволяють значною мірою пом’якшити проблеми холодного старту й розрідженості: колаборативна складова виявляє латентні вподобання, а контентна — компенсує брак взаємодій за рахунок атрибутів об’єктів. У даній роботі реалізовано каскадну гібридну схему «матрична факторизація + повторне ранжування класифікатором».

1.2 Особливості предметної області: рекомендації відеоігор

1.2.1 Платформа Steam та її екосистема

Steam — провідна онлайн-платформа цифрової дистрибуції відеоігор, що налічує понад 50 000 ігор у каталозі та десятки мільйонів активних користувачів [1]. Платформа надає не лише магазин, а й широку соціальну інфраструктуру: відгуки користувачів, теги, списки бажаного, статистику ігрового часу. Масштаб каталогу робить проблему інформаційного перевантаження особливо гострою, а наявність багатого неявного зворотного зв’язку (відгуки, години гри) створює сприятливі умови для побудови рекомендаційних систем.

Steam надає публічні дані про взаємодії користувачів з іграми, що використовуються в академічних дослідженнях. Саме на основі такого датасету (понад 41 млн відгуків) виконано експериментальну частину даної роботи [4].

1.2.2 Неявний зворотний зв’язок та ігровий час

Рекомендації відеоігор мають низку особливостей порівняно з класичними доменами електронної комерції та медіа. Ключовим неявним сигналом зацікавленості є тривалість ігрового процесу (кількість награних годин), яка має виражений правосторонній розподіл: більшість користувачів грають у конкретну гру відносно недовго, проте існує значна частка «важких»

користувачів із сотнями годин. Тривалість гри корелює із задоволеністю та є інформативною ознакою для прогнозування рекомендації.

Зворотний зв’язок користувачів є переважно неявним і бінарним (рекомендує / не рекомендує), причому з вираженим зміщенням у бік позитивних оцінок (positivity bias) [7], що створює дисбаланс класів і ускладнює навчання класифікаторів. Особливістю неявного зворотного зв’язку є те, що відсутність взаємодії не обов’язково означає негативне ставлення — користувач міг просто не натрапити на гру

1.2.3 Контентні характеристики та бізнес-значення

Ігровий контент характеризується великою кількістю жанрів і тегів, що робить контентні ознаки особливо інформативними для опису об’єктів. Зазначені особливості зумовлюють доцільність гібридного підходу: колаборативна фільтрація використовує масив неявних взаємодій для виявлення латентних вподобань, тоді як контентні характеристики ігор компенсують розрідженість даних та проблему холодного старту.

З погляду бізнес-цінності рекомендаційні системи безпосередньо впливають на ключові показники ігрових платформ: тривалість сесій, конверсію у купівлю та утримання користувачів. Якісні персоналізовані рекомендації знижують вартість пошуку релевантного контенту, підвищують задоволеність користувачів і сприяють просуванню «довгого хвоста» каталогу — менш відомих ігор, які інакше залишилися б непоміченими. Для платформ масштабу Steam навіть незначне підвищення релевантності рекомендацій трансформується у суттєвий приріст доходу, що пояснює значні інвестиції індустрії у розвиток відповідних алгоритмів.

1.3 Огляд методів побудови рекомендаційних систем

1.3.1 Методи на основі пам’яті (memory-based)

Методи на основі пам’яті (memory-based) безпосередньо використовують матрицю взаємодій для обчислення подібності між користувачами або об’єктами [6]. До них належать user-based та item-based підходи, що формують рекомендації на основі найближчих сусідів. Перевагою цих методів є простота та інтерпретованість, недоліком — низька масштабованість через квадратичну обчислювальну складність та чутливість до розрідженості даних.

У підході item-based колаборативної фільтрації, на відміну від user-based, обчислюється подібність не між користувачами, а між об’єктами: рекомендуються ігри, схожі на ті, які користувач уже оцінив позитивно. Item-based підхід часто є стабільнішим за user-based, оскільки множина об’єктів змінюється повільніше за множину користувачів, а матриця подібності об’єктів може бути обчислена офлайн [17]. Для оцінювання подібності застосовують різні міри: косинусну подібність, коефіцієнт кореляції Пірсона, що враховує систематичні зсуви оцінок, та коефіцієнт Жаккара для бінарних взаємодій.

1.3.2 Методи на основі моделі (model-based)

Методи на основі моделі (model-based) попередньо навчають компактну модель, що апроксимує матрицю взаємодій. Найпоширенішим представником є матрична факторизація, зокрема сингулярний розклад (SVD), що відображає

користувачів та об’єкти у спільному прихованому просторі невеликої розмірності [8]. Кожен користувач і об’єкт подаються вектором латентних факторів, а прогноз обчислюється як їхній скалярний добуток. Такі методи стійкіші до розрідженості та значно ефективніші на великих датасетах, що й обумовило їх вибір як основи першого етапу пайплайну в даній роботі.

Для неявного зворотного зв’язку розроблено спеціалізовані варіанти матричної факторизації, що враховують рівень довіри до спостережень (метод Hu, Koren, Volinsky [7]), а також ранжувальні підходи, які безпосередньо оптимізують порядок об’єктів у списку рекомендацій (Bayesian Personalized Ranking [18]).

1.3.3 Типи зворотного зв’язку

Важливою характеристикою даних є тип зворотного зв’язку. Явний зворотний зв’язок (explicit feedback) передбачає прямі числові оцінки користувача (наприклад, рейтинг від 1 до 5), тоді як неявний зворотний зв’язок (implicit feedback) фіксує лише факт взаємодії — перегляд, купівлю, рекомендацію. Дані Steam належать до неявного типу: цільова змінна є бінарною ознакою «рекомендує / не рекомендує». Робота з неявним зворотним зв’язком має свою специфіку: відсутність взаємодії не обов’язково означає негативне ставлення, що враховується відповідними методами зважування [7].

1.3.4 Нейромережеві підходи

Сучасним напрямом є нейромережеві методи, зокрема Neural Collaborative Filtering [9], що замінюють скалярний добуток латентних векторів на нелінійну функцію взаємодії, яка реалізується багатошаровим перцептроном і яка отримується в результаті навчання. До цієї групи також належать автокодувальники для колаборативної фільтрації та рекурентні й трансформерні моделі для послідовних рекомендацій. Попри високу потенційну якість, нейромережеві методи потребують значних обчислювальних ресурсів та великих обсягів даних для навчання, тому в межах даної роботи вони розглядаються як напрям подальших досліджень.

Узагальнене порівняння основних методів колаборативної фільтрації за їхніми перевагами й недоліками наведено в таблиці 1.1.

Таблиця 1.1 — Порівняння методів колаборативної фільтрації

Метод	**Тип**	**Переваги**	**Недоліки**
User-based CF	memory-based	інтерпретованість, простота	низька масштабованість
Item-based CF	memory-based	стабільність, офлайн-обчислення	чутливість до холодного старту
Матрична факторизація (SVD)	model-based	стійкість до розрідженості, масштабованість	менша інтерпретованість
Neural CF	model-based	нелінійні взаємодії, висока якість	потреба у даних і ресурсах

1.4 Проблеми та виклики рекомендаційних систем

1.4.1 Проблема холодного старту

Проблема холодного старту (cold start) є однією з фундаментальних у рекомендаційних системах [3]. Розрізняють холодний старт нового користувача (про якого ще немає історії взаємодій) та нового об’єкта (який ще ніхто не оцінив). Чисто колаборативні методи не здатні рекомендувати у цих умовах. Типовими стратегіями пом’якшення є залучення контентних ознак об’єктів, використання демографічних даних користувача, рекомендація популярних об’єктів за замовчуванням та активне опитування вподобань нового користувача.

1.4.2 Розрідженість та масштабованість

Матриця взаємодій реальних платформ є надзвичайно розрідженою: користувачі взаємодіють лише з мізерною часткою каталогу, тож переважна більшість елементів матриці невідома. Висока розрідженість погіршує якість memory-based методів і вимагає застосування методів зниження розмірності. Окремою проблемою є масштабованість: повне обчислення подібності між усіма парами користувачів має квадратичну складність і є неприйнятним для десятків мільйонів взаємодій, що зумовлює застосування семплування, наближених алгоритмів та матричної факторизації.

1.4.3 Зміщення популярності та різноманіття рекомендацій

Рекомендаційні системи схильні до зміщення популярності (popularity bias): популярні об’єкти отримують більше взаємодій, що змушує систему рекомендувати їх ще частіше, утворюючи зворотний зв’язок. Це поглиблює ефект «фільтраційного міхура» та знижує різноманіття й новизну рекомендацій. Для протидії застосовують поправки на популярність, метрики «поза точністю» (покриття, різноманітність, новизна) та регуляризацію. У даній роботі для пом’якшення зміщення популярності у SVD-оцінку додано відповідну поправку.

1.5 Огляд програмних засобів та аналогів

Провідні ігрові платформи (Steam, Epic Games Store, PlayStation Store) використовують власні пропрієтарні рекомендаційні системи, деталі реалізації яких не розкриваються. Система Steam Interactive Recommender, за наявною інформацією, базується на нейромережевих методах, навчених на історії ігрового часу користувачів. У суміжних доменах класичними прикладами є рекомендації Netflix (матрична факторизація та глибоке навчання) та item-to-item колаборативна фільтрація Amazon. Спільною рисою цих рішень є опора на неявний зворотний зв’язок та поєднання колаборативних і контентних сигналів.

У дослідницькому середовищі поширені спеціалізовані бібліотеки колаборативної фільтрації: Surprise (класичні алгоритми та матрична факторизація), implicit (методи для неявного зворотного зв’язку), LightFM (гібридні моделі). Програмною основою більшості рішень є екосистема мови Python: для обробки даних застосовують pandas і NumPy, для класичних

моделей та матричної факторизації — scikit-learn [15], для градієнтного бустингу — XGBoost [13] та LightGBM [14]. Саме цей набір інструментів обрано для реалізації в даній роботі.

Для узагальненого порівняння наявних підходів у таблиці 1.2 зведено основні класи рекомендаційних рішень за критеріями персоналізації, стійкості до проблеми холодного старту, обчислювальної складності та інтерпретованості.

Таблиця 1.2 — Порівняльний аналіз підходів до побудови рекомендаційних систем

Підхід	**Персоналі-зація**	**Холодний старт**	**Склад-ність**	**Інтерпрето-ваність**
Контентна фільтрація	середня	стійка	низька	висока
User-based CF	висока	вразлива	висока	висока
Матрична факторизація (SVD)	висока	вразлива	середня	середня
Neural CF	дуже висока	вразлива	висока	низька
Гібридний (SVD + ML)	висока	часткова	середня	середня

Як видно з таблиці 1.2, гібридний підхід поєднує високу персоналізацію матричної факторизації з частковою стійкістю до холодного старту за рахунок контентних ознак, зберігаючи помірну обчислювальну складність та інтерпретованість. Саме тому його обрано основою для даної роботи.

1.6 Огляд наукових публікацій за темою

Теоретичну основу дослідження сформовано на базі ключових наукових публікацій у галузі рекомендаційних систем. Узагальнений огляд методів колаборативної фільтрації наведено в роботі Su та Khoshgoftaar [6], а фундаментальні засади побудови систем — у довідниках Ricci, Rokach, Shapira [2] та Aggarwal [3]. Класичний item-based підхід описано в роботі Sarwar та співавторів [17].

Методи матричної факторизації, що стали стандартом після конкурсу Netflix Prize, узагальнено в роботі Koren, Bell, Volinsky [8]. Специфіку неявного зворотного зв’язку розглянуто в роботі Hu, Koren, Volinsky [7], а ранжувальний підхід BPR — у роботі Rendle та співавторів [18]. Сучасні нейромережеві методи представлено роботою He та співавторів про Neural Collaborative Filtering [9].

Питання коректного оцінювання рекомендаційних систем висвітлено в роботах Järvelin та Kekäläinen щодо метрики NDCG [12] та Cremonesi, Koren, Turrin щодо оцінювання задач top-N рекомендацій [20]; саме звідти запозичено протокол оцінювання, застосований у роботі. Методи градієнтного бустингу, використані на етапі повторного ранжування, описано в роботах Chen та Guestrin (XGBoost) [13] і Ke та співавторів (LightGBM) [14]. Практики формування й опису датасетів рекомендаційних систем узагальнено в роботі Harper та Konstan [19].

1.7 Постановка задачі дослідження

У роботі розв’язуються дві взаємопов’язані задачі. Перша — задача бінарної класифікації: за поведінковими та контентними ознаками

передбачити значення цільової змінної is_recommended, тобто чи порекомендує користувач гру. Друга — задача формування рекомендацій: для заданого користувача побудувати впорядкований список топ-K релевантних ігор.

Першу задачу розв’язують класифікатори машинного навчання, які водночас слугують компонентом повторного ранжування у другій задачі. Другу задачу розв’язує двоетапний пайплайн «матрична факторизація + повторне ранжування». Для оцінювання якості класифікації використовують метрики Accuracy, Precision, Recall, F1 та ROC-AUC, причому остання є основною через дисбаланс класів. Якість рекомендацій оцінюють метриками HitRate@K та NDCG@K за протоколом leave-one-out.

Висновки до розділу 1

У першому розділі розглянуто теоретичні засади побудови рекомендаційних систем для ігрових платформ. Описано три основні підходи до формування рекомендацій — контентну фільтрацію, колаборативну фільтрацію та гібридні методи — і обґрунтовано доцільність гібридного підходу. Проаналізовано особливості ігрового домену та платформи Steam, що зумовлюють вибір методів: правосторонній розподіл ігрового часу, неявний зворотний зв’язок, інформативність контентних ознак та дисбаланс класів.

Здійснено огляд методів побудови рекомендаційних систем — memory-based, model-based та нейромережевих — і наведено їх порівняння. Проаналізовано фундаментальні проблеми галузі: холодний старт, розрідженість, масштабованість та зміщення популярності. Виконано огляд існуючих програмних рішень, аналогів і ключових наукових публікацій за темою. Сформульовано дві задачі дослідження — класифікації та формування

рекомендацій — і визначено метрики їх оцінювання, що формують основу для розгляду методів у розділі 2.

РОЗДІЛ 2 МЕТОДИ ТА АЛГОРИТМИ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Формалізація задачі

Нехай задано множину користувачів U та множину ігор I. Взаємодії між ними подаються матрицею R розмірності $|U| \times |I|$, де елемент $r(u,i)$ відображає факт позитивної рекомендації гри i користувачем u. У задачах з неявним зворотним зв'язком матриця R є бінарною та розрідженою — більшість її елементів невідомі [7]. Задача рекомендацій полягає у прогнозуванні відсутніх елементів матриці та формуванні для кожного користувача впорядкованого за спаданням прогнозованої релевантності списку ігор, яких він ще не бачив.

2.2 Колаборативна фільтрація на основі косинусної подібності

Колаборативна фільтрація на основі подібності користувачів (user-based collaborative filtering) є одним із найбільш інтерпретованих методів [6]. Її ключова ідея полягає у пошуку «сусідів» цільового користувача — користувачів зі схожою поведінкою — та формуванні рекомендацій на основі об'єктів, які сусіди оцінили позитивно. Ступінь подібності двох користувачів обчислюється за допомогою косинусної подібності:

$$sim(u,v) = \frac{u \cdot v}{\|u\| \cdot \|v\|}$$

де u та v — вектори взаємодій двох користувачів. Значення косинусної подібності лежить у діапазоні від 0 до 1: значення, близьке до 1, вказує на високу схожість користувачів. Прогнозований інтерес цільового користувача

до об’єкта, якого він ще не бачив, обчислюється як зважена сума оцінок сусідів:

$$\hat{r}(u,i) = \frac{\sum_{v \in N(u)} sim(u,v) r(v,i)}{\sum_{v \in N(u)} |sim(u,v)|}$$

де $N(u)$ — множина K найближчих сусідів цільового користувача. Основним практичним обмеженням методу є його обчислювальна складність: повне обчислення подібності між усіма парами користувачів вимагає $O(|U|^2 \cdot |I|)$операцій, що є неприйнятним для великих датасетів. У даній роботі ця проблема вирішується шляхом семплювання підмножини користувачів [4].

2.3 Матрична факторизація методом усіченого SVD

Матрична факторизація є одним із найефективніших підходів до задачі колаборативної фільтрації. Її ідея полягає в апроксимації розрідженої матриці взаємодій R добутком двох матриць значно меншого розміру, що відображають приховані (латентні) фактори користувачів та об’єктів відповідно [7]. Усічений сингулярний розклад (Truncated SVD) для матриці R:

$$R \approx U_k \Sigma_k V_k^T$$

де U_k — матриця прихованих факторів користувачів, V_k — матриця прихованих факторів об’єктів, Σ_k— діагональна матриця сингулярних чисел, k — кількість латентних вимірів. Матриці U_k та V_k інтерпретуються як представлення (embeddings) користувачів і об’єктів у спільному прихованому просторі розмірності k [8]. Прогнозований інтерес користувача до об’єкта обчислюється як скалярний добуток відповідних прихованих векторів :

$$\hat{r}(u,i) = p_u^T q_i$$

Вибір кількості латентних вимірів k є важливим гіперпараметром: занадто мале значення призводить до недонавчання, надто велике — до

перенавчання та надмірних обчислювальних витрат. На відміну від фіксованого апріорного вибору, у даній роботі значення k обґрунтовано емпірично шляхом перебору з оцінюванням цільової метрики (підрозділ 3.6). Для врахування популярності об'єктів до базової SVD-оцінки додається зважена поправка на популярність:

$$score(u, i) = p_u^T q_i + \alpha \cdot pop(i)$$

де $pop(i)$— нормована популярність об'єкта, α — ваговий коефіцієнт, що регулює її вплив. Обчислення усіченого SVD виконується рандомізованим алгоритмом Халка–Мартінссона–Троппа [21], що є особливо ефективним для розріджених матриць [8].

Слід зазначити, що класичний SVD визначено лише для повністю заданих матриць, тоді як матриця взаємодій є розрідженою. Тому на практиці застосовують варіанти матричної факторизації, що навчаються лише на відомих елементах із регуляризацією для запобігання перенавчанню. Найпоширенішими є метод почергових найменших квадратів (Alternating Least Squares, ALS) та стохастичний градієнтний спуск (SGD), що мінімізують регуляризований квадратичний функціонал помилки на спостережуваних взаємодіях [8]. Зазначені процедури належать до класичних методів безумовної оптимізації — мінімізації функцій багатьох змінних [22, 23].

У даній роботі використано усічений SVD на зваженій матриці неявного зворотного зв'язку як обчислювально ефективний та стійкий до розрідженості варіант, що не потребує підбору швидкості навчання та добре масштабується на мільйонах взаємодій.

2.4 Класифікатори машинного навчання для повторного ранжування

Другим етапом двоетапного пайплайну є повторне ранжування (reranking) кандидатів, отриманих від SVD, за допомогою класифікатора машинного навчання. Задача класифікатора — прогнозувати ймовірність того, що користувач порекомендує конкретну гру, на основі сформованих ознак [10]. У даній роботі досліджуються п’ять класифікаторів: три базові та два сучасні градієнтні бустинги.

Логістична регресія є базовою лінійною моделлю для задач бінарної класифікації. Ймовірність належності об’єкта до позитивного класу:

$$P(y = 1 \mid x) = \frac{1}{1+e^{-(w^T x+b)}}$$

де x — вектор ознак, w — вектор вагових коефіцієнтів, b — зсув. Логістична регресія є ефективним базовим рішенням (baseline), що добре масштабується на великих датасетах [10].

Дерево рішень (Decision Tree) — нелінійна модель, що рекурсивно розбиває простір ознак на прямокутні регіони за допомогою порогових умов. На кожному вузлі вибирається ознака та поріг, що максимально зменшують нечистоту вузлів-нащадків. Для задачі класифікації нечистота вузла вимірюється критерієм Джині:

$$Gini = 1 - \sum_c p_c^2$$

де p_c — частка об’єктів класу c у вузлі. Основним гіперпараметром моделі є максимальна глибина дерева, що обмежує перенавчання.

Випадковий ліс (Random Forest) — ансамблева модель, що будує множину дерев рішень на різних бутстреп-вибірках з навчального набору та з випадковою підмножиною ознак на кожному вузлі. Фінальний прогноз отримується усередненням прогнозів окремих дерев:

$$\hat{y} = \frac{1}{T}\sum_{t=1}^{T} h_t\,(x)$$

де T— кількість дерев, h_t— прогноз t −го дерева. Рандомізація при побудові дерев знижує кореляцію між ними та зменшує дисперсію ансамблю [11].

Градієнтний бустинг (XGBoost [13], LightGBM [14]) будує ансамбль дерев послідовно, де кожне наступне дерево виправляє помилки попередніх, мінімізуючи функцію втрат методом градієнтного спуску у функціональному просторі:

$$F_m(x) = F_{m-1}(x) + \eta \cdot h_m(x)$$

де η — темп навчання (learning rate), h_m— m −те дерево. Градієнтні бустинги є сучасним стандартом для табличних даних і додані до дослідження за рекомендацією щодо розширення набору моделей. Для нормалізації числових ознак перед навчанням лінійних моделей застосовується стандартизація (StandardScaler):

$$z = \frac{x-\mu}{\sigma} \tag{2.1}$$

де μ та σ — середнє значення та стандартне відхилення ознаки на навчальній вибірці. Для компенсації дисбалансу класів усі моделі використовують механізм балансування ваг класів.

Проблема дисбалансу класів є типовою для задач з неявним зворотним зв’язком і потребує спеціальних методів. Існує три основні групи підходів.

1. Методи на рівні даних — передискретизація (oversampling) міноритарного класу, зокрема алгоритм SMOTE, що синтезує нові приклади, або субдискретизація (undersampling) мажоритарного класу.
2. Методи на рівні алгоритму — зважування класів (class weighting), за якого помилки на міноритарному класі штрафуються сильніше, та параметр scale_pos_weight у градієнтних бустингах.
3. Корекція порогу класифікації (decision threshold tuning) на основі аналізу ROC- або precision-recall кривої.

У даній роботі застосовано зважування класів як обчислювально ефективний підхід, що не збільшує обсяг навчальних даних і безпосередньо інтегрований у використовувані бібліотеки.

2.5 Поведінкові та контентні ознаки (feature engineering)

Етап feature engineering формує вхідні ознаки класифікатора, що поділяються на два блоки.

Поведінкові ознаки формуються зі статистики відгуків: helpful (кількість позначок «корисно»), funny (кількість «смішно»), hours (кількість награних годин) та похідні від них — логарифм годин hours_log = ln(1 + hours), що згладжує правосторонній розподіл; частка корисних позначок helpful_ratio; сумарна активність engagement_score; бінарна ознака is_high_playtime, що вказує на перевищення медіанного часу гри.

Контентні ознаки, додані для підвищення якості рекомендацій згідно з рекомендацією рецензента, формуються з характеристик ігор: порядковий рейтинг гри, частка позитивних відгуків positive_ratio, логарифм кількості відгуків, ціна, знижка, ознака безкоштовності, підтримувані платформи (Windows, macOS, Linux), а також 30 найчастіших тегів (жанрів) у вигляді бінарних ознак. Один і той самий механізм формування ознак використовується і для навчання класифікаторів, і для повторного ранжування, що усуває розбіжність ознак між етапами навчання та застосування моделі.

2.6 Метрики оцінювання якості рекомендаційних систем

Оцінювання рекомендаційних систем поділяють на офлайн- та онлайн-методи. Онлайн-оцінювання (A/B-тестування на реальних користувачах) є найбільш достовірним, проте дорогим і недоступним у межах дослідницької роботи. Офлайн-оцінювання на історичних даних потребує коректної стратегії розбиття: випадкове розбиття взаємодій може призвести до «витоку»

інформації з майбутнього у минуле, тому в рекомендаційних системах застосовують або хронологічне розбиття (за часом), або протокол leave-one-out, що приховує по одній взаємодії на користувача. Останній обрано в даній роботі як стандарт сучасних досліджень RecSys.

Для оцінювання якості рекомендаційних систем у даній роботі використовується стандартний протокол leave-one-out з семплюванням (Sampled LOO), запропонований у роботі He et al. [9]. Відповідно до цього протоколу, для кожного тестового користувача одна позитивна взаємодія вилучається з навчальної вибірки та використовується як цільова; система ранжує цільовий об'єкт серед 99 випадково відібраних негативних об'єктів. Семплування негативів суттєво знижує обчислювальну складність порівняно з ранжуванням усього каталогу і є усталеною практикою для великих датасетів.

Для оцінювання якості класифікації будують матрицю помилок (Confusion matrix) та обчислюють частку істинно-позитивних (TPR) і хибно-позитивних (FPR) прогнозів:

$$TPR = \frac{TP}{TP+FN}, FPR = \frac{FP}{FP+TN} \tag{2.2}$$

де TP, FN, FP, TN — кількість істинно-позитивних, хибно-негативних, хибно-позитивних та істинно-негативних прогнозів відповідно. Площа під ROC-кривою (ROC-AUC) інтерпретується як імовірність того, що модель присвоїть випадковому позитивному об'єкту вищу оцінку, ніж випадковому негативному. HitRate@K — частка користувачів, для яких цільовий об'єкт потрапив до топ-K:

$$HitRate@K = \frac{1}{|U_{test}|} \sum_u [ran\, k_u \leq K]$$

Precision@K — частка релевантних об'єктів серед K рекомендованих:

$$Precision@K = \frac{HitRate@K}{K}$$

NDCG@K враховує позицію цільового об'єкта у списку, надаючи більшу вагу об'єктам, розташованим вище:

$$NDCG@K = \frac{1}{|U_{test}|}\sum_{u}\frac{\ln 2}{\ln(rank_u+1)}$$

NDCG@K є більш інформативною метрикою порівняно з HitRate@K, оскільки враховує не лише факт попадання цільового об’єкта в топ-K, а й конкретну позицію у списку [12].

2.7 Додаткові метрики та аспекти оцінювання

Окрім основних метрик, для всебічного оцінювання рекомендаційних систем застосовують низку додаткових показників. Recall@K у протоколі з єдиним прихованим позитивним об’єктом збігається з HitRate@K, оскільки повнота визначається попаданням єдиного релевантного об’єкта до топ-K. Середній обернений ранг (Mean Reciprocal Rank, MRR) оцінює, наскільки високо в списку розташований перший релевантний об’єкт:

$$MRR = \frac{1}{|U_{test}|}\sum_{u}\frac{1}{rank_u}$$

Поряд із метриками точності дедалі більшого значення набувають метрики «поза точністю» (beyond-accuracy):

1) покриття каталогу (catalog coverage) — частка об’єктів, що взагалі рекомендуються;
2) різноманітність (diversity) — несхожість об’єктів усередині списку рекомендацій;
3) новизна (novelty) — здатність системи рекомендувати непопулярні, але релевантні об’єкти.

Ці метрики важливі для уникнення ефекту «фільтраційного міхура» та надмірної концентрації рекомендацій на популярних іграх. У даній роботі основну увагу приділено метрикам точності ранжування як визначальним для

якості персоналізації, тоді як аналіз метрик різноманітності визначено як напрям подальших досліджень.

Висновки до розділу 2

У другому розділі формалізовано задачу та описано математичний апарат методів для її розв’язання:

1) метод колаборативної фільтрації на основі косинусної подібності;
2) матрична факторизація усіченим SVD;
3) п’ять класифікаторів машинного навчання: випадковий ліс (Random Forest), дерева рішень (Decision Tree), логістична регресія, градієнтні бустинги (XGBoost та LightGBM)

Описано два блоки ознак — поведінкові та контентні. Наведено метрики оцінювання якості класифікації та ранжування. Отримані теоретичні положення формують основу для практичної реалізації, описаної в розділі 3.

РОЗДІЛ 3 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

3.1 Вибір технологій та інструментів розробки

Програмну реалізацію виконано мовою Python 3.13 — фактичним стандартом у галузі аналізу даних і машинного навчання. Для обробки та зберігання даних використано бібліотеки pandas і NumPy, а також колонковий формат Parquet для кешування проміжних артефактів, що суттєво прискорює повторні запуски. Класичні моделі та матрична факторизація реалізовані засобами бібліотеки scikit-learn [15], градієнтні бустинги — бібліотеками XGBoost [13] і LightGBM [14], обробка даних — бібліотекою pandas [16]. Для роботи з розрідженими матрицями застосовано модуль scipy.sparse, для візуалізації — Matplotlib і Seaborn. Розробку та документування проведено в середовищі Jupyter Notebook, що забезпечує наочний наскрізний ланцюг «задача → рішення → результат».

3.2 Опис набору даних Steam

Емпіричною основою дослідження слугує відкритий набір даних «Game Recommendations on Steam» [4], що містить понад 41 мільйон очищених і попередньо оброблених записів про рекомендації користувачів, а також детальну інформацію про ігри. Датасет складається з чотирьох файлів, характеристики яких наведено в таблиці 3.1. Цільовою змінною є бінарний атрибут is_recommended — факт позитивної рекомендації гри користувачем.

Датасет не містить особистої інформації: усі ідентифікатори анонімізовано на етапі попередньої обробки [4].

Таблиця 3.1 — Характеристики файлів датасету Steam

Файл	**Записів**	**Розмір**	**Призначення**
recommendations.csv	41 154 794	≈1,9 ГБ	Відгуки (взаємодії)
users.csv	14 306 064	≈184 МБ	Профілі користувачів
games.csv	50 872	≈4,6 МБ	Атрибути ігор
games_metadata.json	50 872	≈17 МБ	Описи й теги ігор

3.3 Дослідницький аналіз даних

3.3.1 Аналіз цільової змінної

Розподіл цільової змінної is_recommended є істотно нерівномірним: близько 85,8% відгуків є позитивними, що відповідає загальновідомій тенденції Steam до переважання позитивних оцінок. Це явище відоме як positivity bias і є характеристикою більшості платформ з неявним зворотним зв’язком [7]. Практичним наслідком дисбалансу є те, що тривіальний класифікатор, який завжди прогнозує мажоритарний клас, досягає точності 0,858; отже, метрика Accuracy є малоінформативною, і основною метрикою якості обрано ROC-AUC, стійку до дисбалансу. Для компенсації дисбалансу під час навчання застосовано зважування класів та стратифіковане розбиття даних.

3.3.2 Аналіз ознаки «кількість годин»

Кількість награних годин (hours) є ключовою поведінковою ознакою. Її розподіл є сильно правостороннім: більшість користувачів грають менше десяти годин у конкретну гру, проте існує значна кількість викидів із сотнями та тисячами годин. Як видно з рисунку 3.1, медіанна кількість годин для позитивних рекомендацій помітно вища, ніж для негативних, що підтверджує гіпотезу про зв'язок між тривалістю гри та задоволеністю користувача та обґрунтовує інформативність цієї ознаки. Для нейтралізації впливу викидів застосовано логарифмічне перетворення $hourlog = ln(1 + hours)$.

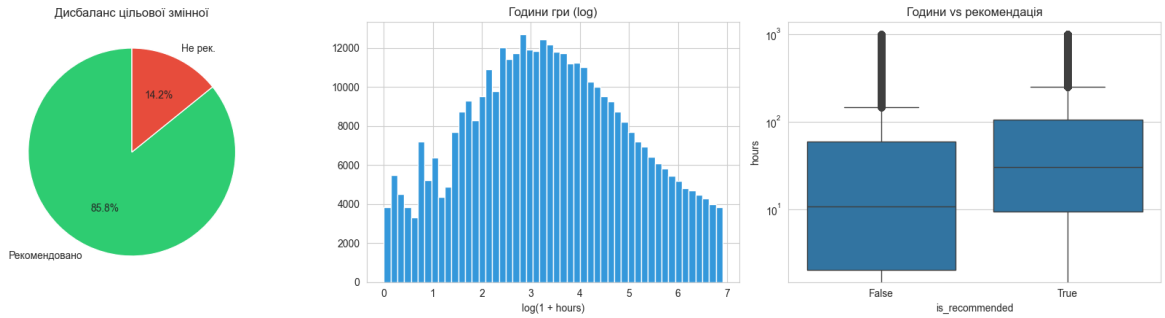


Рисунок 3.1 — Дисбаланс цільової змінної та розподіл кількості годин гри

3.3.3 Кореляційний аналіз ознак

Для виявлення взаємозв'язків між ознаками та цільовою змінною побудовано кореляційну матрицю (рисунок 3.2). Аналіз показує, що найсильніше з цільовою змінною is_recommended корелюють контентні ознаки positive_ratio та rating_ord, а також поведінкові ознаки тривалості гри, що підтверджує доцільність їх включення до моделі. Водночас ознаки hours та hours_log очікувано сильно корельовані між собою, як і helpful з engagement_score, що враховано при інтерпретації важливості ознак.

Помірний рівень мультиколінеарності є прийнятним для деревних моделей та градієнтного бустингу, стійких до корельованих ознак.

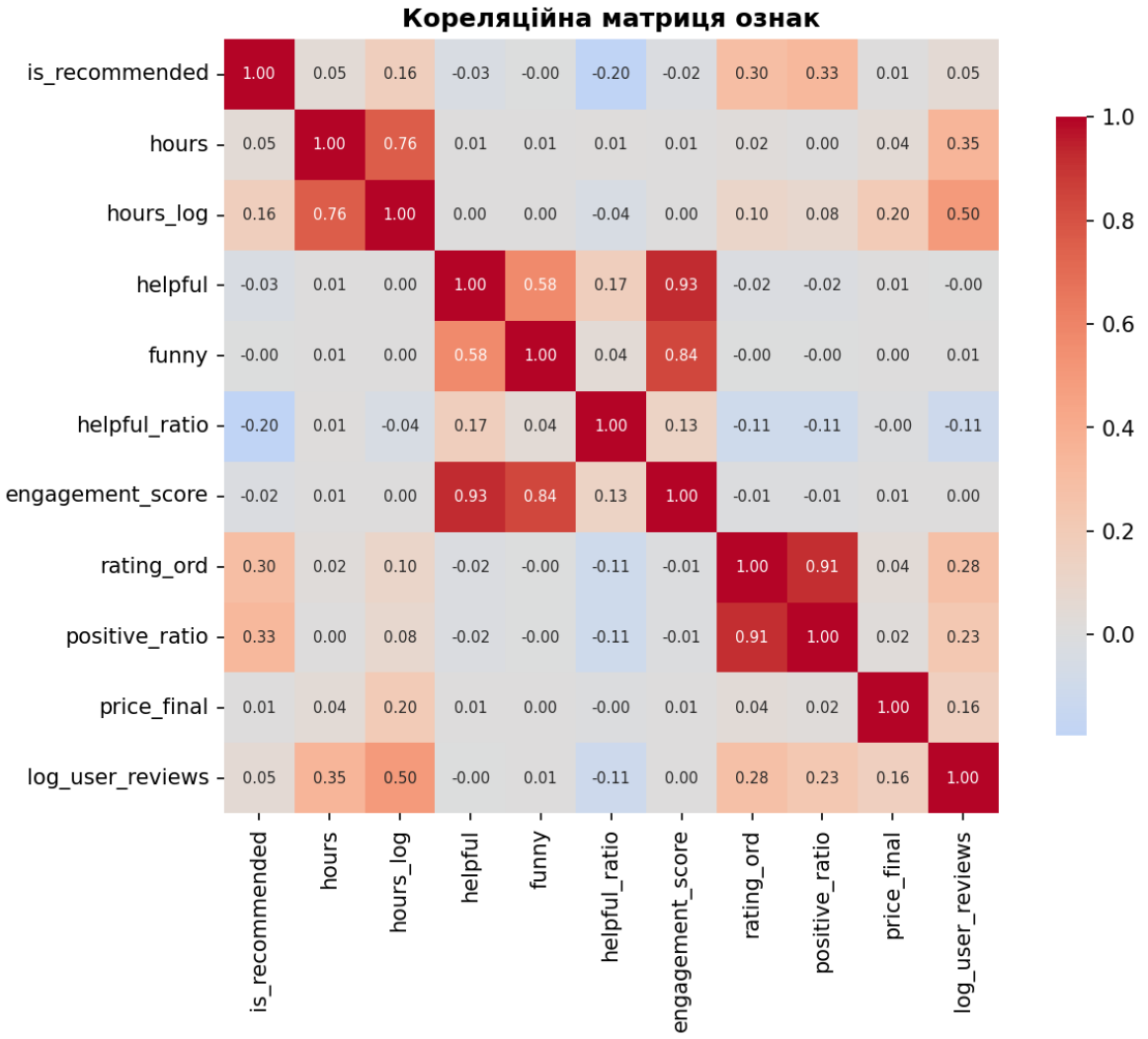


Рисунок 3.2 — Кореляційна матриця ознак

3.3.4 Аналіз популярності ігор

Розподіл кількості відгуків на гру є вкрай нерівномірним і має характер «довгого хвоста», що типово для каталогів цифрового контенту. Як видно з рисунку 3.3, невелика кількість надпопулярних ігор збирає десятки тисяч відгуків, тоді як переважна більшість ігор має лише десятки або сотні відгуків. Розподіл у логарифмічних координатах близький до лінійного, що свідчить

про степеневий характер розподілу популярності. Ця особливість обґрунтовує застосування поправки на популярність у SVD-оцінці (див. підрозділ 2.3) та фільтрації непопулярних ігор (щонайменше 200 відгуків) при побудові матриці взаємодій.

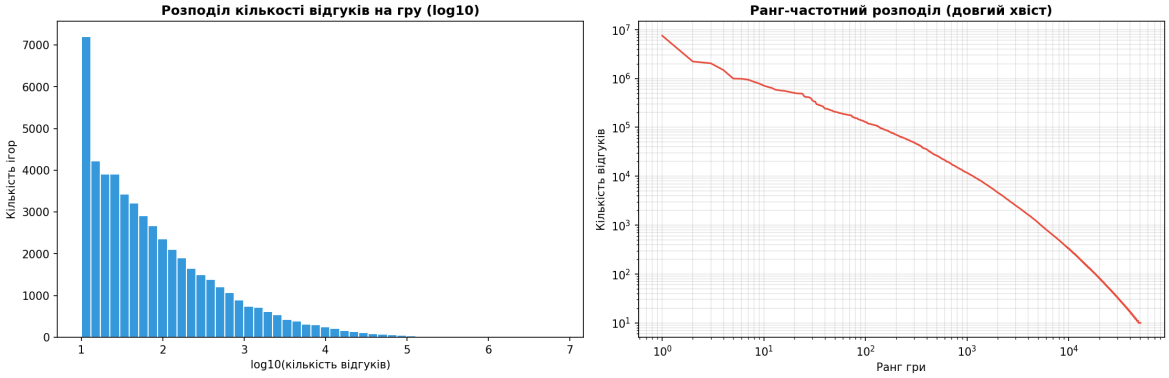


Рисунок 3.3 — Розподіл популярності ігор (довгий хвіст)

3.3.5 Аналіз жанрового та рейтингового розподілу

Аналіз контентних характеристик ігор (рисунок 3.4) показує, що найпоширенішими тегами-жанрами є Indie, Singleplayer, Action та Adventure, що відображає структуру каталогу Steam. Розподіл агрегованих рейтингів ігор зміщений у бік позитивних оцінок (Positive, Very Positive та Mostly Positive переважають), що узгоджується з виявленим раніше positivity bias на рівні окремих відгуків. Багатий жанровий простір обґрунтовує доцільність використання тегів як контентних ознак: 30 найчастіших тегів додано до набору ознак у вигляді бінарних індикаторів.

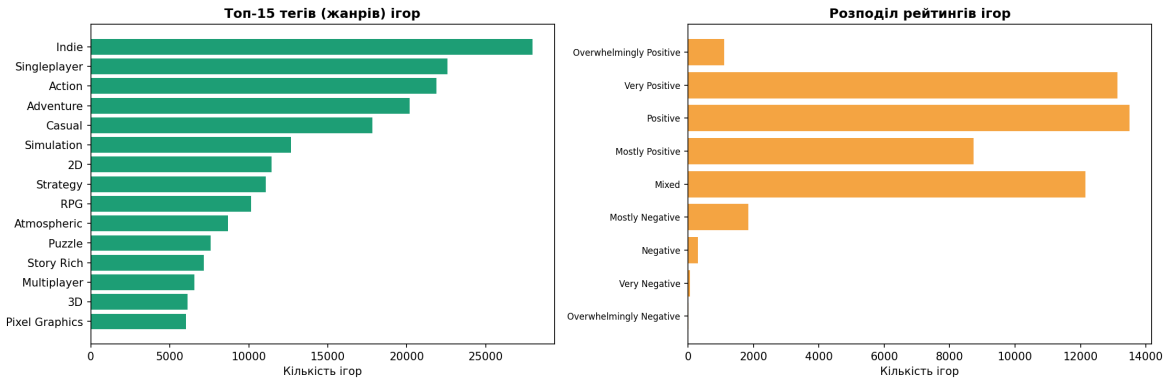


Рисунок 3.4 — Розподіл тегів-жанрів та рейтингів ігор

3.3.6 Аналіз розрідженості матриці взаємодій

Матриця взаємодій «користувач × гра», що є основою колаборативної фільтрації, характеризується надзвичайно високою розрідженістю. Для відібраної вибірки з 40 000 активних користувачів та 8 928 популярних ігор густота заповнення матриці становить лише близько 0,4% (розрідженість 99,6%), як показано на рисунку 3.5. Розподіл кількості взаємодій на користувача залишається правостороннім навіть після фільтрації активних користувачів, а більшість ігор отримує відносно невелику кількість взаємодій. Висока розрідженість є фундаментальним викликом для рекомендаційних систем: вона унеможливлює пряме порівняння користувачів за спільними іграми та обґрунтовує застосування методів зниження розмірності (матричної факторизації), здатних узагальнювати приховані закономірності навіть за обмеженої кількості спостережень.

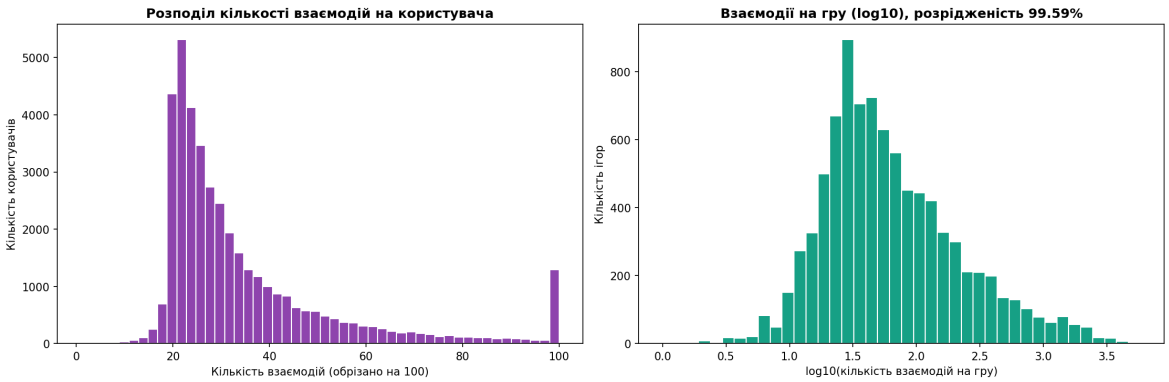


Рисунок 3.5 — Розподіл активності користувачів та розрідженість матриці взаємодій

3.4 Підготовка даних та feature engineering

3.4.1 Обробка пропущених значень

Аналіз якості даних показав відсутність пропущених значень для ключових атрибутів (is_recommended, hours, helpful, funny). Нульові значення атрибутів helpful та funny є семантично коректними і означають відсутність відповідних реакцій, а не відсутність даних; тому для обробки застосовано заповнення нулями.

3.4.2 Відтворюване семплування

Через значний обсяг даних застосовано відтворюване семплування з фіксованим зерном генератора (random_state = 42). Для задачі класифікації сформовано рівномірний випадковий зріз обсягом близько 400 тисяч відгуків.

Для рекомендаційної системи відібрано всі взаємодії 40 000 випадково обраних активних користувачів (щонайменше 20 відгуків) з популярними іграми (щонайменше 200 відгуків), що забезпечує достатню щільність матриці взаємодій для матричної факторизації. Усі проміжні артефакти кешуються у форматі Parquet, тож первинний файл обробляється лише один раз. Такий підхід усуває розбіжність між описом і реалізацією та забезпечує повну відтворюваність експериментів.

3.4.3 Формування набору ознак

Сформований набір ознак поєднує поведінкові та контентні характеристики, описані в підрозділі 2.5. Поведінкові ознаки обчислюються зі статистики відгуків, контентні — з атрибутів ігор (рейтинг, ціна, теги, платформи). Зведений перелік основних груп ознак наведено в таблиці 3.2.

Таблиця 3.2 — Набір ознак для навчання класифікаторів

Група ознак	**Приклади**	**Кількість**
Поведінкові	helpful, funny, hours, hours_log, helpful_ratio, engagement_score, is_high_playtime	7
Контентні (числові)	rating_ord, positive_ratio, log_user_reviews, price_final, discount	5
Контентні (бінарні)	is_free, win, mac, linux, теги-жанри	34
Усього	—	46

3.4.4 Розбиття та стандартизація даних

Дані розбиваються на навчальну та тестову вибірки у співвідношенні 80/20 зі стратифікацією за цільовою змінною, що гарантує збереження розподілу класів в обох вибірках. Перед навчанням лінійних моделей усі числові ознаки стандартизуються за формулою (2.1) за допомогою StandardScaler, що є обов'язковим для коректної роботи логістичної регресії та покращує збіжність решти моделей.

3.5 Архітектура двоетапного пайплайну рекомендацій

Запропонована система рекомендацій реалізується у вигляді двоетапного пайплайну. На першому етапі (candidate retrieval) для кожного цільового користувача генерується множина кандидатів-ігор засобами матричної факторизації: будується зважена розріджена матриця «користувач × гра», де вага дорівнює $log(1 + hours) + 1$, після чого виконується її усічений SVD-розклад. Близькість користувача й гри обчислюється як скалярний добуток латентних векторів із поправкою на популярність (див. підрозділ 2.3). На другому етапі (reranking) кандидати додатково оцінюються навченим класифікатором, а підсумкова гібридна оцінка формується як зважена сума нормованої SVD-оцінки та ймовірності від класифікатора.

Архітектура «отримання кандидатів + повторне ранжування» (retrieval-then-ranking) є галузевим стандартом промислових рекомендаційних систем. Її перевага полягає у поділі задачі на два етапи з різними вимогами: на першому етапі швидкий і масштабований метод (матрична факторизація) відбирає з тисяч ігор десятки найперспективніших кандидатів, а на другому — обчислювально дорожчий, але точніший класифікатор переранжовує лише цю

невелику множину. Такий поділ дозволяє поєднати масштабованість із якістю, чого неможливо досягти застосуванням складної моделі до всього каталогу одразу.

3.6 Реалізація матричної факторизації та вибір кількості факторів

3.6.1 Побудова матриці взаємодій

На основі відібраних взаємодій будується розріджена матриця «користувач × гра». Через значну розрідженість (густота заповнення менше 0,5%) для її зберігання та обробки використовується формат scipy.sparse.csr_matrix, що суттєво зменшує витрати пам’яті. Розмір матриці після фільтрації та leave-one-out розбиття становить 39 978 користувачів × 8 924 ігор.

3.6.2 Обґрунтування кількості латентних факторів

Кількість латентних факторів k обґрунтовано емпірично шляхом перебору значень з оцінюванням якості ранжування (NDCG@10) та поясненої дисперсії. Результати наведено в таблиці 3.3 та на рисунку 3.6. Якість виходить на плато в діапазоні 16–128 факторів і знижується при 256, тоді як пояснена дисперсія монотонно зростає. Це наочно демонструє, що надлишкові фактори моделюють шум, а пояснена дисперсія не є цільовою метрикою якості ранжування. За результатами перебору обрано k = 64 як оптимальний компроміс між якістю та обчислювальними витратами.

Таблиця 3.3 — Залежність якості ранжування від кількості латентних факторів

k (факторів)	Пояснена дисперсія	NDCG@10
16	10,7%	0,467
32	16,3%	0,453
64	24,6%	0,477
128	36,0%	0,444
256	50,0%	0,386

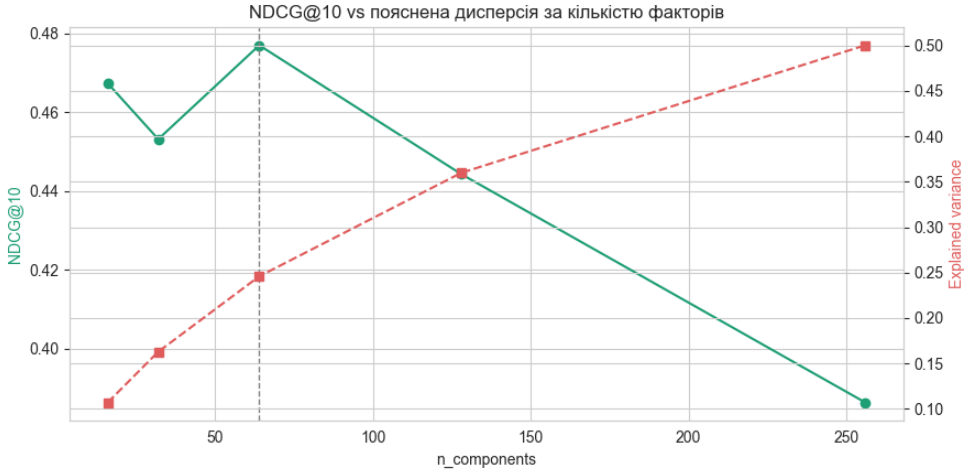


Рисунок 3.6 — NDCG@10 та пояснена дисперсія залежно від k

3.7 Навчання класифікаторів та повторне ранжування

П’ять класифікаторів навчаються на навчальній вибірці з повним набором із 46 ознак (таблиця 3.2). Для компенсації дисбалансу класів застосовано балансування ваг класів. Базові моделі (логістична регресія, дерево рішень, випадковий ліс) реалізовано засобами scikit-learn, градієнтні бустинги — бібліотеками XGBoost та LightGBM. Гіперпараметри моделей додатково оптимізовано методом перебору по сітці (Grid Search) з трикратною крос-валідацією за метрикою ROC-AUC; установлено, що значення за замовчуванням близькі до оптимальних. На етапі повторного ранжування

кандидати від SVD оцінюються найкращим класифікатором, а фінальна гібридна оцінка обчислюється як зважена сума нормованої SVD-оцінки та ймовірності класифікатора.

3.8 Порівняння класифікаторів та аналіз результатів

3.8.1 Зведене порівняння

Результати оцінки п’яти класифікаторів на тестовій вибірці за метриками Accuracy, Precision, Recall, F1 та ROC-AUC наведено в таблиці 3.4 та на рисунку 3.7. Найкращу якість за основною метрикою ROC-AUC демонструють градієнтні бустинги: XGBoost досягає ROC-AUC 0,832.

Таблиця 3.4 — Порівняння класифікаторів за метриками бінарної класифікації

Модель	**Accuracy**	**Precision**	**Recall**	**F1**	**ROC-AUC**
XGBoost	0,759	0,948	0,760	0,844	0,832
LightGBM	0,757	0,948	0,758	0,842	0,832
Random Forest	0,772	0,944	0,781	0,855	0,826
Decision Tree	0,757	0,943	0,763	0,844	0,817
Logistic Regression	0,756	0,941	0,763	0,843	0,811

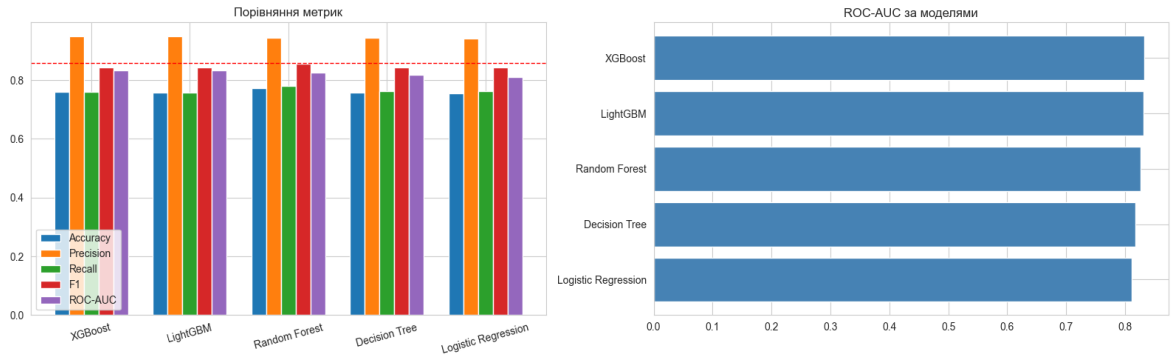


Рисунок 3.7 — Порівняння метрик якості класифікаторів

3.8.2 Вплив контентних ознак

Для перевірки впливу контентних ознак проведено абляційний експеримент. Модель XGBoost, навчена лише на поведінкових ознаках, досягає ROC-AUC 0,714, що збігається з результатом попередньої версії системи. Додавання контентних ознак підвищує ROC-AUC до 0,832 (приріст +0,118). Це кількісно доводить, що розширення набору ознак контентними характеристиками ігор дає вимірюваний приріст якості класифікації та підтверджує доцільність відповідної рекомендації.

3.8.3 Аналіз найкращої моделі: Confusion matrix

Аналіз матриці помилок (Confusion matrix) найкращої моделі (рисунок 3.8) показує, що більшість помилок становлять хибно-позитивні прогнози. У контексті рекомендаційної системи такі помилки є менш критичними, ніж хибно-негативні: система може запропонувати гру, яку користувач зрештою не оцінить, проте не відфільтрує гру, що могла б йому сподобатися.

3.8.4 ROC-аналіз

Для комплексної оцінки якості класифікатора використано ROC-криву та площу під нею (ROC-AUC, див. підрозділ 2.6). Значення ROC-AUC найкращої моделі (0,832) суттєво перевищує значення 0,5 випадкового класифікатора та значення 0,71 попередньої версії, що використовувала лише числові ознаки. Це підтверджує, що модель систематично впорядковує об’єкти краще за випадкове вгадування в усьому діапазоні порогів, що є критично важливим у задачі ранжування кандидатів.

3.9 Аналіз важливості ознак

Важливість ознак найкращої моделі (рисунок 3.9) показує, що найбільший внесок у прогноз дають частка позитивних відгуків positive_ratio, рейтинг гри та поведінкові ознаки залученості. Це узгоджується з результатами дослідницького аналізу даних і підтверджує визначальну роль як контентних характеристик якості гри, так і поведінкових сигналів залученості користувача.

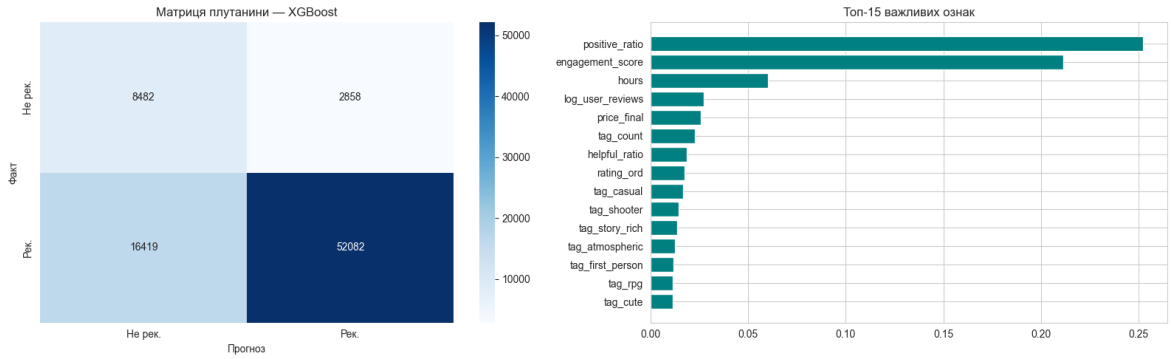


Рисунок 3.7 — Confusion matrix та важливість ознак найкращої моделі

3.10 Оцінювання рекомендаційної системи за протоколом Sampled LOO

Оцінювання повного пайплайну проведено за протоколом Sampled LOO (підрозділ 2.6) на 2000 користувачів. Порівняно дві конфігурації: чистий SVD та SVD із повторним ML-ранжуванням найкращим класифікатором. Результати наведено в таблиці 3.5 та на рисунку 3.9. Метод SVD забезпечує HitRate@10 ≈ 0,69 та NDCG@10 ≈ 0,46.

Таблиця 3.5 — Якість рекомендацій за протоколом Sampled LOO

Конфігурація	**K**	**HitRate@K**	**NDCG@K**
SVD	5	0,571	0,418
SVD	10	0,690	0,456
SVD	20	0,792	0,478
SVD + XGBoost	5	0,537	0,399
SVD + XGBoost	10	0,631	0,434
SVD + XGBoost	20	0,715	0,466

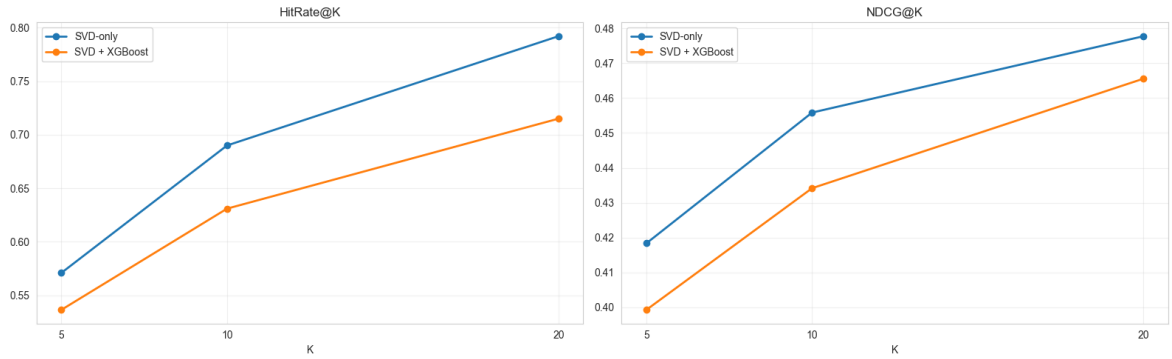


Рисунок 3.9 — Метрики рекомендацій для конфігурацій SVD та SVD + ML

Важливим результатом є те, що повторне ML-ранжування не покращує якості рекомендацій, а на більших значеннях K навіть погіршує її. Додатковий перебір ваги класифікатора у гібридній оцінці показав, що NDCG максимізується за нульової ваги ML, тобто при чистому SVD. Причина полягає в тому, що класифікатор оцінює узагальнену «якість гри», а не персоналізовану релевантність для конкретного користувача. Таким чином,

оптимальною конфігурацією є чистий SVD, а персоналізоване реранжування потребує ознак рівня пари «користувач–гра» (напрям подальших досліджень).

Для ілюстрації роботи системи у таблиці 3.6 наведено топ-10 персоналізованих рекомендацій, згенерованих для реального користувача з тестової вибірки на основі його історії взаємодій. Усі рекомендовані ігри є новими для користувача (відсутні в його історії), що підтверджує коректність фільтрації вже переглянутих ігор.

Таблиця 3.6 — Приклад топ-10 рекомендацій для користувача

№	**Гра**	**Позит. відгуків**
1	Company of Heroes 2	72%
2	Deus Ex: Mankind Divided	76%
3	Control Ultimate Edition	86%
4	ASTRONEER	86%
5	Tabletop Simulator	92%
6	Dead Space (2008)	90%
7	Layers of Fear (2016)	86%
8	New World	59%
9	The Binding of Isaac	90%
10	STAR WARS Jedi: Fallen Order	85%

3.11 Аналіз обчислювальної складності та часу виконання

Окремої уваги потребує обчислювальна ефективність пайплайну при роботі з датасетом обсягом понад 41 млн записів. Завантаження та підрахунок статистик виконуються потоковою обробкою файлу частинами (chunking) за

один прохід зі складністю O(N), що дозволяє обробляти дані, які не вміщуються в оперативну пам’ять цілком. Обчислення усіченого SVD має складність, близьку до лінійної відносно кількості ненульових елементів розрідженої матриці, завдяки рандомізованому алгоритму. Оцінювання за протоколом Sampled LOO виконується для фіксованої вибірки користувачів і не залежить від повного каталогу ігор. Заміри часу виконання основних етапів на типовому персональному комп’ютері наведено в таблиці 3.7.

Таблиця 3.7 — Обчислювальна складність та час виконання етапів пайплайну

Етап обробки	**Складність**	**Час виконання**
Сканування датасету (один прохід)	O(N)	≈ 42 с
Побудова вибірок та артефактів	O(N)	≈ 90 с
Навчання п’яти класифікаторів	O(n·d·T)	≈ 2–3 хв
Усічений SVD-розклад	O(nnz·k)	≈ 3 с
Оцінювання Sampled LOO (2000 юзерів)	O(m·K)	≈ 1 хв

Наведені результати свідчать, що завдяки потоковій обробці, кешуванню проміжних артефактів у форматі Parquet та використанню розріджених структур даних система забезпечує прийнятний час виконання навіть на масштабних даних без застосування спеціалізованої обчислювальної інфраструктури.

3.12 Збереження результатів та артефактів

За результатами експериментів навчені моделі та допоміжні об’єкти серіалізуються для подальшого розгортання. Збережені артефакти включають навчені класифікатори, об’єкт StandardScaler, зведену таблицю метрик model_comparison.csv та результати оцінювання рекомендаційної системи. Кешовані у форматі Parquet вибірки забезпечують швидке відтворення експериментів без повторної обробки первинного датасету.

Висновки до розділу 3

У третьому розділі описано програмну реалізацію та експериментальне дослідження гібридної рекомендаційної системи. Обґрунтовано вибір технологій, проведено дослідницький аналіз даних та відтворюване семплування. Показано, що розширення набору ознак контентними характеристиками та застосування градієнтного бустингу підвищує ROC-AUC класифікації з 0,714 до 0,832. Емпірично обґрунтовано вибір кількості латентних факторів SVD (k = 64). Установлено, що чистий SVD забезпечує найкращу якість ранжування (HitRate@10 ≈ 0,69), тоді як наївне ML-реранжування не покращує рекомендацій, оскільки використовувані ознаки не є персоналізованими — обґрунтований результат, що визначає напрям подальшого вдосконалення системи.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться техніко-економічне обґрунтування розробки гібридної рекомендаційної системи для магазину комп'ютерних ігор.

Сучасні рекомендаційні системи потребують не лише високої точності персоналізованих прогнозів, а й оптимізації обчислювальних ресурсів, витрачених на їх розробку, обробку розріджених матриць великих розмірностей та подальшу експлуатацію. Тому програмний продукт у цій роботі розглядається як складний об'єкт, що потребує комплексного аналізу: від технічної реалізації алгоритмів матричної факторизації та градієнтного бустингу до оцінки економічної доцільності обраного стеку технологій.

Для вибору оптимального варіанту проектування застосовується метод функціонально-вартісного аналізу (ФВА). Цей метод дозволяє детально дослідити кожну функцію майбутньої системи, оцінити її важливість та зіставити з витратами на її впровадження. Аналіз проводиться поетапно: спочатку визначаються основні та допоміжні функції продукту, формуються варіанти їх виконання, після чого на основі технічних параметрів та розрахованої собівартості обирається остаточний варіант реалізації програмної системи.

4.1 Постановка задачі проектування

Метою даного етапу є оцінка та вибір оптимальної стратегії розробки програмного продукту, що дозволяє автоматизувати процес формування персоналізованих рекомендацій ігор на основі поведінкових та контентних даних платформи Steam. Продукт повинен забезпечувати інструментарій для

колаборативної фільтрації, зниження розмірності та машинного реранжування.

Вкажемо основні функціональні вимоги до програмного продукту.

1. Алгоритмічна база — реалізація методів матричної факторизації (усічений SVD) та ансамблевих моделей машинного навчання (XGBoost, LightGBM).
2. Оптимізація пам'яті — робота з сильно розрідженими матрицями (розрідженість понад 99%) та великими обсягами даних (понад 41 млн відгуків).
3. Аналітична частина — розрахунок метрик якості ранжування (NDCG@K, HitRate@K) та метрик класифікації (ROC-AUC).
4. Ергономічність — забезпечення швидкодії пайплайну та кешування проміжних результатів.

4.2 Обґрунтування функцій програмного продукту

На основі аналізу вимог до гібридної рекомендаційної системи, було виділено основні функції, які визначають технічну реалізацію та вартість розробки. Головна функція F_0 — розробка гібридної рекомендаційної системи для магазину ігор. На основі цієї функції виділено наступні підфункції:

F_1 — Вибір мови програмування:

а) Python;

б) C++.

F_2— Базовий алгоритм отримання кандидатів (Candidate Retrieval):

а) Матрична факторизація (усічений SVD);

б) Класична User-based колаборативна фільтрація.

F_3 — Алгоритм повторного ранжування (Reranking):

а) Градієнтний бустинг над деревами рішень (XGBoost / LightGBM);

б) Нейромережевий підхід (Neural Collaborative Filtering).

F_4 — Формат збереження проміжних артефактів даних:

а) Колонковий формат Parquet;

б) Текстовий формат CSV.

Виходячи з цих функцій, проаналізуємо переваги та недоліки кожного варіанту за допомогою позитивно-негативної матриці (табл. 4.1).

Таблиця 4.1 — Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1 – Мова програмування	А – Python	Швидка розробка, наявність потужних бібліотек (scikit-learn, pandas)	Нижча швидкість виконання сирого коду
	Б – C++	Максимальна продуктивність та контроль пам'яті	Тривалий час написання коду, складність налагодження
F_2 – Алгоритм відбору	А – Усічений SVD	Висока швидкодія на розріджених матрицях, масштабованість	Потребує підбору кількості латентних факторів
	Б – User-based CF	Висока інтерпретованість рекомендацій	Квадратична обчислювальна складність, не масштабується

Продовження таблиці 4.1

Функції	Варіанти реалізації	Переваги	Недоліки
F_3– Реранжування	А – XGBoost / LightGBM	Відмінна робота з табличними даними, високий ROC-AUC	Схильність до перенавчання без налаштування гіперпараметрів
	Б – Neural CF	Здатність моделювати складні нелінійні взаємодії	Вимагає наявності GPU та великого часу на навчання
F_4– Формат даних	А – Parquet	Швидке читання/запис, сильне стиснення даних	Бінарний формат, не читається звичайними текстовими редакторами
	Б – CSV	Універсальність та простота	Повільна обробка, великий розмір файлів на диску

На основі проведеного аналізу для кожної підфункції робимо вибір на користь найбільш доцільних рішень:

Для F_1 перевагу отримала мова Python. Величезна екосистема бібліотек для Data Science (pandas, NumPy, scikit-learn) дозволяє значно скоротити терміни розробки пайплайну порівняно з C++.

Для F_2 обрано усічений SVD. Він обчислювально ефективніший за Memory-based методи при роботі з десятками мільйонів записів.

Для F_3 обрано ансамблеві моделі XGBoost / LightGBM, оскільки вони є

галузевим стандартом для табличних ознак і не потребують таких значних апаратних потужностей (GPU), як нейромережі.

Для F_4 обрано формат Parquet для ефективного кешування, що критично впливає на швидкість повторних запусків системи.

За результатами аналізу сформуємо два альтернативні варіанти реалізації програмного продукту для подальшого порівняння:

$F_1(\text{а}) - F_2(\text{а}) - F_3(\text{а}) - F_4(\text{а})$ (Python + SVD + XGBoost + Parquet).

$F_1(\text{а}) - F_2(\text{б}) - F_3(\text{а}) - F_4(\text{б})$ (Python + User-based CF + XGBoost + CSV).

4.3 Обґрунтування системи параметрів програмного продукту

Для кількісного оцінювання та порівняння обраних варіантів необхідно встановити систему техніко-економічних параметрів:

1) X_1 – Точність класифікації та ранжування (ROC-AUC, %): відображає здатність системи правильно впорядкувати релевантні об'єкти.

2) X_2 – Швидкість обробки матриці взаємодій (секунди): час, необхідний для генерації кандидатів та оцінки моделі на великих масивах даних.

3) X_3 – Витрати оперативної пам'яті (МБ): визначає ресурсомісткість під час побудови розріджених матриць.

4) X_4 – Час розробки системи (людино-години): відображає трудомісткість написання коду та налаштування пайплайну.

Відповідно до обраних параметрів визначимо їх граничні значення (найгірші, середні та найкращі) у табл. 4.2.

Таблиця 4.2 — Основні параметри програмного продукту

Назва параметра	Позначення	Одиниці виміру	Найгірше	Середнє	Найкраще
Точність моделей	X_1	%	60	75	85
Швидкість розрахунків	X_2	секунди	300	100	10
Витрати пам'яті	X_3	МБ	8000	4000	1000
Час розробки	X_4	люд-год	120	80	40

За даними таблиці 4.2 побудовано графічні характеристики параметрів: точності моделей (рис. 4.1), швидкості розрахунків (рис. 4.2), витрат пам’яті (рис. 4.3) та часу розробки (рис. 4.4).

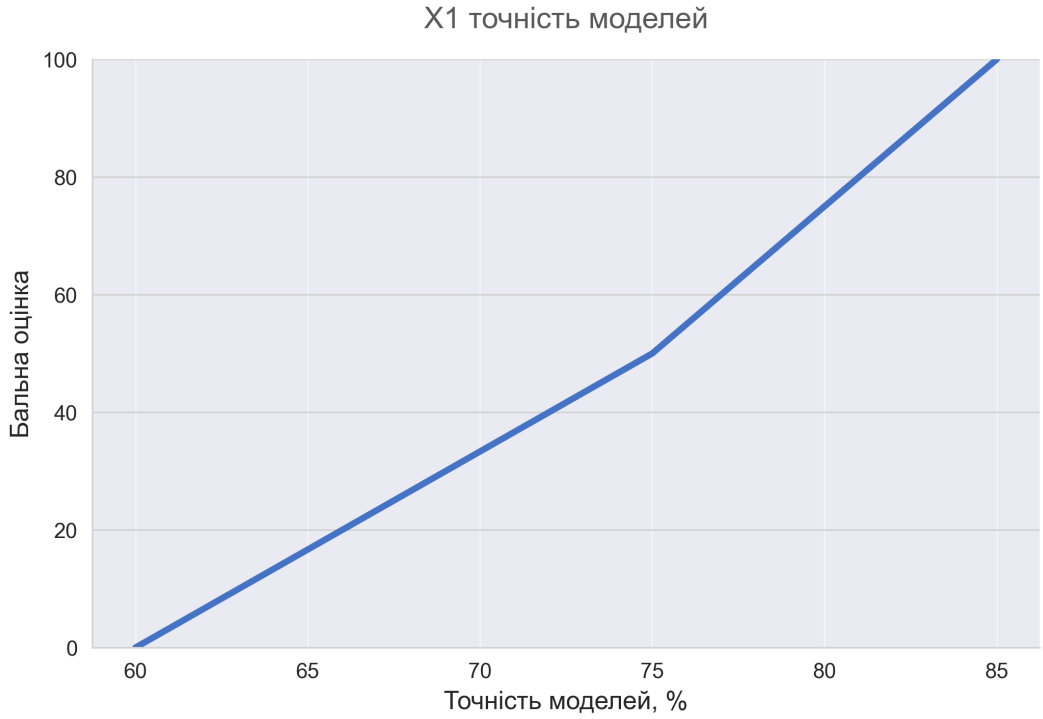


Рисунок 4.1 – X1, точність моделей

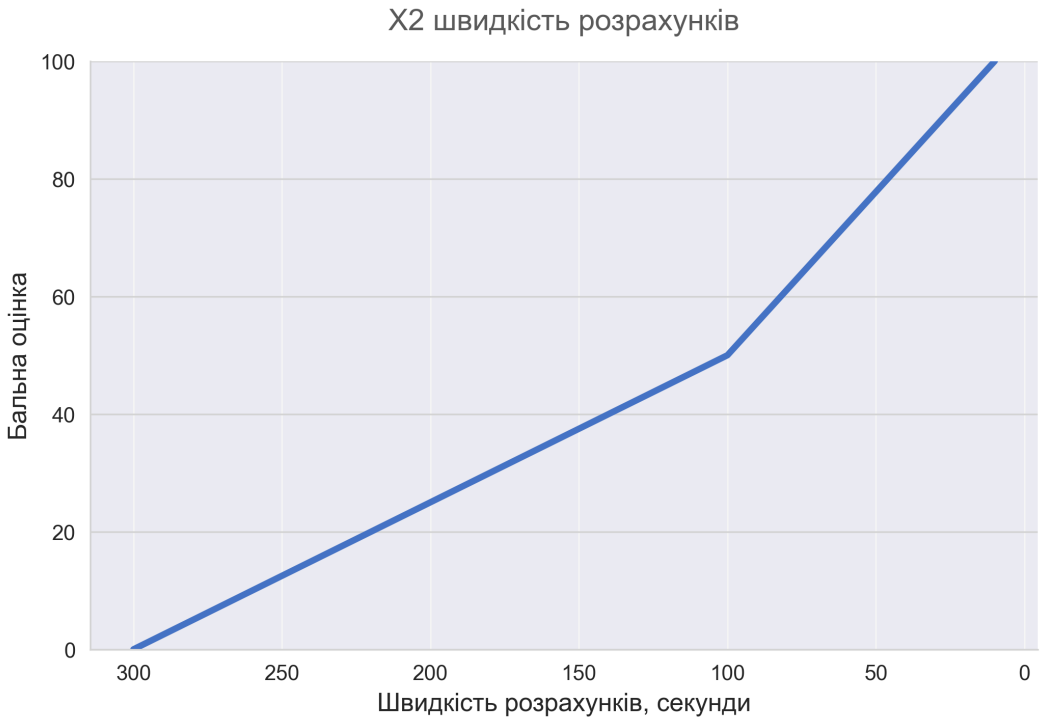


Рисунок 4.2 – X2, швидкість розрахунків

Залежності бальної оцінки від витрат оперативної пам’яті та часу розробки показано на рис. 4.3 та рис. 4.4 відповідно.

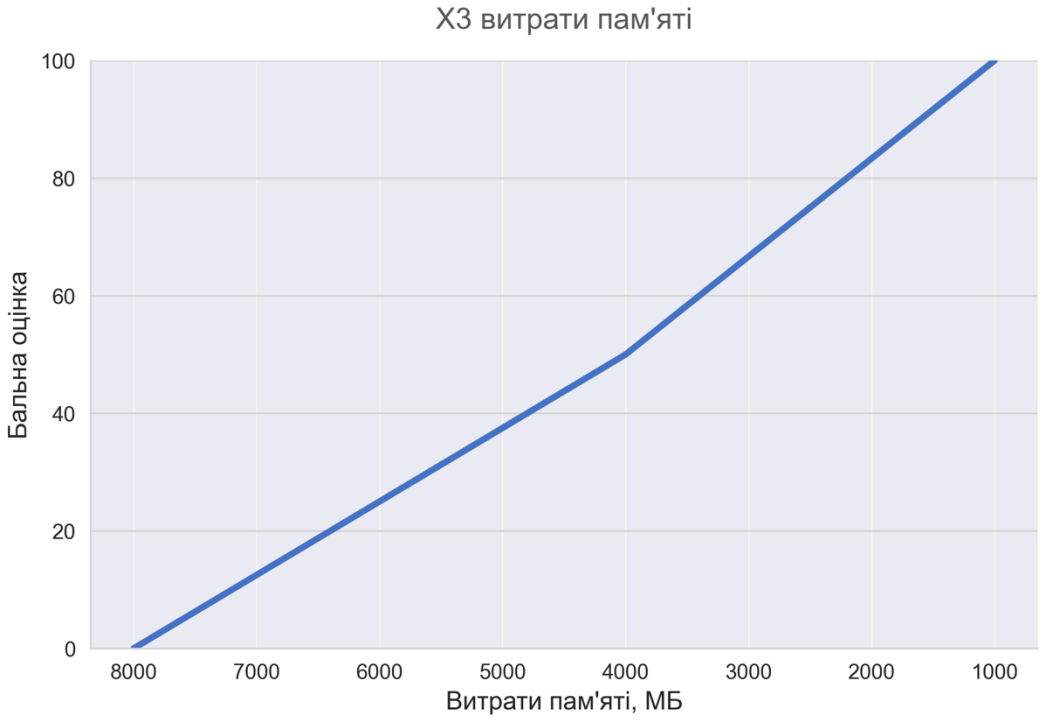


Рисунок 4.3 – X3, витрати пам’яті

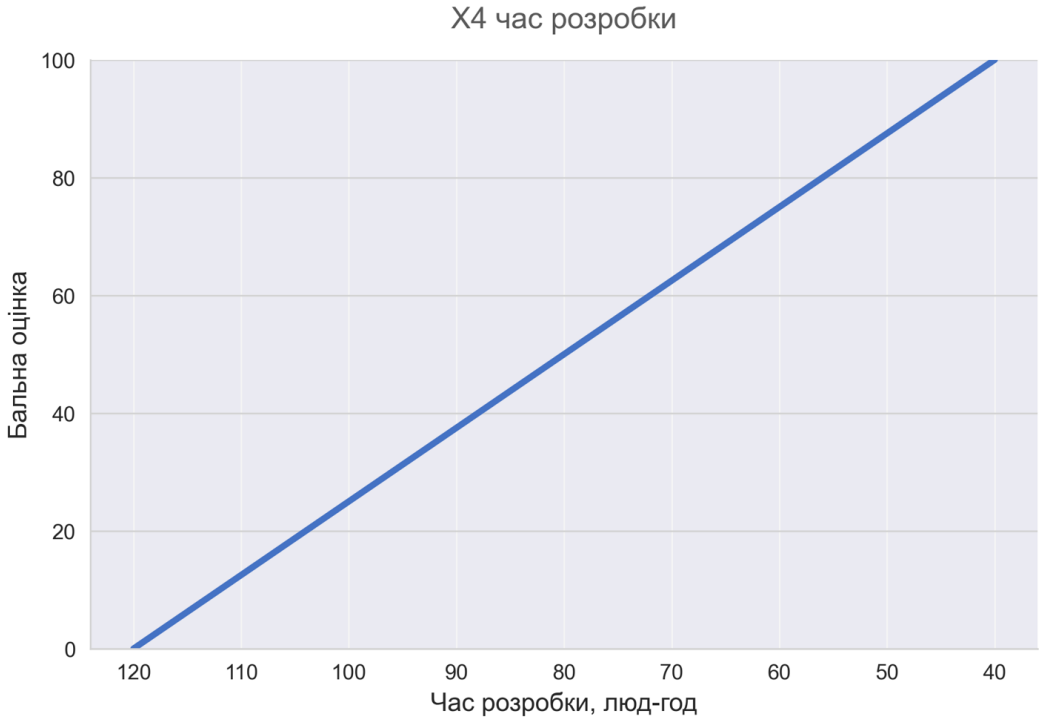


Рисунок 4.4 – X4, час розробки

4.4 Аналіз експертного оцінювання параметрів

Для визначення вагомості кожного параметра залучено експертну групу з 7 осіб. Кожен експерт незалежно присвоїв ранги параметрам від 1 до 4, де 1 — найважливіший параметр. Результати зведено у таблицю 4.3.

Таблиця 4.3 — Результати ранжування параметрів

Параметр	1	2	3	4	5	6	7	Сума рангів R_i	Відхилення Δ_i	Δ_i^2
X_1	1	2	1	1	2	1	2	10	-7.5	56.25
X_2	2	1	2	2	1	2	1	11	-6.5	42.25
X_3	3	4	3	3	4	3	4	24	6.5	42.25
X_4	4	3	4	4	3	4	3	25	7.5	56.25
Разом	10	10	10	10	10	10	10	70	0	197

Перевіримо достовірність експертних оцінок:

а) загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.1)$$

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5 \quad (4.2)$$

в)загальна сума квадратів відхилень:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197. \quad (4.3)$$

г) коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 197}{7^2(4^3 - 4)} = 0{,}8. \quad (4.4)$$

Отримане значення $W = 0.80 > W_k = 0.67$, отже результати ранжування вважаються достовірними. Ступінь переваги визначається за формулою:

$$a_{ij} = \begin{cases} 1.5 \text{ якщо } X_i > X_j \\ 1.0 \text{ якщо } X_i = X_j \\ 0.5 \text{ якщо } X_i < X_j \end{cases}.$$

Результати попарного порівняння параметрів зведено у таблицю 4.4.

Таблиця 4.4 — Попарне порівняння параметрів

Параметри	1	2	3	4	5	6	7	Кінцева оцінка	Числове значення
X1 і X2	>	>	>	>	<	>	>	>	1,5
X1 і X3	>	>	>	>	>	>	>	>	1,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	>	>	<	>	>	>	>	>	1,5
X2 і X4	>	>	>	>	>	>	>	>	1,5
X3 і X4	>	<	>	>	<	>	>	>	1,5

За числовими оцінками переваги складено матрицю $A=\|a_{ij}\|$ та ітеративно (до різниці результатів < 2 %) обчислено коефіцієнти вагомості (табл. 4.5).

Таблиця 4.5 — Розрахунок вагомості параметрів

Параметри	X1	X2	X3	X4	Σ (1 іт.)	Кві (1 іт.)	Σ (2 іт.)	Кві (2 іт.)	Σ (3 іт.)	Кві (3 іт.)
X1	1	1,5	1,5	1,5	5,5	0,3438	21,25	0,3602	77,875	0,3605
X2	0,5	1	1,5	1,5	4,5	0,2812	16,25	0,2754	59,125	0,2737
X3	0,5	0,5	1	1,5	3,5	0,2188	12,25	0,2076	44,875	0,2078
X4	0,5	0,5	0,5	1	2,5	0,1562	9,25	0,1568	34,125	0,1580
Σ						1,0000		1,0000		1,0000

Коефіцієнти вагомості $K_{\text{ві}}$ розраховуються ітеративно. Значення вагомості після другої ітерації (відхилення < 2%) склали: $K_{\text{в1}} = 0.3605$, $K_{\text{в2}} = 0.2737$, $K_{\text{в3}} = 0.2078$, $K_{\text{в4}} = 0.1580$.

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту реалізації окремо. Коефіцієнт технічного рівня для кожного варіанта розраховується за формулою:

$$K_K = \sum_{i=1}^{n} K_{\text{ві}} \cdot B_i (4.6)$$

де $K_{\text{ві}}$– коефіцієнт вагомості i-го параметра; B_i – бальна оцінка i-го параметра (від 0 до 100).

Бальні оцінки визначаються на основі абсолютних значень параметрів для кожного варіанту, виходячи з графіків (які були побудовані на основі табл. 4.2).

Розрахунок показників рівня якості наведено у таблиці 4.6.

Таблиця 4.6 — Розрахунок показників рівня якості варіантів реалізації функцій

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1, F2, F3, F4	Варіант 1 (а-а-а-а)	X1	83	90	0,3605	32,45
		X2	15	95	0,2737	26,00
		X3	2000	85	0,2078	17,66
		X4	50	85	0,1580	13,43
F1, F2, F3, F4	Варіант 2 (а-б-а-б)	X1	75	60	0,3605	21,63
		X2	200	35	0,2737	9,58
		X3	6000	30	0,2078	6,23
		X4	90	40	0,1580	6,32

За формулою визначаємо рівень якості кожного варіанту:

$$K_{К1} = 32.45 + 26.00 + 17.66 + 13.43 = 89.54$$

$$K_{К2} = 21.63 + 9.58 + 6.23 + 6.32 = 43.76$$

За результатами розрахунку, Варіант 1 має значно вищий коефіцієнт рівня якості. Таким чином, для подальшої реалізації обрано саме його (використання усіченого SVD, XGBoost та формату Parquet).

4.6 Економічний аналіз варіантів розробки ПП

Обидва варіанти охоплюють два ключові завдання:

1. Розробка базового SVD алгоритму та обробки матриці (група складності 2, ступінь новизни Б): базова трудомісткість $T_R = 36$ люд-днів.
2. Розробка ML-моделей (XGBoost) та ансамблювання (група 3, ступінь Б): $T_R = 64$ люд-днів.

Загальна трудомісткість обчислюється за формулою:

$$T_O = T_R \cdot K_P \cdot K_{\text{СК}} \cdot K_M \cdot K_{\text{СТ}} \cdot K_{\text{СТ.М}}$$

де $K_P = 1.21$, $K_{\text{СК}} = 1$, $K_M = 0.85$(Python), $K_{\text{СТ}} = 0.8$, $K_{\text{СТ.М}} = 1$.

Для першого та другого завдання відповідно:

$$T_1 = 36 \cdot 1.21 \cdot 1 \cdot 0.85 \cdot 0.8 \cdot 1 \approx 29.62 \text{ люд-днів}$$

$$T_2 = 64 \cdot 1.21 \cdot 1 \cdot 0.85 \cdot 0.8 \cdot 1 \approx 52.66 \text{ люд-днів}$$

Загальна трудомісткість для кожного варіанту у годинах (Варіант 2 потребує додаткових 10 днів на обробку неефективного формату CSV та Memory-based методів):

$$T_I = (29.62 + 52.66) \cdot 8 = 658.24 \text{ люд-год}$$

$$T_{II} = (29.62 + 52.66 + 10.0) \cdot 8 = 738.24 \text{ люд-год}$$

Середня погодинна ставка (за окладами розробника 34 000 грн та керівника 52 000 грн):

$$C_{\text{Ч}} = \frac{34000 + 52000}{2 \cdot 21.1 \cdot 8} = 254.74 \text{ грн/год}$$

Заробітна плата з урахуванням додаткової (КД = 1.2):

$$C_{\text{ЗП}} = C_{\text{Ч}} \cdot T_i \cdot K_{\text{Д}}$$

$$C_{\text{ЗП1}} = 254.74 \cdot 658.24 \cdot 1.2 = 201215.55 \text{ грн}$$

$$C_{ЗП2} = 254.74 \cdot 738.24 \cdot 1.2 = 225670.52 \text{ грн}$$

Відрахування на єдиний соціальний внесок (22%):

$$C_{ВІД1} = 201215.55 \cdot 0.22 = 44267.42 \text{ грн}$$

$$C_{ВІД2} = 225670.52 \cdot 0.22 = 49647.51 \text{ грн}$$

Визначимо витрати на оплату однієї машино-години. ЕОМ обслуговує розробника з окладом 34 000 грн, коефіцієнт зайнятості КЗ = 0,2:

$$C_{Г} = 12 \cdot 34000 \cdot 0.2 = 81600 \text{ грн}$$

З урахуванням додаткової заробітної плати: $C_{ЗП(ЕОМ)} = 81600 \cdot (1 + 0.2) = 97920$ грн

Відрахування на єдиний соціальний внесок (22%):$C_{ВІД(ЕОМ)} = 97920 \cdot 0.22 = 21542.40$ грн

Амортизаційні відрахування (норма амортизації 25%, вартість ПК 55 000 грн):$C_{А} = 1.15 \cdot 0.25 \cdot 55000 = 15812.50$ грн

Витрати на ремонт та профілактику:$C_{Р} = 1.15 \cdot 55000 \cdot 0.05 = 3162.50$ грн

Ефективний річний фонд часу роботи ПК:$T_{ЕФ} = (365 - 104 - 11 - 8) \cdot 8 \cdot 0.85 = 1645.60$ год

Витрати на електроенергію (тариф 13,64 грн/кВт·год — 2-й клас напруги, потужність 0,3 кВт):$C_{ЕЛ} = 1645.60 \cdot 0.3 \cdot 0.2 \cdot 13.64 = 1346.76$ грн

Накладні витрати:$C_{Н} = 55000 \cdot 0.67 = 36850$ грн

Річні експлуатаційні витрати: $C_{ЕКС} = 97920 + 21542.40 + 15812.50 + 3162.50 + 1346.76 + 36850 = 176634.16$ грн

Собівартість однієї машино-години становить як відношення річних експлуатаційних витрат до ефективного фонду часу: $C_{М-Г} = 176634.16/1645.60 = 107.34$ грн/год. Витрати на машинний час:

$$C_{M} = C_{М-Г} \cdot T_{i}$$

$$C_{M1} = 107.34 \cdot 658.24 = 70655.48 \text{ грн}$$

$$C_{M2} = 107.34 \cdot 738.24 = 79242.68 \text{ грн}$$

Накладні витрати (67% від базової заробітної плати):

$$C_{\text{Н1}} = 201215.55 \cdot 0.67 = 134814.42 \text{ грн}$$

$$C_{\text{Н2}} = 225670.52 \cdot 0.67 = 151199.25 \text{ грн}$$

Повна вартість розробки:

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

$$C_{\text{ПП1}} = 201215.55 + 44267.42 + 70655.48 + 134814.42 = 450952.87 \text{ грн}$$

$$C_{\text{ПП2}} = 225670.52 + 49647.51 + 79242.68 + 151199.25 = 505759.96 \text{ грн}$$

4.7 Вибір кращого варіанту ПП за техніко-економічним рівнем

Коефіцієнт техніко-економічного рівня:

$$K_{\text{ТЕР}j} = \frac{K_{Kj}}{C_{\text{Ф}j}}$$

$$K_{\text{ТЕР1}} = \frac{89.54}{450952.87} = 1.99 \cdot 10^{-4}$$

$$K_{\text{ТЕР2}} = \frac{43.76}{505759.96} = 0.87 \cdot 10^{-4}$$

Порівнявши отримані значення, встановлено, що перший варіант має вищий коефіцієнт техніко-економічного рівня. Отже, розробка пайплайну на основі усіченого SVD, XGBoost та формату Parquet забезпечує кращу якість рекомендацій при менших витратах на обчислювальні ресурси та розробку.

4.8 Висновки до розділу 4

У цьому розділі було проведено функціонально-вартісний аналіз програмного продукту для формування персоналізованих рекомендацій комп'ютерних ігор.

Визначено функціональну структуру продукту та сформовано два варіанти реалізації. Проведено експертне оцінювання, яке встановило, що найбільш пріоритетним параметром для системи є точність класифікації та ранжування ($K_{в1} = 0.3605$). Розрахунок показників рівня якості показав значну перевагу Варіанту 1 ($К1 = 89.54$ проти $K_{К2} = 43.76$), що зумовлено високою ефективністю матричної факторизації та ансамблевих моделей на розріджених даних.

Економічний розрахунок довів, що повна вартість розробки обраного варіанту склала 450 952,87 грн, що на 54 807,09 грн менше, ніж вартість розробки альтернативного варіанту. За критерієм техніко-економічного рівня оптимальним визначено Варіант 1 (Python, SVD, XGBoost, Parquet), який і було застосовано у практичній реалізації дипломної роботи.

ВИСНОВКИ

У дипломній роботі розроблено та досліджено гібридну рекомендаційну систему для магазину комп’ютерних ігор на основі методів колаборативної фільтрації та машинного навчання.

Основні результати роботи:

— проведено огляд підходів до побудови рекомендаційних систем та обґрунтовано вибір гібридного підходу, що поєднує матричну факторизацію та повторне ранжування класифікатором;

— виконано дослідницький аналіз датасету, що виявив дисбаланс класів (85,8% позитивних) та правосторонній розподіл ігрового часу;

— реалізовано двоетапний пайплайн рекомендацій на основі усіченого SVD та повторного ранжування;

— за протоколом Sampled LOO встановлено, що чистий SVD забезпечує найкращу якість ранжування, тоді як наївне ML-реранжування не покращує рекомендацій через відсутність персоналізованих ознак.

Практичне значення роботи полягає у створенні відтворюваного програмного пайплайну, придатного для інтеграції в платформи цифрової дистрибуції відеоігор. Напрямами подальших досліджень є застосування персоналізованих ознак рівня пари «користувач–гра», текстових ознак відгуків (TF-IDF, трансформерні ембедінги), нейромережевих методів (Neural Collaborative Filtering) та реалізація REST API для промислового розгортання системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Valve Corporation. Steam Store. URL: https://store.steampowered.com/ (дата звернення: 15.04.2026).

2. Ricci F., Rokach L., Shapira B. Recommender Systems Handbook. 2nd ed. New York : Springer, 2015. 1003 p. URL: https://link.springer.com/book/10.1007/978-1-4899-7637-6 (дата звернення: 16.04.2026).

3. Aggarwal C. C. Recommender Systems: The Textbook. Cham : Springer, 2016. 498 p. URL: https://link.springer.com/book/10.1007/978-3-319-29659-3 (дата звернення: 16.04.2026).

4. Kozyriev A. Game Recommendations on Steam. Kaggle, 2023. URL: https://www.kaggle.com/datasets/antonkozyriev/game-recommendations-on-steam (дата звернення: 18.04.2026).

5. Burke R. Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction. 2002. Vol. 12, no. 4. P. 331–370. URL: https://doi.org/10.1023/A:1021240730564 (дата звернення: 18.04.2026).

6. Su X., Khoshgoftaar T. M. A Survey of Collaborative Filtering Techniques. Advances in Artificial Intelligence. 2009. Vol. 2009. P. 1–19. URL: https://doi.org/10.1155/2009/421425 (дата звернення: 20.04.2026).

7. Hu Y., Koren Y., Volinsky C. Collaborative Filtering for Implicit Feedback Datasets. Proceedings of the 8th IEEE International Conference on Data Mining. Pisa, 2008. P. 263–272. URL: https://doi.org/10.1109/ICDM.2008.22 (дата звернення: 20.04.2026).

8. Koren Y., Bell R., Volinsky C. Matrix Factorization Techniques for Recommender Systems. Computer. 2009. Vol. 42, no. 8. P. 30–37. URL: https://doi.org/10.1109/MC.2009.263 (дата звернення: 20.04.2026).

9. He X., Liao L., Zhang H., Nie L., Hu X., Chua T.-S. Neural Collaborative Filtering. Proceedings of the 26th International Conference on World Wide Web.

Perth, Australia, 2017. P. 173–182. URL: https://doi.org/10.1145/3038912.3052569 (дата звернення: 21.04.2026).

10. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning. 2nd ed. New York : Springer, 2009. 745 p. URL: https://hastie.su.domains/ElemStatLearn/ (дата звернення: 21.04.2026).

11. Breiman L. Random Forests. Machine Learning. 2001. Vol. 45, no. 1. P. 5–32. URL: https://doi.org/10.1023/A:1010933404324 (дата звернення: 21.04.2026).

12. Järvelin K., Kekäläinen J. Cumulated Gain-based Evaluation of IR Techniques. ACM Transactions on Information Systems. 2002. Vol. 20, no. 4. P. 422–446. URL: https://doi.org/10.1145/582415.582418 (дата звернення: 22.04.2026).

13. Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016. P. 785–794. URL: https://doi.org/10.1145/2939672.2939785 (дата звернення: 22.04.2026).

14. Ke G., Meng Q., Finley T. et al. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 3146–3154. URL: https://papers.nips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html (дата звернення: 23.04.2026).

15. Pedregosa F. et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011. Vol. 12. P. 2825–2830. URL: https://www.jmlr.org/papers/v12/pedregosa11a.html (дата звернення: 23.04.2026).

16. McKinney W. Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference. 2010. P. 56–61. URL: https://doi.org/10.25080/Majora-92bf1922-00a (дата звернення: 23.04.2026).

17. Sarwar B., Karypis G., Konstan J., Riedl J. Item-based Collaborative Filtering Recommendation Algorithms. Proceedings of the 10th International

Conference on World Wide Web, Hong Kong, 2001. P. 285–295. URL: https://doi.org/10.1145/371920.372071 (дата звернення: 24.04.2026).

18. Rendle S., Freudenthaler C., Gantner Z., Schmidt-Thieme L. BPR: Bayesian Personalized Ranking from Implicit Feedback. Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, Montreal, Canada, 2009. P. 452–461. URL: https://dl.acm.org/doi/10.5555/1795114.1795167 (дата звернення: 24.04.2026).

19. Harper F. M., Konstan J. A. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems, 2015. Vol. 5, no. 4. P. 1–19. URL: https://doi.org/10.1145/2827872 (дата звернення: 25.04.2026).

20. Cremonesi P., Koren Y., Turrin R. Performance of Recommender Algorithms on Top-N Recommendation Tasks. Proceedings of the 4th ACM Conference on Recommender Systems, Barcelona, Spain, 2010. P. 39–46. URL: https://doi.org/10.1145/1864708.1864721 (дата звернення: 25.04.2026).

21. Halko N., Martinsson P.-G., Tropp J. A. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. SIAM Review. 2011. Vol. 53, no. 2. P. 217–288. URL: https://doi.org/10.1137/090771806 (дата звернення: 26.04.2026).

22. Яковлева А. П., Спекторський І. Я. Методи оптимізації. Комп'ютерний практикум. Київ: КПІ ім. Ігоря Сікорського, 2021. 82 с.

23. Яковлева А. П. Методи оптимізації та дослідження операцій. Конспект лекцій. Київ: КПІ ім. Ігоря Сікорського, 2022. 71 с.

ДОДАТОК А
ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, classification_report, roc_auc_score, roc_curve
)

sns.set_style('whitegrid')
plt.rcParams['figure.figsize'] = (10, 5)

print("✓ Бібліотеки імпортовано успішно!")
#%% md
## 2. Load Data (Sample Only)
#%%
import os
data_path = 'recommendations.csv'

# Read only sample
df = pd.read_csv(data_path)

print(f"\n✓ Датасет(recommendations) завантажився: {df.shape}")
print(f"Колонки: {df.columns.tolist()}")
print(f"Memory usage: {df.memory_usage(deep=True).sum() / (1024**2):.2f} MB")
#%%
# Display first rows
df.head()
#%%
# Dataset info
```

```
print("Типи колонок:")
print("="*80)
df.info()
print("\nОсновні статистики по колонкам:")
print(df.describe())
#%% md
## 3. Data Quality Check
#%%
print("Пропущені значення:")
print(df.isnull().sum())
print(f"\nДублікати: {df.duplicated().sum()}")
#%% md
## 4. EDA - Target Variable
#%%
# Target variable
target_col = 'is_recommended'

print(f"Target: {target_col}")
print("\nDistribution:")
print(df[target_col].value_counts())
print("\nPercentages:")
print(df[target_col].value_counts(normalize=True) * 100)

# Visualization
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

df[target_col].value_counts().plot(kind='bar', ax=ax1, color=['#2ecc71', '#e74c3c'])
ax1.set_title('Recommendation Distribution', fontsize=14, fontweight='bold')
ax1.set_ylabel('Count')
ax1.set_xticklabels(['Recommended', 'Not Recommended'], rotation=0)

df[target_col].value_counts().plot(kind='pie', ax=ax2, autopct='%1.1f%%',
                                   colors=['#2ecc71', '#e74c3c'])
ax2.set_title('Recommendation %', fontsize=14, fontweight='bold')
ax2.set_ylabel('')

plt.tight_layout()
plt.show()
#%% md
## 5. EDA - Hours Analysis
#%%
# Hours analysis
```

```
print("Hours Statistics:")
print(df['hours'].describe())

fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# Distribution
axes[0].hist(df['hours'], bins=50, color='skyblue', edgecolor='black')
axes[0].set_title('Hours Distribution', fontweight='bold')
axes[0].set_xlabel('Hours')
axes[0].set_ylabel('Frequency')

# Log scale
axes[1].hist(np.log1p(df['hours']), bins=50, color='coral', edgecolor='black')
axes[1].set_title('Hours (Log Scale)', fontweight='bold')
axes[1].set_xlabel('Log(Hours + 1)')
axes[1].set_ylabel('Frequency')

# By recommendation
df.boxplot(column='hours', by=target_col, ax=axes[2])
axes[2].set_title('Hours by Recommendation', fontweight='bold')
axes[2].set_xlabel('Recommended')

plt.tight_layout()
plt.show()
#%% md
## 6. Feature Engineering (Simple)
#%%
# Create simple features from available columns
df_ml = df.copy()

# Handle missing values
df_ml = df_ml.fillna(0)

# Feature engineering
df_ml['hours_log'] = np.log1p(df_ml['hours'])
df_ml['helpful_ratio'] = df_ml['helpful'] / (df_ml['helpful'] + df_ml['funny'] + 1)
df_ml['engagement_score'] = df_ml['helpful'] + df_ml['funny']
df_ml['is_high_playtime'] = (df_ml['hours'] > df_ml['hours'].median()).astype(int)

print("✓ Features created:")
print("  - hours_log")
print("  - helpful_ratio")
```

```
print("  - engagement_score")
print("  - is_high_playtime")

# Clean memory
del df
gc.collect()
#%% md
## 7. Prepare Features for ML (NO X_combined complexity)
#%%
# Simple feature selection - NO text processing, NO X_combined
feature_columns = [
    'helpful',
    'funny',
    'hours',
    'hours_log',
    'helpful_ratio',
    'engagement_score',
    'is_high_playtime'
]

# Features (X) and target (y)
X = df_ml[feature_columns].copy()
y = df_ml[target_col].astype(int).copy()

print(f"Features (X): {X.shape}")
print(f"Target (y): {y.shape}")
print(f"\nFeatures used: {feature_columns}")
print(f"\nClass distribution:")
print(y.value_counts())

# Clean memory
del df_ml
gc.collect()
#%% md
## 8. Train-Test Split & Scaling
#%%
# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f"Training set: {X_train.shape}")
```

```
print(f"Test set: {X_test.shape}")

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print(f"\n✓ Data ready for training!")

# Clean memory
del X, y
gc.collect()
#%% md
## 9. Model Training (3 Lightweight Models)
#%%
# Initialize results
results = {}
print("Training 3 lightweight models...\n")
#%%
# Model 1: Logistic Regression
print("[1/3] Training Logistic Regression...")
gc.collect()

model = LogisticRegression(random_state=42, max_iter=300, n_jobs=1)
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)
y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]

results['Logistic Regression'] = {
    'model': model,
    'accuracy': accuracy_score(y_test, y_pred),
    'precision': precision_score(y_test, y_pred, zero_division=0),
    'recall': recall_score(y_test, y_pred, zero_division=0),
    'f1_score': f1_score(y_test, y_pred, zero_division=0),
    'roc_auc': roc_auc_score(y_test, y_pred_proba),
    'predictions': y_pred,
    'predictions_proba': y_pred_proba
}

print(f"  Accuracy: {results['Logistic Regression']['accuracy']:.4f}")
print(f"  F1-Score: {results['Logistic Regression']['f1_score']:.4f}")
```

```
del model, y_pred, y_pred_proba
gc.collect()
#%%
# Model 2: Decision Tree
print("\n[2/3] Training Decision Tree...")
gc.collect()

model = DecisionTreeClassifier(random_state=42, max_depth=8)
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)
y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]

results['Decision Tree'] = {
    'model': model,
    'accuracy': accuracy_score(y_test, y_pred),
    'precision': precision_score(y_test, y_pred, zero_division=0),
    'recall': recall_score(y_test, y_pred, zero_division=0),
    'f1_score': f1_score(y_test, y_pred, zero_division=0),
    'roc_auc': roc_auc_score(y_test, y_pred_proba),
    'predictions': y_pred,
    'predictions_proba': y_pred_proba
}

print(f"  Accuracy: {results['Decision Tree']['accuracy']:.4f}")
print(f"  F1-Score: {results['Decision Tree']['f1_score']:.4f}")

del model, y_pred, y_pred_proba
gc.collect()
#%%
# Model 3: Random Forest (Minimal)
print("\n[3/3] Training Random Forest...")
gc.collect()

model = RandomForestClassifier(
    random_state=42,
    n_estimators=20,
    max_depth=8,
    n_jobs=1,
    max_samples=0.7
)
```

```
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)
y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]

results['Random Forest'] = {
    'model': model,
    'accuracy': accuracy_score(y_test, y_pred),
    'precision': precision_score(y_test, y_pred, zero_division=0),
    'recall': recall_score(y_test, y_pred, zero_division=0),
    'f1_score': f1_score(y_test, y_pred, zero_division=0),
    'roc_auc': roc_auc_score(y_test, y_pred_proba),
    'predictions': y_pred,
    'predictions_proba': y_pred_proba
}

print(f"  Accuracy: {results['Random Forest']['accuracy']:.4f}")
print(f"  F1-Score: {results['Random Forest']['f1_score']:.4f}")

del model, y_pred, y_pred_proba
gc.collect()

print("\n✓ All models trained!")
#%% md
## 10. Model Comparison
#%%
# Comparison table
comparison_df = pd.DataFrame({
    'Model': list(results.keys()),
    'Accuracy': [results[m]['accuracy'] for m in results],
    'Precision': [results[m]['precision'] for m in results],
    'Recall': [results[m]['recall'] for m in results],
    'F1-Score': [results[m]['f1_score'] for m in results],
    'ROC-AUC': [results[m]['roc_auc'] for m in results]
}).sort_values('F1-Score', ascending=False)

print("Model Performance Comparison:")
print("="*80)
print(comparison_df.to_string(index=False))

best_model_name = comparison_df.iloc[0]['Model']
print(f"\n🏆 Best Model: {best_model_name}")
```

```
print(f"   F1-Score: {comparison_df.iloc[0]['F1-Score']:.4f}")
#%%
# Visualize comparison
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# All metrics
x = np.arange(len(comparison_df))
width = 0.15
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score', 'ROC-AUC']

for i, metric in enumerate(metrics):
    axes[0].bar(x + i*width, comparison_df[metric], width, label=metric)

axes[0].set_xlabel('Model')
axes[0].set_ylabel('Score')
axes[0].set_title('Model Performance Comparison', fontweight='bold', fontsize=14)
axes[0].set_xticks(x + width * 2)
axes[0].set_xticklabels(comparison_df['Model'], rotation=15)
axes[0].legend()
axes[0].grid(alpha=0.3)

# F1-Score
axes[1].barh(comparison_df['Model'], comparison_df['F1-Score'], color='steelblue')
axes[1].set_xlabel('F1-Score')
axes[1].set_title('F1-Score by Model', fontweight='bold', fontsize=14)
axes[1].grid(alpha=0.3, axis='x')

plt.tight_layout()
plt.show()
#%% md
## 11. Best Model Evaluation
#%%
# Confusion Matrix
y_pred_best = results[best_model_name]['predictions']
cm = confusion_matrix(y_test, y_pred_best)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Recommended', 'Recommended'],
            yticklabels=['Not Recommended', 'Recommended'])
plt.title(f'Confusion Matrix - {best_model_name}', fontweight='bold', fontsize=14)
plt.ylabel('Actual')
```

```
plt.xlabel('Predicted')
plt.show()
#%%
# Classification Report
print("Classification Report:")
print("="*80)
print(classification_report(y_test, y_pred_best,
                            target_names=['Not Recommended', 'Recommended']))
#%%
# ROC Curve
y_pred_proba_best = results[best_model_name]['predictions_proba']
fpr, tpr, _ = roc_curve(y_test, y_pred_proba_best)
roc_auc_val = roc_auc_score(y_test, y_pred_proba_best)

plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC (AUC = {roc_auc_val:.4f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random')
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title(f'ROC Curve - {best_model_name}', fontweight='bold', fontsize=14)
plt.legend()
plt.grid(alpha=0.3)
plt.show()
#%% md
## 12. Feature Importance
#%%
# Feature importance
best_model = results[best_model_name]['model']

if hasattr(best_model, 'feature_importances_'):
    feat_imp = pd.DataFrame({
        'Feature': feature_columns,
        'Importance': best_model.feature_importances_
    }).sort_values('Importance', ascending=False)

    print("Feature Importance:")
    print(feat_imp)

    plt.figure(figsize=(10, 6))
    plt.barh(feat_imp['Feature'], feat_imp['Importance'], color='teal')
    plt.xlabel('Importance')
    plt.title(f'Feature Importance - {best_model_name}', fontweight='bold', fontsize=14)
```

```
    plt.gca().invert_yaxis()
    plt.tight_layout()
    plt.show()
elif hasattr(best_model, 'coef_'):
    coef = pd.DataFrame({
        'Feature': feature_columns,
        'Coefficient': best_model.coef_[0]
    }).sort_values('Coefficient', key=abs, ascending=False)

    print("Feature Coefficients:")
    print(coef)

    plt.figure(figsize=(10, 6))
    colors = ['green' if x > 0 else 'red' for x in coef['Coefficient']]
    plt.barh(coef['Feature'], coef['Coefficient'], color=colors)
    plt.xlabel('Coefficient')
    plt.title(f'Feature Coefficients - {best_model_name}', fontweight='bold', fontsize=14)
    plt.axvline(x=0, color='black', linestyle='--', linewidth=0.8)
    plt.gca().invert_yaxis()
    plt.tight_layout()
    plt.show()
#%% md
## 13. Save Results
#%%
# Save predictions
predictions_df = pd.DataFrame({
    'Actual': y_test,
    'Predicted': y_pred_best,
    'Probability': y_pred_proba_best
})
predictions_df.to_csv('predictions.csv', index=False)

# Save comparison
comparison_df.to_csv('model_comparison.csv', index=False)

print("✓ Results saved!")
print("  - predictions.csv")
print("  - model_comparison.csv")
#%% md
## 14. Summary
#%%
print("="*80)
```

```
print("PROJECT SUMMARY")
print("="*80)
print(f"\nDataset: Steam Game Recommendations")
print(f"Original Size: 41+ million rows")
print(f"Sample Used: 100,000 rows (0.24% of full dataset)")
print(f"Features: {len(feature_columns)} numerical features (no text processing)")
print(f"\n🏆 Best Model: {best_model_name}")
print("="*80)
print(f"Accuracy:  {results[best_model_name]['accuracy']:.4f}")
print(f"Precision: {results[best_model_name]['precision']:.4f}")
print(f"Recall:    {results[best_model_name]['recall']:.4f}")
print(f"F1-Score:  {results[best_model_name]['f1_score']:.4f}")
print(f"ROC-AUC:   {results[best_model_name]['roc_auc']:.4f}")
print("="*80)
print("\n✓ Memory-Optimized Pipeline Complete!")
print("="*80)
#%% md
## 15.Import as pkl file
#%%
# Save Logistic Regression Model
import pickle

# Get the trained Logistic Regression model
lr_model = results['Logistic Regression']['model']

# Save model and scaler as pickle files
with open('logistic_regression_model.pkl', 'wb') as f:
    pickle.dump(lr_model, f)

with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)

# Save feature names for reference
with open('feature_columns.pkl', 'wb') as f:
    pickle.dump(feature_columns, f)

print("✓ Model saved successfully!")
print("Files created:")
print("  - logistic_regression_model.pkl")
print("  - scaler.pkl")
print("  - feature_columns.pkl")
print(f"\nModel: Logistic Regression")
```

```
print(f"Features: {feature_columns}")
print(f"Accuracy: {results['Logistic Regression']['accuracy']:.4f}")

#%% md
## 17. Recommendation System — Data Preparation
#
# 🏗️ Двоетапний пайплайн:
#   Stage 1 — User-Based Collaborative Filtering (знайти кандидатів)
#   Stage 2 — Classifier Reranking (відранжувати через LR з секцій 9-12)
#%%
from scipy.sparse import csr_matrix
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import LabelEncoder

print("Завантажуємо ПОВНИЙ датасет для recommendation system...")

# Беремо всі рядки — пам'яті вистачає
df_rec = pd.read_csv(
    'recommendations.csv',
    usecols=['user_id', 'app_id', 'is_recommended', 'hours', 'helpful', 'funny']
)

print(f"✓ Завантажено: {df_rec.shape[0]:,} рядків")

# --- Фільтруємо активних юзерів та популярні ігри ---
user_counts = df_rec['user_id'].value_counts()
game_counts = df_rec['app_id'].value_counts()

active_users  = user_counts[user_counts >= 5].index
popular_games = game_counts[game_counts >= 30].index

df_cf = df_rec[
    df_rec['user_id'].isin(active_users) &
    df_rec['app_id'].isin(popular_games)
].copy()

print(f"\nПісля фільтрації (≥5 відгуків на юзера, ≥30 на гру):")
print(f"  Записів : {df_cf.shape[0]:,}")
print(f"  Юзерів  : {df_cf['user_id'].nunique():,}")
print(f"  Ігор    : {df_cf['app_id'].nunique():,}")

# --- Агрегати по іграх (для Stage 2) ---
```

```
game_stats = df_rec.groupby('app_id').agg(
    avg_helpful   = ('helpful',       'mean'),
    avg_funny     = ('funny',         'mean'),
    avg_hours     = ('hours',         'mean'),
    review_count  = ('is_recommended', 'count'),
    positive_rate = ('is_recommended', 'mean')
).reset_index()

# --- Середні години по кожному юзеру ---
user_avg_hours = df_rec.groupby('user_id')['hours'].mean()

print("\n✓ Все готово!")
gc.collect()
#%% md
## 18. Stage 1 — User-Based Collaborative Filtering

#%%
user_enc = LabelEncoder()
game_enc = LabelEncoder()

df_cf['user_idx'] = user_enc.fit_transform(df_cf['user_id'])
df_cf['game_idx'] = game_enc.fit_transform(df_cf['app_id'])

n_users = int(df_cf['user_idx'].max()) + 1
n_games = int(df_cf['game_idx'].max()) + 1

# Sparse user-item матриця; значення = is_recommended (0 / 1)
user_item_matrix = csr_matrix(
    (df_cf['is_recommended'].astype(float).values,
     (df_cf['user_idx'].values, df_cf['game_idx'].values)),
    shape=(n_users, n_games)
)

sparsity = 1 - user_item_matrix.nnz / (n_users * n_games)
print(f"User-Item матриця: {n_users:,} юзерів × {n_games:,} ігор")
print(f"  Заповнених комірок : {user_item_matrix.nnz:,}")
print(f"  Sparsity           : {sparsity:.3%}")


def get_cf_candidates(target_user_id, top_n=30, n_similar=50, sim_sample=3000):
    """
    Stage 1: повертає DataFrame [app_id, cf_score] — кандидати через User-CF.
```

```
    sim_sample — скільки випадкових юзерів беремо для обчислення подібності.
    """
    if target_user_id not in user_enc.classes_:
        return None

    target_idx = user_enc.transform([target_user_id])[0]
    target_vec = user_item_matrix[target_idx]

    # Семплюємо підмножину юзерів для cosine similarity
    sample_size = min(sim_sample, n_users)
    rng = np.random.RandomState(42)
    sample_idx = rng.choice(n_users, sample_size, replace=False)

    sims = cosine_similarity(target_vec, user_item_matrix[sample_idx])[0]

    # Топ-K схожих (виключаємо самого юзера)
    top_k = [p for p in np.argsort(sims)[::-1]
             if sample_idx[p] != target_idx][:n_similar]

    seen = set(target_vec.indices)
    scores = {}
    for pos in top_k:
        orig = sample_idx[pos]
        w = sims[pos]
        row = user_item_matrix[orig]
        for g, r in zip(row.indices, row.data):
            if g not in seen:
                scores[g] = scores.get(g, 0) + w * r

    if not scores:
        return None

    best = sorted(scores.items(), key=lambda x: x[1], reverse=True)[:top_n]
    return pd.DataFrame({
        'app_id':   game_enc.inverse_transform([g for g, _ in best]),
        'cf_score': [s for _, s in best]
    })


print("\n✓ CF готовий!")
#%% md
## 19. Stage 2 — Reranking класифікатором
```

```
#%%
MEDIAN_HOURS = df_rec['hours'].median()
print(f"Медіана годин із датасету: {MEDIAN_HOURS:.2f}")

# ----------------------------------------------------------------------------
# Словник усіх трьох навчених моделей (з секції 9)
# ----------------------------------------------------------------------------
RERANKERS = {
    'Logistic Regression': results['Logistic Regression']['model'],
    'Decision Tree':       results['Decision Tree']['model'],
    'Random Forest':       results['Random Forest']['model'],
}

print("\nДоступні ранкери:")
for name in RERANKERS:
    acc = results[name]['accuracy']
    f1  = results[name]['f1_score']
    print(f"  • {name:<22}  accuracy={acc:.4f}  f1={f1:.4f}")


def _build_features_for_candidates(candidates: "pd.DataFrame",
                                   target_user_id) -> "pd.DataFrame":
    """
    Внутрішня функція: добудовує фічі для DataFrame кандидатів.
    Повертає DataFrame із доданими колонками фічей (ті ж 7, що і в секції 6).
    """
    user_hours = user_avg_hours.get(target_user_id, MEDIAN_HOURS)

    df = candidates.merge(game_stats, on='app_id', how='left').fillna(0)

    # Фічі, ідентичні тренувальному pipeline (секція 6)
    df['helpful']         = df['avg_helpful']
    df['funny']           = df['avg_funny']
    df['hours']           = user_hours          # персональний час гравця
    df['hours_log']       = np.log1p(user_hours)
    df['helpful_ratio']   = df['avg_helpful'] / (df['avg_helpful'] + df['avg_funny'] + 1)
    df['engagement_score'] = df['avg_helpful'] + df['avg_funny']
    df['is_high_playtime'] = int(user_hours > MEDIAN_HOURS)

    return df
```

```
FEAT_COLS = ['helpful', 'funny', 'hours', 'hours_log',
        'helpful_ratio', 'engagement_score', 'is_high_playtime']


def rerank_with_model(candidates: "pd.DataFrame",
                target_user_id,
                model_name: str = 'Logistic Regression') -> "pd.DataFrame":
    """
    Stage 2: отримує кандидатів від CF, додає фічі, застосовує вибраний
    класифікатор і повертає відранжований DataFrame.

    Параметри
    ----------
    candidates    : DataFrame [app_id, cf_score] від get_cf_candidates()
    target_user_id: id гравця
    model_name    : один із 'Logistic Regression', 'Decision Tree', 'Random Forest'

    Повертає
    ----------
    DataFrame [app_id, cf_score, clf_proba, hybrid_score,
            review_count, positive_rate, avg_hours]
    відсортований за hybrid_score (спадно).

    Примітка: clf_proba відображає «якість гри» (game quality score)
    на основі агрегованих фічей, а не персоналізований скор для конкретного
    юзера. Персоналізація повністю виконується на Stage 1 (CF-cosine).
    # TODO: для справжньої персоналізації Stage 2 потрібні user-specific
    #       фічі (жанрові вподобання, avg% лайків юзера тощо).
    """
    clf = RERANKERS[model_name]
    df  = _build_features_for_candidates(candidates, target_user_id)

    X_cand      = scaler.transform(df[FEAT_COLS].values)
    df['clf_proba'] = clf.predict_proba(X_cand)[:, 1]

    # Нормалізація CF-скору в [0, 1]
    cf_min, cf_max = df['cf_score'].min(), df['cf_score'].max()
    cf_norm = (df['cf_score'] - cf_min) / (cf_max - cf_min + 1e-8)

    # Нормалізація кількості відгуків (популярність)
    rc_min, rc_max = df['review_count'].min(), df['review_count'].max()
```

```
    rc_norm = (df['review_count'] - rc_min) / (rc_max - rc_min + 1e-8)

    # Гібридний скор: 60% CF + 40% класифікатор - 10% штраф за популярність
    # Штраф -0.1*rc_norm: щоб уникнути домінування найпопулярніших ігор.
    # Ваги підібрані евристично; оптимально — налаштовувати через grid search.
    df['hybrid_score'] = (
        0.6 * cf_norm
        + 0.4 * df['clf_proba']
        - 0.1 * rc_norm
    )

    out_cols = ['app_id', 'cf_score', 'clf_proba', 'hybrid_score',
                'review_count', 'positive_rate', 'avg_hours']
    return df[out_cols].sort_values('hybrid_score', ascending=False).reset_index(drop=True)


print("\n✓ Функція rerank_with_model готова!")
print(f"  Підтримує: {list(RERANKERS.keys())}")
#%% md
## 20. Demo: реальний юзер — його історія та рекомендації

# Беремо юзерів з ТЕСТОВОЇ вибірки:
# В секції 8 сплітили рядки X/y, user_id вже не було в X.
# Тому беремо активних юзерів з df_cf — вони реальні і ми можемо
# верифікувати рекомендації проти їхньої справжньої історії.

# Топ юзери за кількістю відгуків (найбагатша історія для демо)
#%%
top_users = (df_cf.groupby('user_id')
             .agg(total=('app_id','count'),
                  liked=('is_recommended','sum'))
             .assign(pos_rate=lambda x: x['liked'] / x['total'])
             .query('15 <= total <= 50 and pos_rate >= 0.5')
             .sort_values('total', ascending=False)
             .head(20))

print("Активні юзери для демо:")
print(f"{'user_id':>12}  {'відгуків':>9}  {'сподобалось':>12}  {'%позитив':>9}")
print("-" * 50)
for uid, row in top_users.iterrows():
    pct = row['liked'] / row['total'] * 100
    print(f"{uid:>12}  {int(row['total']):>9}  {int(row['liked']):>12}  {pct:>8.0f}%")
```

```
# --- Вибираємо юзера ---
# Змінюй тут або запускай input() вручну
DEMO_USER = 5617457  # ← замінити на будь-який з таблиці вище
TOP_N = 10

print(f"\n{'=' * 60}")
print(f"🎮 DEMO USER: {DEMO_USER}")
print(f"{'=' * 60}")

# --- Показуємо реальну історію юзера ---
user_history = (df_cf[df_cf['user_id'] == DEMO_USER]
                .merge(game_stats[['app_id', 'review_count', 'positive_rate']], on='app_id', how='left')
                .sort_values('is_recommended', ascending=False))

liked = user_history[user_history['is_recommended'] == True]
disliked = user_history[user_history['is_recommended'] == False]

print(f"\n📋 Реальна історія ({len(user_history)} ігор):")
print(f"  ✅ Порекомендував : {len(liked)}")
print(f"  ❌ Не порекомендував: {len(disliked)}")

print(f"\nПорекомендовані ігри (перші 8):")
for _, row in liked.head(8).iterrows():
    hrs = row['hours']
    print(f"  app_id={int(row['app_id']):<8}  "
          f"hours={hrs:>7.1f}  "
          f"pop.positive={row['positive_rate']:.0%}")

if len(disliked) > 0:
    print(f"\nНЕ порекомендовані ігри (перші 5):")
    for _, row in disliked.head(5).iterrows():
        print(f"  app_id={int(row['app_id']):<8}  "
              f"hours={row['hours']:>7.1f}  "
              f"pop.positive={row['positive_rate']:.0%}")

# --- Запускаємо повний пайплайн ---
print(f"\n{'=' * 60}")
print(f"🔄 Запускаємо двоетапний пайплайн...")
print(f"{'=' * 60}")
```

```
# Stage 1
candidates = get_cf_candidates(DEMO_USER, top_n=TOP_N * 3)
print(f"\n[Stage 1] CF знайшов {len(candidates)} кандидатів")

# Stage 2
ranked = rerank_with_classifier(candidates, DEMO_USER)
final = ranked.head(TOP_N)
print(f"[Stage 2] Класифікатор відранжував → топ-{TOP_N}:\n")

# --- Гарний вивід ---
print(f"{'#':>2}  {'app_id':>8}  {'CF':>6}  {'LR%':>5}  {'hybrid':>7}  "
      f"{'відгуків':>9}  {'позит.':>7}  {'avg hrs':>8}")
print("-" * 65)
for i, row in final.iterrows():
    cf_bar = "▓" * int(row['clf_proba'] * 10)
    print(f"{i + 1:>2}. {int(row['app_id']):>8}  "
          f"{row['cf_score']:>6.3f}  "
          f"{row['clf_proba']:>5.2f}  "
          f"{row['hybrid_score']:>7.4f}  "
          f"{int(row['review_count']):>9,}  "
          f"{row['positive_rate']:>7.0%}  "
          f"{row['avg_hours']:>8.1f}")

# Перевіряємо: чи немає в рекомендаціях ігор, що юзер вже бачив
overlap = set(final['app_id']) & set(user_history['app_id'])
print(f"\n✅ Перетин з вже зіграними іграми: {len(overlap)} "
      f"({'є проблема' if overlap else 'окей, нових 100%'})")

#%% md
## 22. Evaluation — Precision@K, Recall@K, Hit Rate@K
# --- Train/Test split по взаємодіям ---
# 80% — будуємо матрицю, 20% — перевіряємо рекомендації
#%%
# ============================================================================
# СЕКЦІЯ 22 (ФІНАЛЬНА v2) — SVD-only vs SVD + ML Rerankers
#
# Виправлено: конфлікт між game_enc від CF (секція 18) та SVD (doc 4).
# Тепер всі SVD-об'єкти зберігаються в окремих змінних svd_*.
# ============================================================================

# %% md
## 22. Evaluation — SVD-only vs SVD + ML Rerankers (Sampled LOO, 2 000 юзерів)
```

```
# %%
import gc

# ---------------------------------------------------------------------------
# Захоплюємо SVD-об'єкти ЯВНО — щоб уникнути конфлікту з CF game_enc
# Ця секція має запускатися одразу після секції з SVD (doc 4)
# ---------------------------------------------------------------------------
svd_game_enc    = game_enc          # LabelEncoder на eligible_games (≥200 відгуків)
svd_user_enc    = user_enc          # LabelEncoder на eligible_users
svd_U           = U                 # (n_users, N_COMP)
svd_V           = V                 # (n_games, N_COMP)
svd_pop_norm    = game_pop_norm     # (n_games,)
svd_mat_seen    = mat_seen          # CSR (n_users × n_games)
svd_all_games   = np.array(svd_game_enc.classes_)   # масив app_id
svd_test_dict   = test_dict         # {user_id: pos_app_id}

# Перевірка сумісності
assert svd_V.shape[0] == len(svd_game_enc.classes_), (
    f"V має {svd_V.shape[0]} рядків, але svd_game_enc — "
    f"{len(svd_game_enc.classes_)} класів. "
    f"Перезапусти SVD-секцію перед цим кодом."
)
print(f"✓ SVD game_enc: {len(svd_game_enc.classes_):,} ігор  "
      f"(V shape: {svd_V.shape})")
print(f"✓ SVD user_enc: {len(svd_user_enc.classes_):,} юзерів")
print(f"✓ Test users  : {len(svd_test_dict):,}")

# ---------------------------------------------------------------------------
# Параметри
# ---------------------------------------------------------------------------
N_EVAL_USERS = 2_000
N_NEGATIVES  = 99
K_VALUES     = [5, 10, 20]
ALPHA_POP    = 0.05   # popularity bias у SVD-only (з doc 4)
ALPHA_SVD    = 0.80   # вага SVD у гібридному скорі
ALPHA_ML     = 0.20   # вага ML  у гібридному скорі
# Примітка: ваги 0.8/0.2 — евристика; SVD домінує бо він персоналізований,
# ML додає лише game-level якість. TODO: підібрати grid search на val-split.

rng_eval = np.random.RandomState(42)
```

```
# ---------------------------------------------------------------------------
# ML-моделі (з секцій 6-9)
# ---------------------------------------------------------------------------
RERANKERS = {
    'Logistic Regression': results['Logistic Regression']['model'],
    'Decision Tree':       results['Decision Tree']['model'],
    'Random Forest':       results['Random Forest']['model'],
}

FEAT_COLS = [
    'helpful', 'funny', 'hours', 'hours_log',
    'helpful_ratio', 'engagement_score', 'is_high_playtime'
]

print(f"\nML-ранкери:")
for name in RERANKERS:
    acc = results[name]['accuracy']
    f1  = results[name]['f1_score']
    print(f"  • {name:<22}  accuracy={acc:.4f}  f1={f1:.4f}")

print(f"\nПараметри оцінки:")
print(f"  Юзерів    : {N_EVAL_USERS:,}")
print(f"  Негативів : {N_NEGATIVES}")
print(f"  Ваги      : SVD={ALPHA_SVD}, ML={ALPHA_ML}, pop={ALPHA_POP}")



# ---------------------------------------------------------------------------
# Допоміжна: будує ML-фічі для набору ігор + конкретного юзера
# ---------------------------------------------------------------------------
def _build_ml_features(eval_app_ids: list, target_user_id) -> np.ndarray:
    """
    Повертає np.array shape (len(eval_app_ids), 7).
    Фічі ідентичні тренувальному pipeline (секція 6).
    """
    user_hours = float(user_avg_hours.get(target_user_id, MEDIAN_HOURS))

    df = pd.DataFrame({'app_id': eval_app_ids})
    df = df.merge(game_stats, on='app_id', how='left').fillna(0)

    df['helpful']         = df['avg_helpful']
    df['funny']           = df['avg_funny']
    df['hours']           = user_hours
```

```
    df['hours_log']       = np.log1p(user_hours)
    df['helpful_ratio']   = df['avg_helpful'] / (df['avg_helpful'] + df['avg_funny'] + 1)
    df['engagement_score'] = df['avg_helpful'] + df['avg_funny']
    df['is_high_playtime'] = int(user_hours > MEDIAN_HOURS)

    return df[FEAT_COLS].values


# ============================================================================
# Головна функція оцінки — єдиний Sampled LOO протокол для всіх режимів
# ============================================================================
def evaluate_mode(mode: str, k: int, n_eval: int = N_EVAL_USERS) -> dict:
    """
    Sampled LOO: 1 pos + 99 neg → ранг pos_game.
    Усі режими — однаковий eval set, різниця лише у формулі скору.

    mode:
      'SVD-only'              — score = SVD + alpha_pop * pop
      'SVD + Logistic Regression'
      'SVD + Decision Tree'
      'SVD + Random Forest'
    """
    hits = 0; ndcg_sum = 0.0; total = 0
    model_name = mode.replace('SVD + ', '') if mode != 'SVD-only' else None
    clf        = RERANKERS[model_name] if model_name else None

    # Семплуємо юзерів — тільки ті що є в SVD-матриці
    valid_users = [u for u in svd_test_dict if u in svd_user_enc.classes_]
    sample      = rng_eval.choice(valid_users, min(n_eval, len(valid_users)), replace=False)

    for uid in sample:
        pos_game = svd_test_dict[uid]

        # Перевіряємо що pos_game є в SVD
        if pos_game not in svd_game_enc.classes_:
            continue

        u_idx = int(svd_user_enc.transform([uid])[0])

        # Seen-ігри юзера (з SVD mat_seen)
        seen_game_idx = svd_mat_seen[u_idx].indices          # індекси в SVD
        seen_app_ids  = set(svd_game_enc.inverse_transform(seen_game_idx))
```

```
# 99 негативів із SVD-ігор
candidates = np.setdiff1d(svd_all_games, list(seen_app_ids))
if len(candidates) < N_NEGATIVES:
    continue
neg_games  = rng_eval.choice(candidates, N_NEGATIVES, replace=False)

# Eval set: 99 neg + 1 pos (всі — SVD app_ids)
eval_app_ids = list(neg_games) + [pos_game]

# Фільтруємо на випадок якщо щось вилізло за межі (додаткова захист)
eval_valid = [g for g in eval_app_ids if g in svd_game_enc.classes_]
if len(eval_valid) < 2 or pos_game not in eval_valid:
    continue

# SVD-індекси (гарантовано 0..n_svd_games-1)
eval_idx = svd_game_enc.transform(eval_valid)

# SVD scores
svd_scores = (svd_U[u_idx] @ svd_V[eval_idx].T
              + ALPHA_POP * svd_pop_norm[eval_idx])

if mode == 'SVD-only':
    final_scores = svd_scores

else:
    # ML reranking
    X_cand   = scaler.transform(_build_ml_features(eval_valid, uid))
    clf_prob = clf.predict_proba(X_cand)[:, 1]

    s_min, s_max = svd_scores.min(), svd_scores.max()
    svd_norm     = (svd_scores - s_min) / (s_max - s_min + 1e-8)

    final_scores = ALPHA_SVD * svd_norm + ALPHA_ML * clf_prob

# Ранг позитивної гри
pos_local = eval_valid.index(pos_game)
rank      = int(np.sum(final_scores > final_scores[pos_local])) + 1

hits     += int(rank <= k)
ndcg_sum += (np.log(2) / np.log(rank + 1)) if rank <= k else 0.0
total    += 1
```

```
    if total == 0:
        return {'Hit Rate': 0, 'Precision': 0, 'Recall': 0, 'NDCG': 0, 'n': 0}

    hr = hits / total
    return {
        'Hit Rate':  hr,
        'Precision': hr / k,
        'Recall':    hr,
        'NDCG':      ndcg_sum / total,
        'n':         total,
    }


# ============================================================================
# Запуск: 4 режими × 3 значення K
# ============================================================================
MODES = [
    'SVD-only',
    'SVD + Logistic Regression',
    'SVD + Decision Tree',
    'SVD + Random Forest',
]

print("\n" + "=" * 72)
print("Запускаємо Sampled LOO evaluation...")
print("=" * 72)

all_eval = {}

for idx, mode in enumerate(MODES, 1):
    all_eval[mode] = {}
    print(f"\n[{idx}/{len(MODES)}] {mode}")
    for k in K_VALUES:
        m = evaluate_mode(mode=mode, k=k)
        all_eval[mode][k] = m
        print(f"  K={k:<3}  HR={m['Hit Rate']:.4f}  "
              f"P={m['Precision']:.4f}  "
              f"R={m['Recall']:.4f}  "
              f"NDCG={m['NDCG']:.4f}  "
              f"(n={m['n']:,})")
    gc.collect()
```

```
# ============================================================================
# Фінальна таблиця
# ============================================================================
print("\n" + "=" * 76)
print("RECOMMENDATION SYSTEM — SAMPLED LOO EVALUATION")
print(f"Протокол: 1 pos + {N_NEGATIVES} neg | Юзерів: {N_EVAL_USERS:,}")
print("=" * 76)
print(f"{'Модель':<30} {'K':<5} {'Hit Rate':<11} {'Precision':<11} "
      f"{'Recall':<11} {'NDCG':<10}")
print("-" * 76)

for mode in MODES:
    for k in K_VALUES:
        m = all_eval[mode][k]
        print(f"{mode:<30} @{k:<4} "
              f"{m['Hit Rate']:<11.4f}"
              f"{m['Precision']:<11.4f}"
              f"{m['Recall']:<11.4f}"
              f"{m['NDCG']:<10.4f}")
    print()

print("=" * 76)

best_mode = max(MODES, key=lambda m: all_eval[m][10]['NDCG'])
print(f"\n🏆 Найкращий пайплайн @ K=10:")
print(f"   Режим    : {best_mode}")
print(f"   NDCG@10  : {all_eval[best_mode][10]['NDCG']:.4f}")
print(f"   Hit Rate : {all_eval[best_mode][10]['Hit Rate']:.4f}")
print("=" * 76)


# ============================================================================
# Візуалізація
# ============================================================================
colors_map = {
    'SVD-only':                '#85B7EB',
    'SVD + Logistic Regression':  '#1D9E75',
    'SVD + Decision Tree':       '#F4A442',
    'SVD + Random Forest':       '#E05C5C',
}
```

```
short_labels = {
   'SVD-only':                  'SVD',
   'SVD + Logistic Regression':  'SVD+LR',
   'SVD + Decision Tree':        'SVD+DT',
   'SVD + Random Forest':        'SVD+RF',
}

fig, axes = plt.subplots(1, 2, figsize=(16, 6))
ks      = K_VALUES
x       = np.arange(len(ks))
bar_w   = 0.18
n_modes = len(MODES)
offsets = np.linspace(-(n_modes-1)/2, (n_modes-1)/2, n_modes) * bar_w

for ax, metric in zip(axes, ['NDCG', 'Hit Rate']):
   for i, mode in enumerate(MODES):
      vals = [all_eval[mode][k][metric] for k in ks]
      bars = ax.bar(x + offsets[i], vals, bar_w,
                    label=short_labels[mode],
                    color=colors_map[mode],
                    edgecolor='white', linewidth=0.5)
      for bar, v in zip(bars, vals):
         ax.text(bar.get_x() + bar.get_width() / 2,
                 bar.get_height() + 0.004,
                 f'{v:.3f}', ha='center', fontsize=7.5, fontweight='bold')

   ax.set_title(f'{metric}@K', fontweight='bold', fontsize=13)
   ax.set_xticks(x)
   ax.set_xticklabels([f'K={k}' for k in ks])
   max_val = max(all_eval[m][k][metric] for m in MODES for k in ks)
   ax.set_ylim(0, min(1.0, max_val * 1.3 + 0.05))
   ax.legend(title='Пайплайн', fontsize=9)
   ax.grid(alpha=0.3, axis='y')
   ax.set_ylabel(metric)

plt.suptitle(
   f'SVD-only vs SVD + ML Rerankers\n'
   f'(Sampled LOO: 1 pos + {N_NEGATIVES} neg, {N_EVAL_USERS:,} юзерів)',
   fontsize=13, fontweight='bold'
)
plt.tight_layout()
plt.savefig('recommendation_metrics_svd_ml.png', dpi=150, bbox_inches='tight')
```

```
plt.show()

print("\n✓ Графік збережено: recommendation_metrics_svd_ml.png")
#%%
## 22 (FIXED BOOSTED). SVD + правильний LOO (sampled evaluation)
# Виправлено: баг у фільтрі + правильний протокол оцінки

from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import LabelEncoder

rng = np.random.RandomState(42)

# ── 1. Aggressive filtering ──────────────────────────────────────────────
MIN_USER_LIKES   = 20
MIN_GAME_REVIEWS = 200

liked_all = df_cf[df_cf['is_recommended'] == True].copy()

eligible_users = set(liked_all['user_id'].value_counts()[lambda x: x >= MIN_USER_LIKES].index)
eligible_games = set(df_cf['app_id'].value_counts()[lambda x: x >= MIN_GAME_REVIEWS].index)

liked_df = liked_all[
    liked_all['user_id'].isin(eligible_users) &
    liked_all['app_id'].isin(eligible_games)
].copy()

print(f"Після aggressive filtering:")
print(f"  Юзерів: {liked_df['user_id'].nunique():,}")
print(f"  Ігор  : {liked_df['app_id'].nunique():,}")

# ── 2. LOO split
# ──────────────────────────────────────────────────────────────────────────

test_dict = {}
train_rows_list = []

for uid, group in liked_df.groupby('user_id'):
    if len(group) < 2:
        continue
    idx = rng.randint(len(group))
    test_dict[uid] = group.iloc[idx]['app_id']   # 1 прихована гра
    mask = np.ones(len(group), dtype=bool)
    mask[idx] = False
```

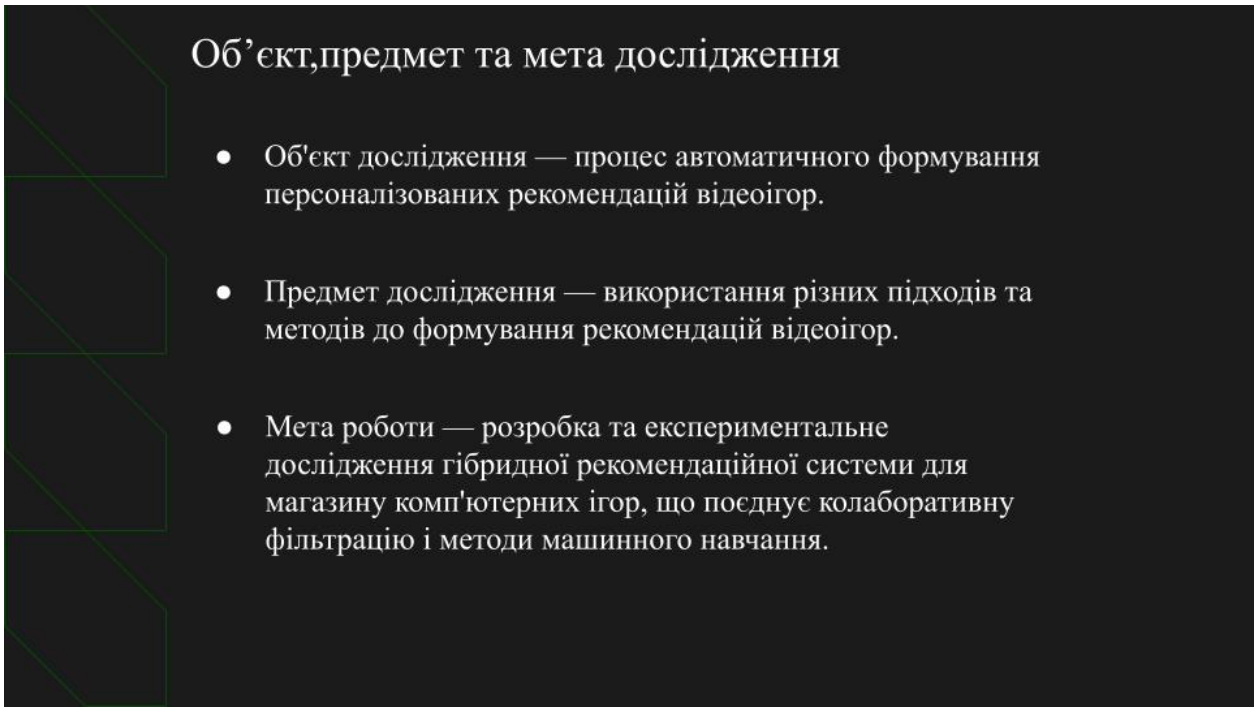
```
    train_rows_list.append(group.iloc[mask])

train_rows = pd.concat(train_rows_list)
print(f"\nLOO split: {len(train_rows):,} train | {len(test_dict):,} test users")

# ── 3. ТІЛЬКИ відфільтровані негативи (ВИПРАВЛЕНИЙ БАГ) ──────────────────
neg_df = df_cf[
    (df_cf['is_recommended'] == False) &       # ← скобки обов'язкові
    (df_cf['user_id'].isin(eligible_users)) &
    (df_cf['app_id'].isin(eligible_games))
].copy()

all_train = pd.concat([train_rows, neg_df])

# Warm users filter
train_counts = train_rows['user_id'].value_counts()
warm_users   = set(train_counts[train_counts >= 15].index)
test_dict    = {u: g for u, g in test_dict.items() if u in warm_users}
print(f"Warm users y test: {len(test_dict):,}")

# ── 4. Encode & sparse matrix ──────────────────────────────────────────────
user_enc = LabelEncoder().fit(all_train['user_id'])
game_enc = LabelEncoder().fit(all_train['app_id'])

all_train['u'] = user_enc.transform(all_train['user_id'])
all_train['g'] = game_enc.transform(all_train['app_id'])

nu = int(all_train['u'].max()) + 1
ng = int(all_train['g'].max()) + 1

pos = all_train[all_train['is_recommended'] == True].copy()
pos['w'] = np.log1p(pos['hours'].fillna(0)) + 1.0

mat_w = csr_matrix(
    (pos['w'].values, (pos['u'].values, pos['g'].values)),
    shape=(nu, ng)
)
mat_seen = csr_matrix(
    (np.ones(len(all_train)), (all_train['u'].values, all_train['g'].values)),
    shape=(nu, ng)
)
```

```
sparsity = 1 - mat_w.nnz / (nu * ng)
print(f"\nWeighted matrix: {mat_w.shape}, sparsity={sparsity:.4%}")

# ── 5. SVD ──────────────────────────────────────────────────────────────────

N_COMP = 128   # більше компонентів → краще
print(f"Fitting SVD (k={N_COMP})...")
svd = TruncatedSVD(n_components=N_COMP, random_state=42, n_iter=10)
U = svd.fit_transform(mat_w)
V = svd.components_.T

game_pop = np.asarray(mat_w.sum(axis=0)).ravel()
game_pop_norm = game_pop / (game_pop.max() + 1e-8)

print(f"Explained variance: {svd.explained_variance_ratio_.sum():.3%}")

# ── 6. ПРАВИЛЬНИЙ LOO: sampled evaluation ──────────────────────────────────
# Стандарт з NCF paper (He et al. 2017):
# Для кожного юзера ранжуємо 1 позитив + 99 випадкових негативів.
# Hit Rate@10 = "чи потрапив позитив у топ-10 з 100".
# Набагато легше ніж ранжувати серед 20k ігор!
N_NEGATIVES = 99

all_game_ids = np.array(game_enc.classes_)

def evaluate_sampled(k=10, n_eval=2000, alpha_pop=0.05):
    hits  = 0
    total = 0

    eval_users = list(test_dict.keys())
    sample = rng.choice(eval_users, min(n_eval, len(eval_users)), replace=False)

    for uid in sample:
        if uid not in user_enc.classes_:
            continue

        u_idx    = int(user_enc.transform([uid])[0])
        pos_game = test_dict[uid]

        if pos_game not in game_enc.classes_:
            continue
```

```
        # Сет вже зіграних ігор
        seen_games = set(game_enc.inverse_transform(mat_seen[u_idx].indices))

        # Семплуємо 99 негативів (ігри яких юзер не бачив)
        candidates = np.setdiff1d(all_game_ids, list(seen_games))
        if len(candidates) < N_NEGATIVES:
            continue
        neg_games = rng.choice(candidates, N_NEGATIVES, replace=False)

        # Набір для оцінки: 1 pos + 99 neg
        eval_games = np.concatenate([[pos_game], neg_games])

        # Фільтруємо тільки ті що є в game_enc
        eval_games_valid = [g for g in eval_games if g in game_enc.classes_]
        if len(eval_games_valid) < 2:
            continue

        eval_idx = game_enc.transform(eval_games_valid)

        # Скор SVD
        scores = U[u_idx] @ V[eval_idx].T + alpha_pop * game_pop_norm[eval_idx]

        # Позиція позитивної гри
        pos_idx_in_eval = eval_games_valid.index(pos_game)
        rank = int(np.sum(scores > scores[pos_idx_in_eval])) + 1  # 1-indexed

        hits  += int(rank <= k)
        total += 1

    return hits / total if total > 0 else 0.0, total


print("\n Sampled LOO evaluation (1 pos + 99 neg, стандарт NCF paper):")
print(f"{'K':<5} {'Hit Rate@K':<15} {'(n users)'}")
print("-"*35)
for k in [5, 10, 20]:
    hr, n = evaluate_sampled(k=k, n_eval=2000)
    print(f"K={k:<3} {hr:.4f}          ({n:,})")
```

ДОДАТОК Б


Розробка рекомендаційної системи для магазину комп’ютерних ігор
Виконав:студент групи КА-21,Олефіренко Мирослав
Дипломний керівник:доцент, к.ф.-м.н.,Яковлева Алла


Об’єкт,предмет та мета дослідження
Об'єкт дослідження — процес автоматичного формування персоналізованих рекомендацій відеоігор.
Предмет дослідження — використання різних підходів та методів до формування рекомендацій відеоігор.
Мета роботи — розробка та експериментальне дослідження гібридної рекомендаційної системи для магазину комп'ютерних ігор, що поєднує колаборативну фільтрацію і методи машинного навчання.

Постановка задачі

Для кожного користувача побудувати персональний список топ-N ігор, які він із найбільшою ймовірністю оцінить позитивно.
Тобто розв'язується задача персоналізованого ранжування (top-N recommendation).

Актуальність

Індустрія цифрової дистрибуції відеоігор стрімко зростає: Steam пропонує десятки тисяч ігор для сотень мільйонів користувачів.Індустрія цифрової дистрибуції відеоігор стрімко зростає: Steam пропонує десятки тисяч ігор для сотень мільйонів користувачів. За таких обсягів каталогу користувач стикається з інформаційним перевантаженням, що знижує задоволеність та конверсію платформи.

Рекомендаційні системи вирішують цю проблему через персоналізацію, підвищуючи залученість та продажі. Однак для ігрових платформ задача ускладнюється переважанням неявного зворотного зв'язку — факту придбання замість явних оцінок.

Розробка ефективних і масштабованих методів персоналізованих рекомендацій для ігрових платформ є актуальною науково-практичною задачею.

Вхідні дані

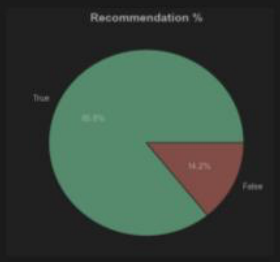
На Kaggle знайдено датасет із платформи Steam, що містить понад 41 мільйон рядків — кожен рядок представляє відгук користувача щодо конкретної гри.

Датасет складається з трьох таблиць: відомості про ігри (games.csv), профілі користувачів (users.csv) та рекомендації (recommendations.csv). Дані обрано через великий обсяг, якість попередньої обробки та можливість комплексного аналізу поведінки користувачів для побудови рекомендаційної системи.

Дослідження даних(EDA)

Для якісного формування набору ознак потрібно провести дослідження вхідних даних. Основну увагу потрібно приділити таблиці користувацьких рекомендацій

Основні інсайти:
Розподіл поля is_recommended(чи потрекомендував користувач гру) є істотно нерівномірним близько 85,8% відгуків є позитивними, що відповідає загальновідомій тенденції Steam до переважання позитивних оцінок.



Також рекомендаційні системи схильні до зміщення популярності (popularity bias): популярні об’єкти отримують більше взаємодій, що змушує систему рекомендувати їх ще частіше, утворюючи зворотний зв’язок.

У нашому випадку 10% ігор генерують 92% взаємодії,щоб з цим боротись **будемо вводити поправку на популярність**.

Дослідження даних(EDA)

Так як повний датасет — понад 41млн. відгуків обробка й навчання на всьому обсязі даних потребує великих обчислювальних ресурсів.Тому була проведена фільтрація малоактивних користувачів/ігор.
Початково завантажено: 41,154,794 рядків

Після фільтрації (≥5 відгуків на юзера, ≥30 на гру):
Записів : 22,270,115
Юзерів : 1,910,822
Ігор : 21,081

Також була побудована матриця взаємодії юзер х гра:
User-Item матриця: 1,910,822 юзерів × 21,081 ігор
Заповнених комірок : 22,270,095
Sparsity : 99.945%

Розрідженість матриці взаємодій > 99 % — переважна більшість пар «користувач × гра» не мають відгуку.

Вибір методу для побудови рекомендаційної системи

Вибір методу для побудови рекомендаційної системи

Content-based системи рекомендують ігри на основі їхніх характеристик (жанр, ціна тощо) та попередніх уподобань користувача. Підхід працює для нових об'єктів, але схильний до одноманітних рекомендацій. **Тому в роботі контентні ознаки використано лише як додаткові на етапі уточнення.**

Колаборативна фільтрація спирається на колективну поведінку — користувачі зі схожими смаками однаково оцінюють ігри. Це дозволяє виявляти приховані закономірності вподобань, **тому саме цей підхід став основою системи**.Усередині цієї родини, своєю чергою, розрізняють три підходи, що відрізняються способом обчислення рекомендацій.

Вибір методу для побудови рекомендаційної системи

Була обрана матрична факторизація (SVD). Замість прямого пошуку сусідів вони навчають латентні фактори: розкладають розріджену матрицю «користувач × гра» на дві компактні матриці меншої розмірності, де кожен користувач і кожна гра представлені вектором із k прихованих ознак. Такий підхід стійкий до розрідженості тому матричну факторизацію обрано як ядро системи — для відбору кандидатів.

Математичне формулювання усіченого SVD:

$$A_k = \sum_{f=1}^{k} \sigma_f \, u_f \, v_f^*$$

Де σі — сингулярні числа, а k — кількість найбільших ненульових сингулярних чисел. ui та vi — це і-ті стовпці матриць U та V відповідно(де U - матриця лівих сингулярних векторів, V - правих).

α_pop — коефіцієнт поправки,емпірично 0.05.

$\tilde{p}_i$ — нормована популярність гри і

Відповідно скор u-го користувача,і-ї гри:

$$\hat{r}(u,i) = (A_k)_{u,i} - \alpha_{\text{pop}}\,\tilde{p}_i = \sum_{f=1}^{k} \sigma_f \,(u_f)_u \,(v_f^*)_i - \alpha_{\text{pop}}\,\tilde{p}_i$$

Аналіз результатів

Тепер,коли отримали двовимірний масив оцінок рекомендаційної системи потрібно оцінити якість цих оцінок.

Якість рекомендаційної системи оцінюється за протоколом leave-one-out з використанням метрик ранжування:

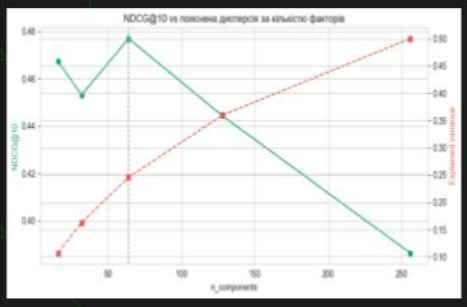
Протокол leave-one-out (LOO) — для кожного користувача одну позитивну взаємодію вилучають як цільову (тестову), а модель навчають на решті. Після цього система ранжує цей цільовий об'єкт серед 99 випадково відібраних негативних ігор (з якими користувач не взаємодіяв). Це визнаний стандарт оцінювання top-N рекомендацій.

HitRate@K — частка тестових випадків, у яких цільовий об'єкт потрапив до топ-K рекомендованого списку. Показує, наскільки часто система «вгадує» потрібну гру серед перших K позицій.

NDCG@K — враховує не лише факт влучання, а й позицію цільового об'єкта: що вище він у списку, то більше значення (нижчі позиції дисконтуються логарифмічно). Нормалізується до діапазону [0; 1], де 1 — об'єкт на першій позиції.

Аналіз результатів

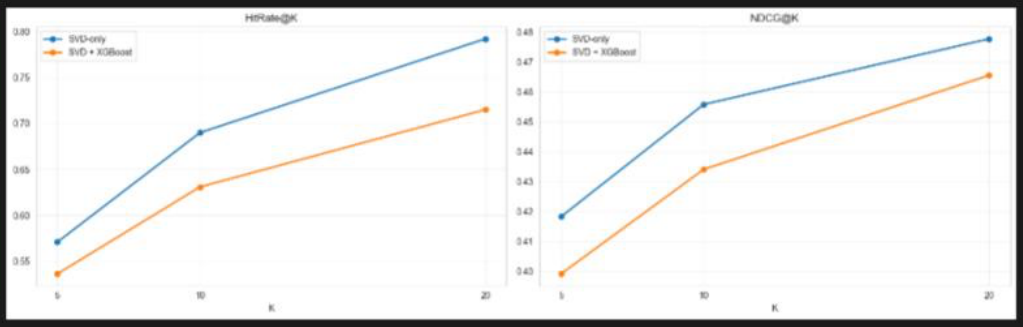
Кількість латентних факторів k обгрунтовано емпірично шляхом перебору значень з оцінюванням якості ранжування (NDCG@10) та поясненої дисперсії. За результатами перебору(з використанням LOO) обрано k = 64 як оптимальний компроміс між якістю та обчислювальними витратами.При значенні k = 64 HitRate@10 = 0.69,тобто для 69% юзерів гра,яку він порекомендує буде в першій десятці рекомендацій.



k (факторів)	Пояснена дисперсія	NDCG@10
16	10,7%	0,467
32	16,3%	0,453
64	24,6%	0,477
128	36,0%	0,444
256	50,0%	0,386

Аналіз результатів

Також була спроба покращити колаборативну фільтрацію реранжуванням за допомогою методів машинного навчання,але це не принесло покращення метрик якості рекомендаційної системи,так як методи машинного навчання прогнозували якість гри,що ,в середньому ,не покращувало рекомендації для юзерів.При дослідженні якості різних методів машинного навчання найкращі результати давав XGBoost,але навіть бустинговий метод не дав покращення рекомендацій.



Висновки

У дипломній роботі розроблено та досліджено гібридну рекомендаційну систему для магазину комп'ютерних ігор на основі методів колаборативної фільтрації та машинного навчання.

- Проведено огляд підходів до побудови рекомендаційних систем та обґрунтовано вибір колаборативного підходу.

- Реалізовано двоетапний пайплайн рекомендацій на основі усіченого SVD та повторного ранжування.

- За протоколом Sampled LOO встановлено, що чистий SVD забезпечує найкращу якість ранжування (HitRate@10 ≈ 0,69), тоді як наївне ML-реранжування не покращує рекомендацій через відсутність персоналізованих ознак.

- Практичне значення: створено відтворюваний програмний пайплайн, придатний для інтеграції в платформи цифрової дистрибуції відеоігор.

Подальші дослідження

- Текстові ознаки відгуків — використання тексту відгуків через TF-IDF та трансформерні ембедінги для глибшого моделювання вподобань.
- Навчання комбінації етапів (learning-to-rank) — замість фіксованого поєднання SVD + ML навчати оптимальну вагу реранжування (напр. LambdaMART), щоб другий етап покращував, а не погіршував ранжування.
- Зменшення зміщення популярності — методи debiasing і метрики різноманіття (diversity, novelty, coverage), щоб система не рекомендувала лише популярні ігри.
- Промислове розгортання — реалізація REST API, кешування артефактів і наближений пошук кандидатів (ANN, напр. FAISS) для масштабування; онлайн A/B-оцінювання.

ДЯКУЮ ЗА УВАГУ