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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістра

за спеціальністю 121 – Інженерія програмного забезпечення

На тему: Дослідження ефективності застосування навчання з підкріпленням для рекомендаційних систем

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-23-1м _____ /О. Д. Россієв/

Керівник кваліфікаційної роботи _____ / Н. Х. Саітгареєв /
Економіко-організаційна частина _____ / _____ /
Нормоконтроль _____ / Н. Х. Саітгареєв /
Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг
2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: магістр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А. М. Стрюк

«___» _____ 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-23-1м Россієву Олександру Дмитровичу

1. Тема: Дослідження ефективності застосування навчання з підкріпленням для рекомендаційних систем затверджено наказом по КНУ №277с від «15» квітня 2024.
2. Термін подання студентом закінченої роботи: «17» грудня 2024.
3. Вихідні дані по роботі: Отримані результати порівнняння моделей з навчанням з підкріпленням і класичних моделей.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): проаналізувати існуючі методи реалізації рекомендаційних систем, дослідити методологю навчання з підкріпленням для рекомендаційних систем, розробити алгоритми для навчання моделей, навчити моделі, зібрати основні метрики та проаналізувати їх.
5. Перелік ілюстративного матеріалу: функціональна діаграма, блок-схема розроблених алгоритмів та роботи програми.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літературних джерел за темою	22.03.2024 – 27.03.2024
2	Аналіз існуючих методів реалізації рекомендаційних систем	28.03.2024 – 31.03.2024
3	Аналіз методології навчання з підкріпленням	01.04.2024 – 10.04.2024
4	Аналіз моделей для реалізації рекомендаційних систем	11.04.2024 – 14.04.2024
5	Підготовка матеріалів першого розділу роботи	15.04.2024 – 20.04.2024
6	Розробка блок-схем та діаграм розроблюваного програмного забезпечення	21.04.2024 – 26.04.2024
7	Підготовка матеріалів другого розділу роботи	27.04.2024 – 01.05.2024
8	Тренування існуючих моделей рекомендаційних систем	02.05.2024 – 08.05.2024
9	Тренування моделі з навчанням з підкріпленням	09.05.2024 – 16.05.2024
10	Підготовка матеріалів третього розділу роботи	17.05.2024 – 20.05.2024
11	Оформлення пояснювальної записки	21.05.2024 – 29.05.2024

Дата видачі завдання: «15» квітня 2024 р.

Студент _____ / О. Д. Россієв /

Керівник роботи _____ / Н. Х. Саітгареєв

РЕФЕРАТ

РЕКОМЕНДАЦІЙНА СИСТЕМА, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, КОНТЕНТНА ФІЛЬТРАЦІЯ, КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, ДВОБАШКТОВА НЕЙРОНА МЕРЕЖА.

Пояснювальна записка: 67 с., 11 рис., 2 таб., 8 дод., 13 джерел.

Метою кваліфікаційної роботи є дослідження ефективності використання навчання з підкріпленням для рекомендаційних систем.

Було проаналізовано популярні методи для вирішення задачі рекомендації, було визначено їх недоліки та переваги. У ході виконання роботи було проведено аналіз сучасних підходів для використання навчання з підкріпленням для вирішення задачі рекомендації

У результаті роботи було розроблено та натреновано три популярні моделі для задачі рекомендаціїтакі такі як контентна фільтрація, колаборативна фільтрація та гібрідна модель для цього використано двобаштову нейронну меружу. Було також розроблено модель навчаня з підкріпленням. В кінці роботи було проведено оцінювання розробленних моделей.

Результати роботи демонструють підвищену ефективність моделі навчання з підкріпленням для розв’язання задачі рекомендації

ABSTRACT

RECOMMENDATION SYSTEM, REINFORCEMENT LEARNING, CONTENT-BASED FILTERING, COLLABORATIVE FILTERING, TWO-TOWER NEURAL NETWORK.

Thesis in: 67 p., 11 fig., 2 tab., 8 app., 9 references.

The purpose of this qualification project is to investigate the effectiveness of using reinforcement learning in recommendation systems. Popular methods for solving recommendation tasks were analyzed, and their advantages and disadvantages were identified. During the course of the work, modern approaches to applying reinforcement learning for recommendation problems were examined.

As a result, three popular models for recommendation tasks were developed and trained: a content-based filtering model, a collaborative filtering model, and a hybrid model based on a two-tower neural network. Additionally, a reinforcement learning model was developed. At the end of the project, all the developed models were evaluated.

The results of the work demonstrate the increased effectiveness of the reinforcement learning model in solving recommendation tasks.

ЗМІСТ

ВСТУП

Ми живемо в зеттабайтову еру. Величезний обсяг інформації, доступної в мережі, призводить до проблеми інформаційного перевантаження, що ускладнює прийняття правильних рішень.

У таких умовах рекомендаційні системи стали незамінним інструментом, який допомагає користувачам знаходити релевантний контент серед безмежного інформаційного простору. Проте традиційні методи рекомендацій стикаються з низкою обмежень, включаючи недостатню адаптивність та проблеми з масштабованістю.

Навчання з підкріпленням, як одна з галузей машинного навчання, пропонує нові можливості для подолання цих викликів. Завдяки здатності навчатися на основі взаємодії з середовищем, цей підхід може забезпечити більш персоналізовані та ефективні рекомендації.

Метою кваліфікаційної роботи є оцінка ефективності застосування навчання з підкріпленням у рекомендаційних системах, що може відкрити нові горизонти у вирішенні проблеми інформаційного перевантаження

Об’єктом кваліфікаційної роботи є рекоменндаційна система. Предметом є ефективність використання навчання з підкріпленням в рекомендацій системі.

Задачі кваліфікаційної роботи:

- аналіз джерел за темою роботи;
- аналіз роботи рекомендаційних систем;
- дослідження методів та інструментів реалізації таких систем;
- обґрунтування вибору моделей та програмних засобів для розробки програмного модуля кваліфікаційної роботи;
- розробка рекомендаційної системи традиційними методами;
- розробка рекомендаційної системи з використанням навчання з підкріпленням;
- порівняння ефективности різних методів рекомендації.

1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Критичний аналіз літературних джерел за темою

Рекомендаційні системи (РС) — це програмні інструменти та алгоритми, які були розроблені з метою допомогти користувачам знаходити цікаві для них елементи, прогнозуючи їхні вподобання або рейтинги щодо цих елементів.

Було запропоновано численні методи для вирішення проблеми рекомендацій, такі як:

- Метод спільної фільтрації(англійською collaborative filtering(CF));
- Метод фільтрації на основі вмісту(англійською content-based filtering(CBF));
- Гібридна фільтрація(англійською hybrid filtering(HF));
- Методи на основні правил.

Коротко схему методів рекомендації можно побачити на рисунку 1.1:

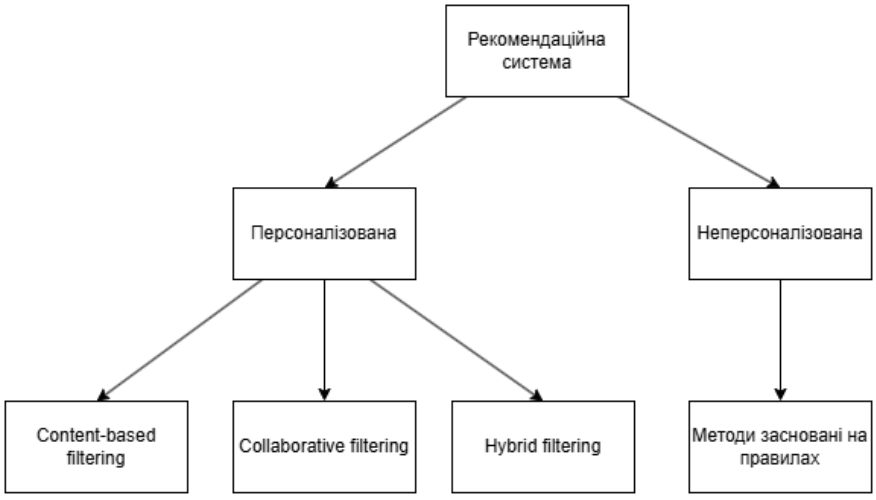

Рисунок 1.1 – Схема методів рекомендації

Метод спільної фільтрації[1] — це ефективний метод у рекомендаційних системах, який використовується для передбачення вподобань або інтересів користувача. Він базується на ідеї, що люди зі схожими смаками та уподобаннями в одній сфері, ймовірно, матимуть подібні вподобання в іншій.

Цей підхід намагається виявити користувачів з подібними смаками, аналізуючи їхні попередні взаємодії, і використовує їхню активність з елементами для прогнозування релевантності схожих елементів для конкретного користувача. Метою спільно-фільтрації є застосування думок інших людей для прогнозування вподобань та інтересів користувача. Це досягається шляхом знаходження користувачів зі схожими вподобаннями та використання їхніх оцінок елементів, щоб передбачити, як розглянутий користувач оцінить ті самі елементи.

Види спільно-фільтрації:

- Користувацька спільно-фільтрація(англійською User-based Collaborative Filtering);
- Предметна спільно-фільтрація(англійською Item-based Collaborative Filtering).

Користувацька спільно-фільтрація визначає групу користувачів, схожих на поточного користувача, на основі їхньої історії оцінок або взаємодій. Рекомендує елементи, які сподобалися цим схожим користувачам, але з якими поточний користувач ще не взаємодіяв.

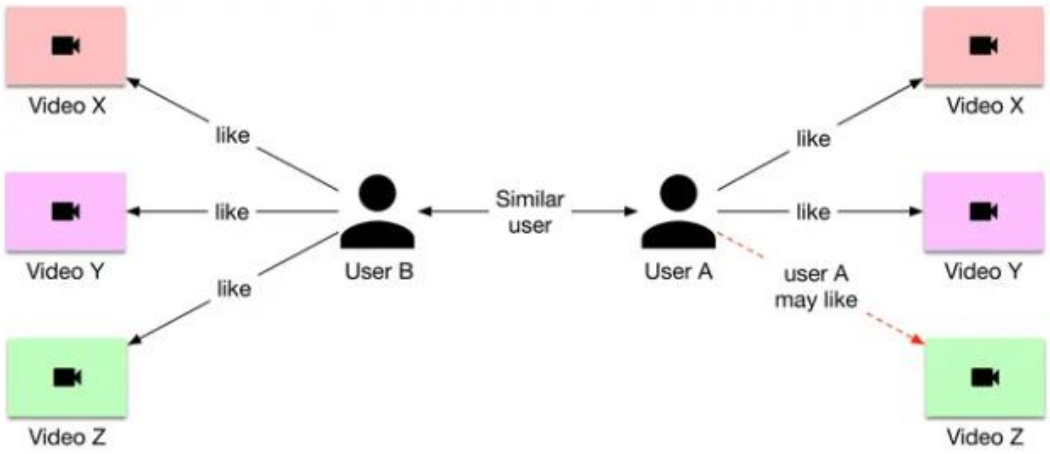

Рисунок 1.2 – Користувацька спільно-фільтрація

На рисунку 1.2 зображений принцип роботи користувацької спільно-фільтрації.

Перевагою цього метода є те, що він добре враховує колективні тенденції та може рекомендувати різноманітний контент.

Недоліком є те, що погано працює при великій кількості користувачів (проблеми масштабування) та страждає від проблеми холодного старту з новими користувачами.

Предметно спільна-фільтрація аналізує схожість між елементами на основі того, як користувачі їх оцінювали. Якщо два елементи мають схожі оцінки від багатьох користувачів, вони вважаються схожими.

Перевагою є те, що цей метод краще масштабується з великою кількістю користувачів, оскільки кількість елементів зазвичай менша.

Недоліком є те, що може не враховувати індивідуальні вподобання користувача настільки точно, як користувацька спільно-фільтрація.

Відповідно до дослідження [2], алгоритми для колаборативних рекомендацій можна поділити на дві загальні категорії: алгоритми на основі пам’яті (або евристичні) та алгоритми на основі моделей.

Алгоритми на основі пам’яті фактично є евристиками, які роблять прогнози оцінок, спираючись на повний набір уже оцінених користувачами елементів. Тобто, значення невідомої оцінки $r_{u,i}$ для користувача u та елемента i зазвичай обчислюється як агрегат оцінок деяких інших (зазвичай, N найбільш подібних) користувачів для цього ж елемента i

На відміну від методів, що базуються на пам’яті, модельно-орієнтовані алгоритми [3] використовують сукупність оцінок для навчання моделі, яка надалі слугує основою для передбачення оцінок.

Ба більше, у [3] було запропоновано метод колаборативної фільтрації в межах підходу машинного навчання. У цьому методі можна застосовувати різноманітні техніки машинного навчання (наприклад, штучні нейронні мережі) у поєднанні з методами вилучення ознак (зокрема, сингулярним розкладом, який є алгебраїчною технікою для зменшення розмірності матриць).

До інших підходів колаборативної фільтрації належать, зокрема, використання методів лінійної регресії[4], моделей максимальної ентропії[5]. байєсівських моделей [6], а також ймовірнісних реляційних моделей [7].

Загалом методи спільної фільтрації мають такі переваги:

- Не потрібні знання доменної області. Спільно-фільтрація не покладається на особливості елементів, це означає, що немає потреби у знаннях доменної області для створення характеристик з елементів;
- Легко виявити нові сфери інтересів користувачів. Система може рекомендувати відео на нові теми, з якими інші схожі користувачі взаємодіяли в минулому;
- Ефективність. Моделі на основі спільно-фільтрації зазвичай швидші та менш вимогливі до обчислювальних ресурсів, ніж контентна фільтрація, оскільки вони не покладаються на особливості елементів.

Недоліки методів спільної фільтрації:

- Проблема холодного старту. Це стосується ситуації, коли для нового відео або користувача доступно обмежену кількість даних, що означає, що система не може надати точні рекомендації. Спільно-фільтрація страждає від проблеми холодного старту через відсутність історичних даних взаємодії для нових користувачів або відео. Ця відсутність взаємодій заважає спільно-фільтрації знаходити схожих користувачів або відео;
- Не може обробляти нішеві інтереси. Спільно-фільтрації важко справлятися з користувачами зі спеціалізованими або нішевими інтересами. Спільно-фільтрація покладається на схожих користувачів для надання рекомендацій, і може бути складно знайти схожих користувачів з нішевими інтересами.

Контентна фільтрація[8] — це тип рекомендаційної системи, яка рекомендує користувачам елементи на основі їхніх минулих вподобань та поведінки. Вона працює шляхом аналізу вподобань користувача з точки зору таких атрибутів, як жанр, режисер, актор або навіть їхньої комбінації, а потім рекомендує інші елементи, які мають схожі характеристики.

Контентна фільтрація базується на припущенні, що користувачі, яким сподобався один елемент, ймовірно, сподобаються схожі елементи. Для створення рекомендацій система спочатку визначає атрибути елементів, з якими користувач взаємодіяв раніше. Потім вона знаходить інші елементи з подібними атрибутами та рекомендує їх користувачу.

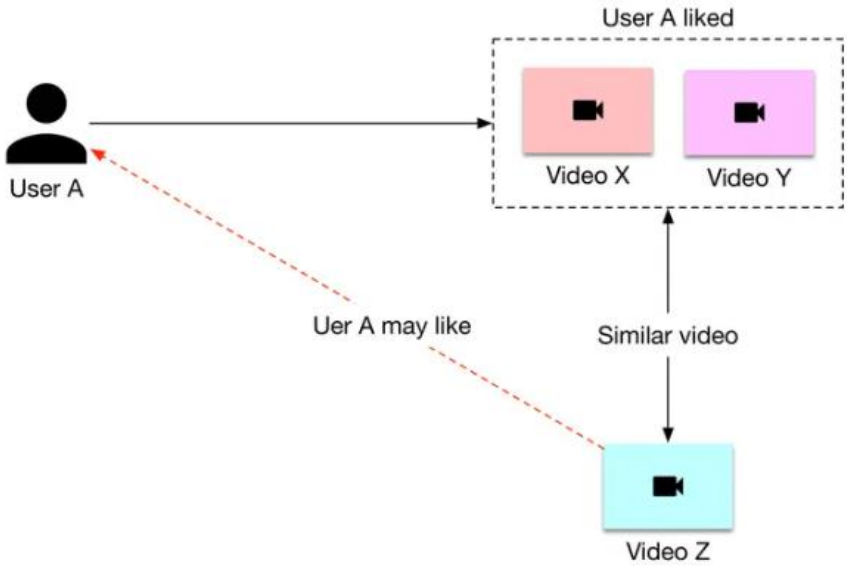

Рисунок 1.3 – Контентна фільтрація відеороликів

Пояснення до діаграмми зображеної на рисунку 1.3:

1. Користувач А взаємодіяв з відео X та Y у минулому;
2. Відео Z схоже на відео X та відео Y;
3. Система рекомендує відео Z користувачеві А

Контентна фільтрація має свої переваги та недоліки. Переваги:

- Можливість рекомендувати нові елементи. За допомогою цього методу нам не потрібно чекати даних про взаємодію

користувачів, щоб створити профілі для нових елементів. Профіль елемента повністю залежить від його характеристик;

- Здатність охопити унікальні інтереси користувачів. Це тому, що ми рекомендуємо елементи на основі попередніх взаємодій користувачів.

Недоліки:

- Складність виявлення нових інтересів користувача;
- Метод вимагає знань доменної області. Нам часто потрібно вручну створювати характеристики елементів.

Таблиця 1.1 – Порівняння контентної фільтрації та спільної фільтрації

	Контентна фільтрація	**Метод спільної фільтрація**
Обробляти нові відео	+	-
Відкриває нові сфери інтересів	-	+
Знання домену не потрібні	-	+
Ефективність	-	+

Як можга бачити з порівняння різних методів на таблиці 1.1 ці два методи є взаємодоповнювальними.

Гібридна фільтрація використовує як спільно-фільтрацію, так і контентну фільтрацію. Як показано на рисунку 1.4, гібридна фільтрація поєднує рекомендаційні системи на основі CF та контенту послідовно або паралельно. На практиці компанії зазвичай використовують послідовну гібридну фільтрацію [9].

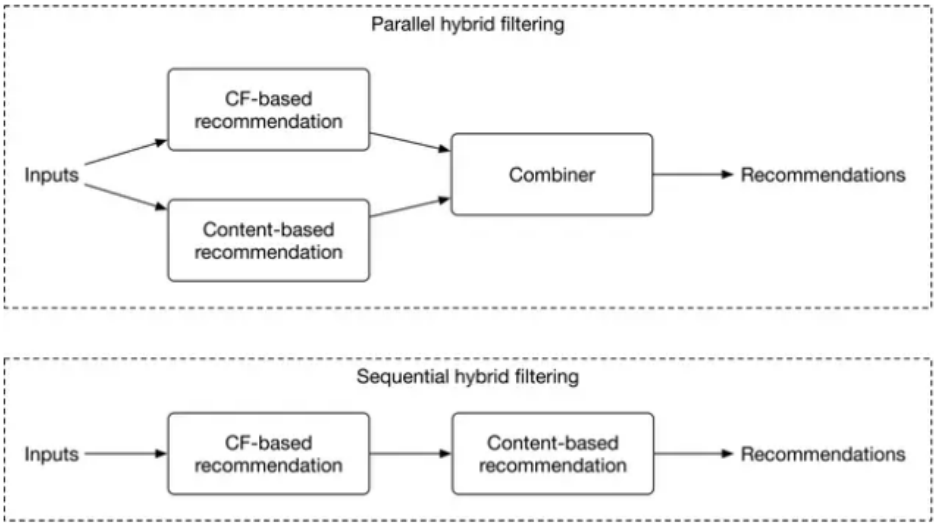

Рисунок 1.4 – Контентна фільтрація відеороликів

Цей підхід забезпечує кращі рекомендації, оскільки використовує два джерела даних: історичні взаємодії користувача та характеристики елементів. Характеристики елементів дозволяють системі рекомендувати релевантні елементи на основі тих, з якими користувач взаємодіяв раніше, а фільтрація на основі спільно-фільтрації допомагає користувачам відкривати нові сфери інтересів.

Багато компаній використовують гібридну фільтрацію, щоб отримати кращі рекомендації. Наприклад, у статті, опублікованій Google [9], описано, як YouTube використовує модель на основі контентної фільтрації як перший етап (генератор кандидатів), а потім модель на основі вмісту як другий етап, щоб рекомендувати відео, цей процес зображено на рисунку 1.5.

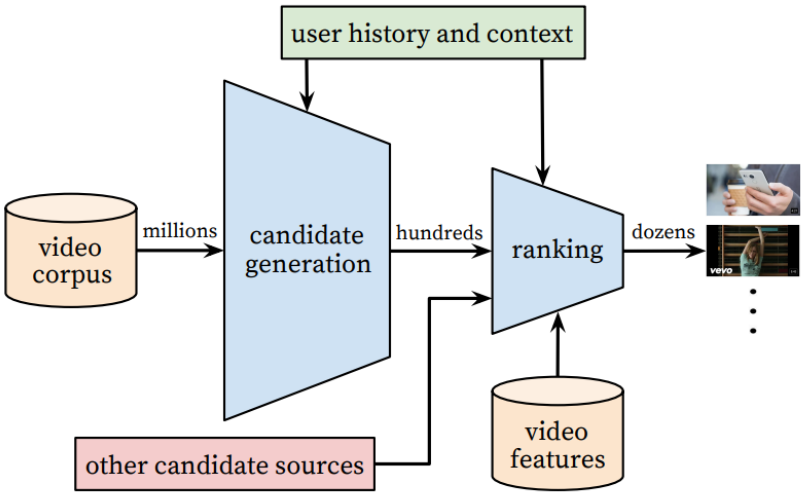

Рисунок 1.5 – Архітектура рекомендаційної системи YouTube

Формулювання проблеми рекомендацій як проблеми послідовного прийняття рішень може краще відображати взаємодію користувача та системи. Тому її можна сформулювати як марковський процес прийняття рішень (англійською markov decision process(MDP)) і вирішити за допомогою алгоритмів навчання з підкріпленням(англійською reinforcement learning(RL)).

Хоча ідея використання RL для рекомендацій не є новою та існує вже близько двох десятиліть, вона була не дуже практичною, головним чином через проблеми масштабованості традиційних алгоритмів RL. Однак нова тенденція з’явилася в цій галузі після впровадження глибокого навчання з підкріпленням(англійською deep reinforcement learning(DRL)), що дозволило застосувати RL до проблеми рекомендацій із великими просторами стану та дій.[10]

Навчання з підкріпленням є областю напівконтрольованого машинного навчання, в якій агент оптимізує свою поведінку через взаємодію з середовищем. Важливим досягненням у цій галузі стало поєднання глибокого навчання з традиційними методами RL, що отримало назву глибоке розширене навчання (англійською Deep Reinforcement Learning(DRL)). Це дозволило застосовувати RL до задач з величезними просторами станів і дій, включаючи самокеровані автомобілі, робототехніку, автоматизацію промисловості,

фінанси, охорону здоров'я та рекомендаційні системи. Унікальна здатність агента RL навчатися через винагороду від середовища без необхідності у навчальних даних робить RL ідеальним підходом для вирішення задач рекомендацій.

Навчальний елемент або приймач рішень називається агентом, а середовище — це все, що знаходиться за межами агента. Відповідно, на кроці часу t агент отримує певні уявлення або інформацію про середовище, які називаються станом (state), і на основі поточного стану виконує дію (action). Після виконання цієї дії він отримує числову винагороду (reward) від середовища і опиняється в новому стані. Цей принцип зображено на рисунку 1.5

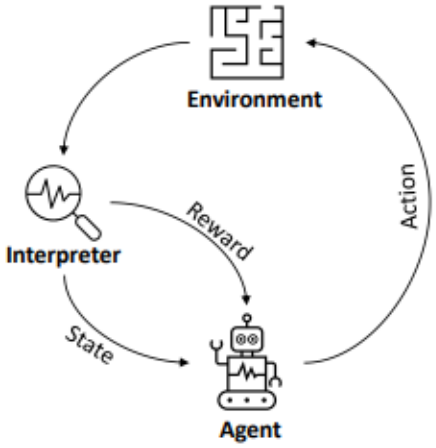

Рисунок 1.6 – Принцип роботи навчання з підкріпленням

Три унікальні особливості навчання з підкріпленням(RL) роблять його ідеальним підходом для вирішення задачі рекомендацій. По-перше, RL здатне враховувати динаміку послідовної взаємодії користувача з системою, коригуючи дії відповідно до безперервного зворотного зв’язку, отриманого від середовища. По-друге, RL враховує довгострокову залученість користувача в систему. Нарешті, хоча наявність оцінок користувачів є корисною, RL за своєю природою не потребує таких оцінок, оптимізуючи свою політику через послідовну взаємодію із середовищем. Усі ці причини свідчать про те, що використання RL може покращити рекомендації, що підтверджується результатами онлайн-досліджень.

Нижче наведено основні методи навчання з підкріпленням, які застосовуються для вирішення задач, де агент взаємодіє з довкіллям, отримуючи винагороду та намагаючись максимізувати її сумарне значення з часом.

Методи на основі ціннісних функцій(англійською Value-Based Methods) фокусуються на оцінюванні ціннісних функцій станів або станів-дій:

- Q-Learning - Один з найвідоміших алгоритмів. Намагається наблизити функцію Q(s, a), яка оцінює очікувану сумарну винагороду (discounted return), починаючи зі стану s, виконуючи дію a та далі дотримуючись оптимальної політики;
- SARSA – схожий на Q-Learning, але діє трохи інакше при оновленні. Використовує дію, яку агент фактично обирає наступною, а не максимізацію по діях.

Методи, що безпосередньо оптимізують політику(англійською Policy-Based Methods) замість оцінювання Q-функції, ці методи прямо параметризують політику $\pi_\theta(a|s)$ і оптимізують параметри θ, щоб максимізувати очікувану сумарну винагороду. До такого методу належить REINFORCE (Monte-Carlo Policy Gradient), він оцінює градієнт очікуваного повернення щодо параметрів політики. Цей метод оновлює політику у напрямку дій, які призвели до високих винагород.

Актор-критик методи(англійською Actor-Critic Methods) - Поєднують ідеї value-based та policy-based підходів. Актор відповідає за політику, тобто видає дії. Критик оцінює поточну політику за допомогою функції цінності (value function) або Q-функції, і дає сигнал для оновлення актора.

Прикладом є:

- A2C (Advantage Actor-Critic), A3C (Asynchronous Advantage Actor-Critic): Використовують функцію "advantage" для стабільнішого навчання. Актор оновлює політику у напрямку дій, які мають позитивну перевагу, а критик оновлює ціннісну функцію;

- PPO (Proximal Policy Optimization): Популярний метод актора-критика, який стабілізує навчання за рахунок обмеження оновлень політики (не дозволяє політиці змінюватися занадто різко між оновленнями).

1.2 Актуальність теми кваліфікаційної роботи

Рекомендаційні системи широко застосовуються в різних галузях, таких як електронна комерція, стримінгові сервіси та соціальні мережі, для покращення користувацького досвіду та підвищення залученості аудиторії.

В електронній комерції[11] рекомендаційні системи допомагають користувачам знаходити товари та послуги, що відповідають їхнім інтересам і потребам. Аналізуючи історію покупок, переглядів та пошукових запитів, системи можуть пропонувати персоналізовані рекомендації. Це не тільки підвищує ймовірність покупки, але й збільшує лояльність клієнтів. Для продавців це означає зростання продажів та можливість більш точного таргетування маркетингових зусиль.

У стримінгових сервісах, таких як Netflix чи Spotify, рекомендаційні системи є невід'ємною частиною платформи. Вони аналізують перегляди, прослуховування та взаємодії користувачів з контентом, щоб пропонувати нові фільми, серіали чи музичні треки, які можуть зацікавити. Це сприяє утриманню користувачів, підвищує їхню залученість та допомагає відкривати новий контент, що відповідає їхнім вподобанням.

У соціальних мережах рекомендаційні системи використовуються для персоналізації стрічки новин, рекомендацій друзів, груп та контенту. Вони аналізують поведінку користувача, його взаємодії та інтереси, щоб забезпечити максимально релевантний контент. Це підвищує час, проведений на платформі, стимулює взаємодію та дозволяє більш ефективно показувати таргетовану рекламу.

Рекомендаційні системи в сфері охорони здоров'я[12] використовуються для покращення якості обслуговування пацієнтів, оптимізації лікування та

підтримки медичних рішень. Лікарі використовують рекомендаційні системи для отримання підказок щодо діагностики та вибору оптимальних методів лікування на основі найновіших досліджень та статистичних даних. Використання рекомендаційних систем у медичній галузі сприяє більш ефективному та персоналізованому підходу до лікування, що підвищує загальний рівень охорони здоров'я.

Загальний вплив:

- Персоналізація досвіду: Рекомендаційні системи забезпечують індивідуальний підхід до кожного користувача, підвищуючи задоволеність та лояльність;
- Підвищення залученості: Персоналізований контент стимулює користувачів більше взаємодіяти з платформою чи сервісом;
- Зростання доходів: Для бізнесу це означає збільшення продажів, ефективніше використання маркетингових ресурсів та можливість монетизації через таргетовану рекламу;
- Оптимізація контенту: Аналізуючи дані користувачів, компанії можуть краще розуміти тренди та вподобання, що дозволяє їм оптимізувати свій контент чи асортимент товарів.

Розвиток технологій штучного інтелекту та машинного навчання збільшив роль рекомендаційних систем. Вони стали важливим інструментом для створення більш інтуїтивного та ефективного користувацького досвіду, що приносить користь як споживачам, так і бізнесу.

Наприклад стримінговий сервіс Netflix використовує просунуті рекомендаційні системи, які націлені на максимальну персоналізацію контенту для кожного користувача. Ця система пройшла довгий еволюційний шлях — від простих фільтраційних моделей до складних гібридних методів з використанням глибинного навчання, ембедингів та контекстних сигналів. Історично Netflix починав з матричної факторизації та подібних методів колаборативної фільтрації (наприклад, моделі, подібні до SVD), які навчаються на основі "хто що дивився та оцінив". Ці методи дозволяють

виявити латентні фактори, що описують інтереси користувачів і характеристики контенту.

Крім колаборативних сигналів (історії переглядів, рейтинги), Нетфлікс враховує метадані про контент (жанри, акторів, режисерів), а також більш складні фічі, отримані з текстового опису, титрів, обкладинок і навіть трейлерів. Це дозволяє не лише покладатися на схожість у поведінці користувачів, а й розуміти сам контент глибше.

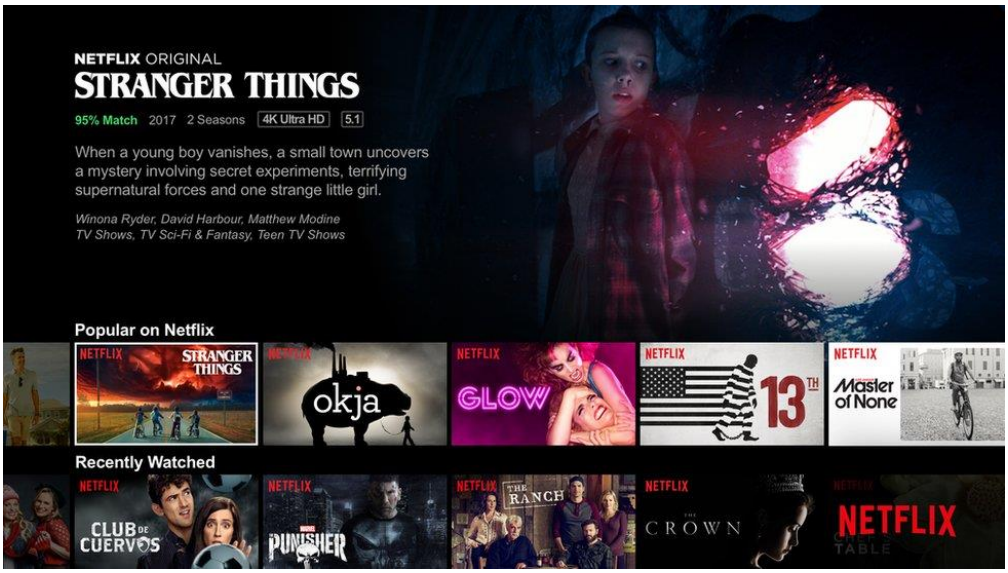

Рисунок 1.7 – Головна сторінка сервісу Netflix

Кожен користувач бачить персоналізовану головну сторінку, що зображена на рисунку 1.7 де ряди сформовано під конкретні вподобання. Netflix не лише обирає, який контент показати, а й вирішує, як його згрупувати, в якому порядку розмістити ряди та навіть яку обкладинку показати для максимального привернення уваги користувача.

Як і у випадку з YouTube, метрикою успіху є не просто клік, а загальний час перегляду, утримання клієнта, а також довгострокова задоволеність користувачів. Нетфлікс аналізує, які шоу користувачі насправді дивляться повністю, до яких повертаються, а які покидають після декількох хвилин. Цей зворотний зв’язок дозволяє системі покращувати точність рекомендацій.

Нетфлікс працює у багатьох країнах світу, тому враховує мовний та культурний контекст. Це означає, що рекомендації користувачу з Іспанії

можуть суттєво відрізнятися від рекомендацій користувачу з Японії, навіть якщо вони мають схожі жанрові вподобання.

Також прикладом рекомендаційної системи з іншої предметної області може слугувати Amazon. Рекомендаційна система Amazon — це високоефективний механізм, побудований переважно на Item-to-Item колаборативній фільтрації, підсиленій контентними сигналами, відгуками та історіями покупок. Її відмінність — проста, проте дуже масштабована й оптимізована структура, що дозволяє генерувати персоналізовані, точні та своєчасні рекомендації для мільйонів користувачів і товарів по всьому світу. На рисунку 1.8 зображено головна сторінка сервісу Amazon.

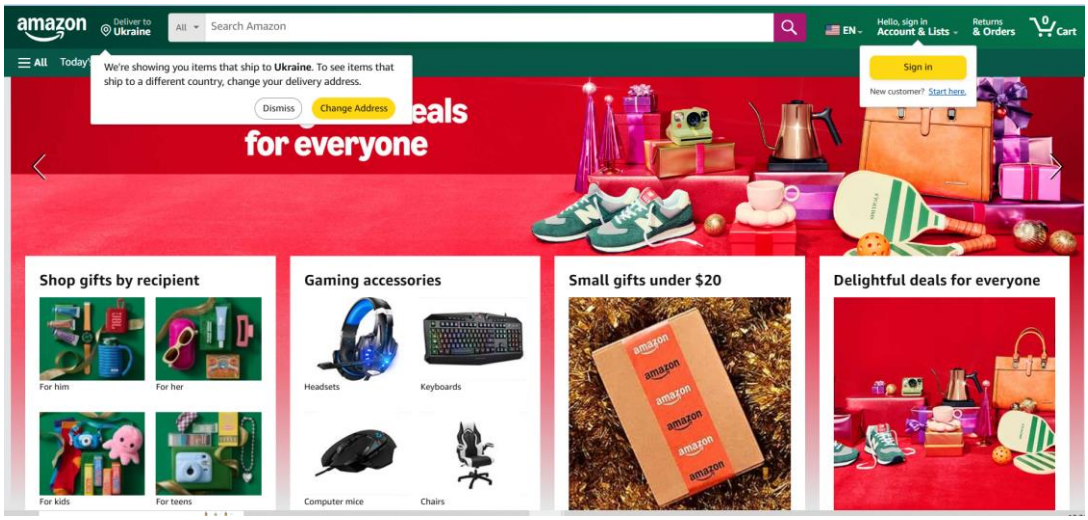

Рисунок 1.8 – Головна сторінка сервісу Amazon

1.3 Цілі та завдання кваліфікаційної роботи

Метою кваліфікаційної роботи є оцінка ефективності застосування навчання з підкріпленням у рекомендаційних системах, що може відкрити нові горизонти у вирішенні проблеми інформаційного перевантаження

Об’єктом кваліфікаційної роботи є рекоменндаційна система. Предметом є ефективність використання навчання з підкріпленням в рекомендацій системі.

Задачі кваліфікаційної роботи:

- аналіз джерел за темою роботи;

- аналіз роботи рекомендаційних систем;
- дослідження методів та інструментів реалізації таких систем;
- обґрунтування вибору моделей та програмних засобів для розробки програмного модуля кваліфікаційної роботи;
- розробка рекомендаційної системи традиційними методами;
- розробка рекомендаційної системи з використанням навчання з підкріпленням;
- порівняння ефективности різних методів рекомендації.

В результаті виконання кваліфікаційної роботи буде розроблено моделі рекомендаційних систем, класичні, та система з навчанням з прідкріпленням і будуть порівняні ключові метрики роботи цих систем.

2 РОЗРОБКА АЛГОРИТМІВ ТА СХЕМ

2.1 Розробка функціональної схем програмного комплексу

Основна ідея рекомендаційної системи реалізовної класичними методами контент фільтрації або колаборативної фільтрації полягає в тому, що на її вхід подається ідентифікатор користувача, модель на основі нього генерую список рекомендацийб, і на виході отрмиуємо його, цей процесс зображено функціональній схемі на рисунку 2.1.

Рисунок 2.1 – Функціональна схема класичної рекомендаційної системи

Рекомендаційна система з навчанням з підкріпленням працює схожим чином однак в процесі роботи враховуються відгуки користувачів, а саме це

може буди придбаний товар який було рекомендовано, або лайкнуте відео тощо. На рисунку 2.2 зображено функціональна схема першого рівня рекомендаційної системи з навчанням з підкріпленням:

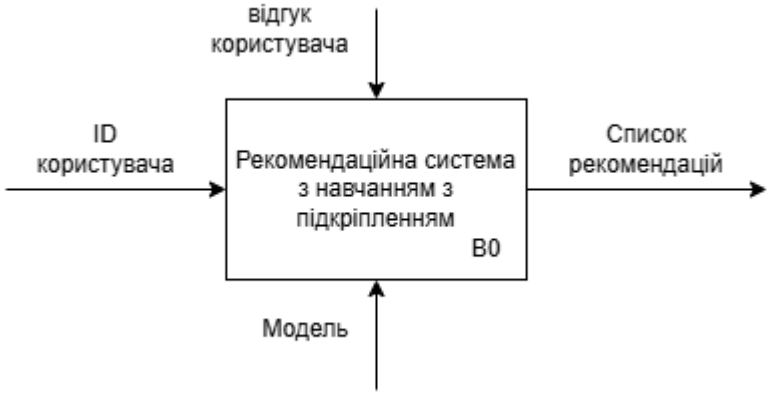

Рисунок 2.2 – Функціональна схема рекомендаційної системи з наввчанням з підкріпленням

2.2 Розробка алгоритму роботи програми

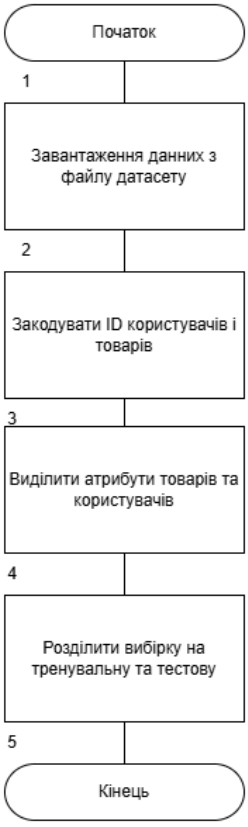

Рисунок 2.3 – блок схема завантаження і обробки датасету

На рисунку 2.3 зображена блок схема роботи алгоритму завантаження та попередньої обробки данних з датасету. Спочатку данні завантажуються в пам'ять програми, потім кодуються ідентифікатори користувачів та товарів, потім виділяються атрибути товарів і користувачів яку будуть використовуватися для навчання моделей і на фінальному етапу датасет буде розділено на навчальну та тестову вибірки.

Рисунок 2.4 – блок схема тренування Two Tower моделі

На рисунку 2.4 зображен алгоритм тренування Two Tower моделі. На першому єтапі ініціалізуються всі шари мережі та формується модель с заданими параметрами, потім в модель завантажуються дання для тренування, після цього модель тренується методом зворотнього поширення помилки, в кінці вона валідується на тестових данних.

Рисунок 2.5 – блок схема тренування моделі навчання з підкріаленням

На рсиунку 2.5 зображена блок схема тренування моделі навчання з підкріпленням. По перше модель ініціалізується з навчальними параметрами, після цього відпувається переднавчання моделі на історичних данних, потім

запускається цикл з навчанням моделі, протягом ітерації модель видає список рекомендацій, після цтого симулюється відповідь користувача і на основі неї модель оновлює ваги мережі.

Рисунок 2.6 – блок схема оцінки моделей

На рисунку 2.6 зображена блока схема оцінки моделей. Спочатку генеруються рекомендації обраної моделі, потім поірвнюються сгенеровані рекомендації з тестовими данними і в кінці розраховуються метрики.

3 РОЗРОБКА МОДЕЛІ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз обраного середовища програмування

Для написання програмної частини дослідження буде використано інтегроване середовище розробки PyCharm (англ. integrated development environment, IDE). Вибір цього інструменту зумовлений його розвиненою

функціональністю редактора, наявністю статичного аналізу коду, можливістю перевірки синтаксису, інтерактивною відладкою, широким асортиментом плагінів та спеціалізацією на розробку мовою Python. Окрім цього, PyCharm пропонує інтерактивні візуальні засоби для роботи з системою контролю версій Git та інструменти для взаємодії з Docker.

Серед небагатьох конкурентів PyCharm можна виокремити Visual Studio Code (VSC). Проте для досягнення подібного до PyCharm функціоналу у VSC доведеться підключити чимало додаткових розширень.

3.2 Аналіз обраних технологій та фреймворків

Для розробки програмного забезпечення було обрано мову Python, адже вона користується великою популярністю у сфері машинного навчання та пропонує різноманітні інструменти й бібліотеки для ефективної роботи з даними. Завдяки широкому набору спеціалізованих пакунків, як-от TensorFlow, PyTorch та scikit-learn, розробники можуть швидко створювати, навчати й оптимізувати складні моделі, а також легко інтегрувати алгоритми машинного навчання у свій програмний продукт. Це, у свою чергу, сприяє підвищенню точності прогнозів, прискоренню обробки інформації та створенню більш гнучких і масштабованих рішень.

У ході проведення даного дослідження була використана бібліотека Keras, адже вона надала змогу оперативно будувати високорівневі нейронні мережі та зосередитися на логіці експерименту, мінімізуючи затрати часу на низькорівневу реалізацію. Для числових розрахунків і маніпулювання багатовимірними масивами було застосовано NumPy, що забезпечило ефективне виконання складних математичних обчислень. Задля підготовки, очищення та систематизації даних у зручному форматі знайшла використання бібліотека Pandas, яка істотно спростила попередній аналіз та створила чітку структуру для подальших етапів дослідження. При визначенні найбільш адекватних методів машинного навчання, а також під час їхнього налаштування й оцінювання, була застосована Scikit-learn, завдяки якій дослідницька робота стала більш гнучкою та варіативною. Для ефективного

управління велекими датасетами було використано бібліотеку Huggingface dateset, яка дозволяє маніпулювати датасетом в дисковому просторі не завантажуючи датасет в оперативну память. Додатково, під час досліджень підходів навчання з підкріпленням була застосована OpenAI Gym, яка розширила можливості моделювання динамічних сценаріїв та сприяла більш ефективному вдосконаленню поведінки агентів. Таким чином, використання наведених інструментів дозволило суттєво підвищити якість, гнучкість та масштабованість проведеного дослідження.

3.3 Архітектури класичних моделей рекомендаційної системи

Для реалізації класичних підходів розв’язання задачі рекомендаціїї, таких як контента фільтрація, колаборативна фільтрація, та гібридного підходу, була обрана модель двошарової нейронної мережі(англійською two tower network). Ця модель дозволяє об’єднати данні різних модальностей в один простір для вбудовування(англійською embedding space). Це досягається шляхом використання двох нейронних мереж, так званих башен, одна мережа дозволяє побудувати простір для данних юзера, інша для данних обєктів рекомендації.

Створення спільного простору вбудовувань дозволяє перейти від простого порівняння окремих ознак або категорій до аналізу схожості високорівневих, абстрактних представлень даних. Завдяки такому підходу рекомендаторна система одержує здатність виявляти складні, нерідко приховані зв’язки між користувачем та контентом. Це дає змогу знаходити спільні риси різнопланових елементів, покращує узагальнювальні можливості моделі та зменшує залежність від явних міток чи розріджених даних. Крім того, такий простір значно спрощує масштабування системи, адже розширення набору користувачів і контенту не потребує докорінної перебудови, а впровадження нових атрибутів та інформаційних джерел стає більш гнучким та ефективним.

Так для того, щоб реалізувати метод контентної фільтрації потрібно використати в якості данних користувача його історичні данні взаємодіїї з обєктами рекомендації, а для данних обєктів рекомендації потрібно використати їх ознаки.

Структура мережі:

- Вхід для користувача: вектор $u \in \mathbb{R}^{d_u}$;
- Вхід для товару: вектор $v \in \mathbb{R}^{d_v}$.

Де $d_u -$ розмірність ознак користувача, $d_v -$ розмірність ознак товару

Обидва вектори пропускаються через окремі мережі (вежі) — через один повнозв'язний шар з активаціями ReLU. Тоді $f_u: \mathbb{R}^{d_u} \to \mathbb{R}^k$ та $f_v: \mathbb{R}^{d_v} \to \mathbb{R}^k$ — відображення, які витягують k-вимірні вектори, що представляють користувача й товар у спільному латентному просторі.

Формули отримання вбудовування ознак користувача і товара:

$$u' = f_u(u) = ReLU(W_u u + b_u), u' \in \mathbb{R}^k \quad (3.1)$$

$$v' = f_v(v) = ReLU(W_v v + b_v), v' \in \mathbb{R}^k \quad (3.2)$$

де $W_u -$ ваги мережі користувача, $W_v -$ ваги мережі товара, $b_u -$ вектор відхилення користувача, $b_v -$ вектор відхилення товару.

Формула скалярного добутку латентних векторів:

$$\hat{y} = \sigma(w_o(u' \cdot v') + b_o) \quad (3.3)$$

де $\cdot$ — скалярний добуток, σ — сигмоїдна функція, w_o і b_o— параметри вихідного шару.

Колаборативна двовежова модель покладається лише на ідентифікатори користувачів та товарів. Вона навчає ембединги без додаткових контентних фіч. Кожному користувачеві та товару відповідає вбудований латентний вектор.

Структура мережі:

- Вхід для користувача: одне число i_u(індекс користувача);
- Вхід для товару: одне число i_v (індекс товару).

Використовуємо вбудовувальні шари для відображення індексу в k-вимірний ембединг:

Формула шару ембединга користувача і товару:

$$u' = E_u(i_u) \in \mathbb{R}^k \quad (3.4)$$

$$v' = E_v(i_v) \in \mathbb{R}^k \quad (3.5)$$

Де E_u та E_v — матриці ембедингів для користувачів та товарів відповідно.

Формула вихідного шару:

$$\hat{y} = \sigma(w_o(u' \cdot v') + b_o) \quad (3.6)$$

де $\cdot$ — скалярний добуток, σ — сигмоїдна функція, w_o і b_o— параметри вихідного шару.

Гібридна двовежова модель поєднує підходи. Використовуються ідентифікатори користувачів та товарів (для колаборативних сигналів) разом з контентними фічами.

Структура мережі:

- Вхід для користувача: індекс i_u та контентний вектор користувача $u \in \mathbb{R}^{d_u}$;
- Вхід для товару: індекс i_v та контентний вектор товару $v \in \mathbb{R}^{d_v}$.

Спочатку витягуємо ембединг за індексом:

Формули отримання вбудовування ознак користувача і товара:

$$u' = E_u(i_u) \in \mathbb{R}^k \quad (3.7)$$

$$v' = E_v(i_v) \in \mathbb{R}^k \quad (3.8)$$

Де E_u та E_v — матриці ембедингів для користувачів та товарів відповідно.

Потім конкатенуємо ембединг користувача з його фічами:

Формула конкатенації еибединга користувача з його фічами:

$$u'' = ReLU(W_u[u', u] + b_u), u' \in \mathbb{R}^k \quad (3.9)$$

$$v'' = ReLU(W_v[v', v] + b_v), v' \in \mathbb{R}^k \quad (3.10)$$

Де $[u', u]$ означає конкатенацію векторів u' та u по ознаковій осі, аналогічно v', v .

Формула вихідного шару:

$$\hat{y} = \sigma(w_o(u'' \cdot v'') + b_o) \quad (3.11)$$

де $\cdot$ — скалярний добуток, σ — сигмоїдна функція, w_o і b_o— параметри вихідного шару.

3.4 Архітектура моделі навчання з пікріпленням

Модель навчання з підкріпленням розглядає рекомендаційну задачу як контекстну бандитну проблему, де на кожному кроці (раунді) система отримує стан користувача (контекст) і повинна вибрати дію (рекомендований товар), після чого отримати негайну нагороду. Задача моделі — вивчити політику вибору дій, яка максимізує середню сумарну винагороду.

На відміну від звичайних статичних моделей, які просто оцінюють, наскільки користувач любить певний товар, ця модель поступово оновлює свої уявлення, адаптуючись до зворотного зв’язку від користувача, тобто винагороди.

Компоненти:

- Стан: вектор ознак користувача, позначимо його як s_t ;
- Дії: набір товарів, які можна рекомендувати. На кроці t модель вибирає дію a_t, що відповідає певному товару.
- Винагорода r_t: відгук користувача на рекомендацію. Наприклад, 1, якщо користувач клікнув або придбав товар, 0 інакше.

Модель використовує наближення функції Q-значень (англійською Q-learning), щоб оцінити корисність дії в даному стані.

Формула Q-функції:

$$Q_\theta(s,a) \approx \mathrm{E}[r_t + \gamma \times maxQ_\theta(s_{t+1},a')|s,a] \quad (3.12)$$

Де, $\theta -$ ваги нейронної мережі, $r_t -$ винагорода на кроці t, $\gamma -$ коефіцієнт дисконтування, який зменшує вплив майбутніх нагород, s_{t+1} наступний стан, $a' -$ можлива дія.

Для наближення Q-функції використана нейронна мережа. Вхід — стан і дія тобто ознаки користувача і товари які можна рекомендувати, вихід — оцінка Q-значення. Після отримання винагороди й спостереження нового стануs_{t+1}, ми оновлюємо θ мінімізуючи помилку Беллмана:

Формула помилки Беллмана:

$$L(\theta) = \left(r_t + \gamma \times maxQ_\theta(s_{t+1},a') - Q_\theta(s_t,a_t)\right)^2 \quad (3.13)$$

Де, $\theta -$ ваги нейронної мережі, $r_t -$ винагорода на кроці t, $\gamma -$ коефіцієнт дисконтування, який зменшує вплив майбутніх нагород, s_{t+1} наступний стан, $a' -$ можлива дія, $s_t -$ стан в момент часу t, $a_t -$ обрана дія в момент часу t.

Мінімізуючи цю похибку, мережа вчиться краще оцінювати значення дій у різних станах.

В якості політики π_θ була обрана $\epsilon -$ жадібна політика яка з імовірністю ϵ обирає випадкову дію, а з імовірністю $1 - \epsilon$ дію з максимальним $Q -$ значенням:

Формула $\epsilon -$ жадібної політики:

$$\pi_\theta(s_t) = \begin{cases} \text{випадкова дія, з імовірністю } \epsilon \\ \arg max_a\, Q_\theta(s_t,a), \text{з імовірністю } 1-\epsilon \end{cases} \quad (3.14)$$

Де, $\theta -$ ваги нейронної мережі, $a -$ дія, $s_t -$ стан в момент часу t

3.5 Програмна реалізація основних функцій

В якості датасету для навчання моделей було обрано Amazon Review dataset[13], в якості категорії товарів було обрані косметичні товари. Датасет складається з метаінформації про товари та оглядів користувачів на товар. Головна мітка це рейтинг(рейтинг) набуває значень від 0 до 5. Для данного дослідження будемо вважати що якщо рейтинг більше 4 то товар є релевантним для користувача.

Нижче наведено лістинг коду завантаження датасету:

```
import datasets

datasets.logging.set_verbosity_error()

from datasets import load_dataset

reviews_dataset = load_dataset("McAuley-Lab/Amazon-Reviews-2023", "raw_review_All_Beauty", trust_remote_code=True)

meta_dataset = load_dataset("McAuley-Lab/Amazon-Reviews-2023", "raw_meta_All_Beauty", trust_remote_code=True)
```

Використовуючи бібліотеку HuggingFace datasets завантажуємо необхідні датасети.

В **Додатку А - Код препроцесингу даних** наведено лістинг коду де відбувається препроцессинг датасету.

Спочатку ми змінюємо назви ознак для зручності, прибираємо непотрібні ознаки та фільтруємо записи які не мають ідентифікаторів, використовючі бібліотеку HuggingFace datasets, це дозволить нам зменшити обсяг завантажуванних в оперативну пам’ять даних. Потім ми завантажуємо датасет в оперативну пам’ять, а саме в pandas.DataFrame для подальшого маніпулювання датасетом. Ми видаляємо дублікати товарів по ідентифікатору

якщо такі є. Об’єднуємо частини датасетів в один по ідентифікатору товару. Після цього кодуємо цільову ознаку, та ідентифікатори користувачів і товарів.

Далі для того щоб отримати вектори ознаки товарів ми об’єднуємо назву товару, його опис та його категорії в єдине текстове поле. Після цього ми виділяємо латентні ознаки товарів за допомогою TF-IDF статистики.

В якості ознаки користувача будемо використовувати середнє значення ознак товарів з якими він взаємодіяв. На цьому єтап препроцеснгу закінчено

В **Додатку А - Код моделі контентної фільтрації** наведено лістинг коду моделі контентної фільтрації де було реалізовано двобаштову модель, що використовує тільки ознаки користувача та товарів.

В **Додатку А - Код моделі колаборативної фільтрації** наведено лістинг коду реалізації моделі колаборативної фільтрації, де було реалізовано двобаштову модель, що використовує тільки ідинтифікатори користувача та товарів.

В **Додатку А - Код моделі гібридної фільтрації** наведено лістинг коду реалізації моделі гібридної фільтрації де було реалізовано двобаштову модель, що використовує і ідинтифікатори і ознаки користувача та товарів.

В **Додатку А - Код моделі навчання з підкріпленням** наведено лістинг коду реалізації моделі на основі навчання з підкріпленням де було реалізовано модель з навчанням з підкріпленням що оптимізує Q-функцію.

В **Додатку А - Код тренування моделі навчання з підкріпленням** наведено лістинг коду тренування моделі навчання з підкріпленням:

Кожен раунд ми перезвантажуємо середовище, і використовуємо початковий стан користувачів тобто їх ознаки. Після цього модель генерує рекомендації тобто дії. Ми беремо ці рекмоендації і змінюємо середовище, отримуючи новий стан і винагороду. Після цього ми оновлюємо модель, і повторюємо це в циклі.

В **Додатку А - Код оцінки моделей** наведено лістинг коду визначення основних метрик та фунціїї оцінки моделей. Де було реалізовано функції розрахунку метрик та функції оцінки.

4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ ТА АНАЛІЗ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ

4.1 Огляд основних метрик оцінки рекомендаційних систем

Для того щоб оцінити ефективність моделей було використано такі метрики як Precision_K, Recall_K, NDCG_K.

Precision_K – точність на відрізку K показує, яку частку рекомендованих у перших K позиціях елементів користувач вважає релевантними.

Якщо у нас є рекомендований список з K елементів, і з них R є релевантними для користувача, тоді:

Формула розрахунку Precision_K:

$$\text{Precision}_K = \frac{R}{K} \tag{3.14}$$

Високе значення Precision_K означає, що серед перших K рекомендацій велика частка справді цікава користувачу. Ця метрика фокусується на точності у верхній частині списку рекомендацій.

$\text{Recall}_K -$ повнота на відрізку K, показує, яку частку з усіх релевантних для користувача елементів ми змогли "впіймати" у перших K рекомендаціях.

Нехай у користувача є загалом T релевантних елементів, а у наших перших K рекомендаціях їх R:

Формула розрахунку $Recall_K$:

$$Recall_K = \frac{R}{T} \tag{3.15}$$

Високий $Recall_K$ означає, що у рекомендованому списку ми покрили більшість релевантних елементів. Це метрика повноти, яка показує, наскільки добре система не пропускає важливі для користувача предмети.

$NDCG_K$ – Normalized Discounted Cumulative Gain враховує не тільки факт наявності релевантних елементів у списку, а й їх позиції. Чим вище у списку розташовані релевантні елементи, тим краще.

Спершу обчислюємо DCG_K (англійською Discounted Cumulative Gain):

Формула DCG_K:

$$DCG_K = \sum_{i=1}^{K} \frac{rel_i}{\log_2(i+1)} \quad (3.16)$$

Де $rel_i = 1$, якщо елемент на позиції і релевантний, і 0 якщо ні.

Ідеальний варіант ($IDCG_K$) – це DCG для найкращого можливого впорядкування релевантних елементів.

Формула $NDCG_K$:

$$NDCG_K = \frac{DCG_K}{IDCG_K} \quad (3.17)$$

$NDCG_K$ дорівнює 1, якщо система ідеально впорядковує всі релевантні елементи на самому початку. Чим нижче NDCG@K, тим гірше ранжування. Ця метрика оцінює як наявність, так і позицію релевантних елементів, нагороджуючи елементи, які стоять вище.

4.2 Порівняння ефективності рекомендаційних систем.

Після оцінки моделей було отримано таблицю результатів:

Таблиця 3.1 – Таблиця оцінки моделей рекомендаційних систем

	Precision_K	$Recall_K$	$NDCG_K$
Модель контентної фільтрації	0.6732	0.7143	0.6937
Модель колаборативної фільтрації	0.7031	0.6123	0.6577
Гібридна модель	0.7563	0.7422	0.7492

Модель навчання з підкріпленням	0.8021	0.7944	0.7982

Як можна бачити з таблиці гібридна модель показує кращі резільтати ніж модель контентної фільтрації та модель колаборативної фільтрації. Але модель навчання з підкріпленням має кращі результати ніж гібридна модель.

4.3 Аналіз економічної ефективності

Аналіз економічної ефективності рекомендаційної системи дуже залежить від предметної області застосування. Так наприклад система для рекомендації відео має на меті збільшити показник залучення користувача, а рекомендаційна система товарів збільшити показник продажів. Тому розрахувати кількістно показник економівчної ефективності є складною задачею.

Нижче наведено основні аспекти та підходи до аналізу економічної ефективності рекомендаційної системи:

Вплив на обсяги продажів та дохід:

- Середній чек (англійською Average Order Value(AOV)): перевірити, чи середній розмір покупки користувачів, які бачать рекомендації, збільшився. Наприклад, якщо система пропонує взаємодоповнюючі товари, це може спонукати користувача купити додатковий товар і таким чином збільшити суму чеку;
- Конверсія (англійською Conversion Rate): виміряти, чи зросла ймовірність того, що користувач, який переглядає рекомендації, здійснить покупку. Вищий рівень конверсії означає більше продажів за той самий трафік;
- Продажі на одного користувача (англійською Revenue per User): порівняти продажі в розрахунку на одного користувача до та після впровадження рекомендацій. Позитивна динаміка свідчитиме про ефективність системи.

Покращення утримання клієнтів та лояльності:

- Частота повторних покупок: Якщо рекомендаційна система допомагає користувачам швидше знаходити цікаві їм товари, вони частіше повертатимуться в магазин. Зростання частоти повторних покупок підвищує довгострокову вартість клієнта (англійською Lifetime Value(LTV));
- Коефіцієнт утримання клієнтів (Retention Rate): Якщо завдяки релевантним рекомендаціям користувачі залишаються з брендом довше, це означає, що інвестиції в рекомендаційну систему окупляються.

Зниження витрат на маркетинг:

- Оптимізація асортименту та цін: fналіз даних рекомендаційної системи може допомогти краще зрозуміти, які товари варто просувати, на яких варто зробити знижки, а які взагалі виключити з асортименту. Це сприяє оптимізації ланцюга постачань і зменшенню витрат;
- Зменшення витрат на залучення клієнтів (CAC, Customer Acquisition Cost):Кращий клієнтський досвід та релевантні рекомендації можуть призвести до органічного зростання бази користувачів (через «сарафанне радіо»), знизивши потребу в дорогих маркетингових кампаніях.

Методики оцінки:

- A/B-тестування: порівняння групи користувачів, які бачать рекомендації, з контрольною групою, яка їх не бачить. Аналіз різниці в конверсії, середньому чеку, частоті повторних покупок та інших метриках дозволяє кількісно оцінити економічний ефект.
- До- та післяаналіз (англійською Pre-Post Analysis): Оцінка показників продажів до впровадження системи та після, з контролем за сезонністю та іншими факторами, що впливають на продажі.

Економічна ефективність рекомендаційної системи не вимірюється однією простою метрикою. Вона залежить від впливу на продажі, утримання клієнтів, зниження маркетингових витрат та оптимізацію асортименту. Використання A/B-тестування, порівняння до/після впровадження та розрахунок ROI допомагають отримати обґрунтований висновок щодо економічної доцільності використання рекомендаційної системи.

ВИСНОВК

В рамках кваліфікаційної роботи було проведено дослідження ефективності використання навчання з підкріпленням для рекмендаційних систем. Під час опрацювання задач роботи були проаналізовані існуючі підходи до розвязання задачі рекомендації такі як контентна фільтрація, колаборативна фльтрація та гібрідні методи. Було проаналізовано метод навчання з підкріпленням та цго використання в контексті рекомендаційних систем.

Було натреванно три класичні моделі для розвязання задачі рекомендації. Була натренована модель навчання з підкріпленням для задачі рекомендації. Для розробки моделей була використана мова програмуання Python та популярні бібліотеки машинного навчання.

Після тренування була проведена оцінка моделей основними технічними метриками і проаналізовано результати.

Дослідження показало що використання навчання з підкріпленням для розвязання задачі рекомендації є ефективним.

ПЕРЕЛІК ПОСИЛАНЬ

1. SHI Y. Collaborative Filtering beyond the User-Item Matrix: A Survey of the State of the Art and Future Challenges / Y. SHI, M. LARSON, A. HANJALIC. // ACM Computing Surveys. – 2014. – №47. – С. 3–45.
2. Breese J. Empirical Analysis of Predictive Algorithms for Collaborative Filtering [Електронний ресурс] / J. Breese, D. Heckerman, C. Kadie. – 2013. – Режим доступу до ресурсу: https://arxiv.org/pdf/1301.7363.
3. Billsus D. Learning Collaborative Information Filters [Електронний ресурс] / D. Billsus, M. Pazzani. – 1998. – Режим доступу до ресурсу: https://ics.uci.edu/~pazzani/Publications/MLC98.pdf.
4. Item-Based Collaborative Filtering Recommendation Algorithms [Електронний ресурс] / B.Sarwar, G. Karypis, J. Konstan, J. Riedl. – 2001. – Режим доступу до ресурсу: https://www.researchgate.net/publication/2369002_Item-based_Collaborative_Filtering_Recommendation_Algorithms.
5. Pavlov D. A Maximum Entropy Approach To Collaborative Filtering in Dynamic, Sparse, High-Dimensional Domains [Електронний ресурс] / D. Pavlov, D. Pennock. – 2002. – Режим доступу до ресурсу: https://proceedings.neurips.cc/paper_files/paper/2002/file/cc7e2b878868cbae992d1fb743995d8f-Paper.pdf.
6. Chen Y. A BAYESIAN MODEL FOR COLLABORATIVE FILTERING [Електронний ресурс] / Y. Chen, E. George. – 1999. – Режим доступу до ресурсу: https://www.researchgate.net/publication/228601858_A_bayesian_model_for_collaborative_filtering.
7. Getoor L. Using Probabilistic Relational Models for Collaborative Filtering [Електронний ресурс] / L. Getoor, M. Sahami. – 1999. – Режим доступу до ресурсу: https://madoc.univ-nantes.fr/mod/resource/view.php?id=237893.

8. Adomavicius G. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions / G. Adomavicius, A. Tuzhilin. // IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING. – 2005. – №17. – С. 734– 749.

9. Covington P. Deep Neural Networks for YouTube Recommendations [Електронний ресурс] / P. Covington, J. Adams, E. Sargin. – 2016. – Режим доступу до ресурсу: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45530.pdf.

10. Li Y. DEEP REINFORCEMENT LEARNING: AN OVERVIEW [Електронний ресурс] / Yuxi Li. – 2018. – Режим доступу до ресурсу: https://arxiv.org/pdf/1701.07274.

11. Schafer B. Recommender Systems in E-Commerce [Електронний ресурс] / B. Schafer, J. Konstan, J. Riedl. – 1999. – Режим доступу до ресурсу: https://www.researchgate.net/publication/2507550_Recommender_Systems_in_E-Commerce.

12. Sezgin E. A systematic literature review on Health Recommender Systems [Електронний ресурс] / E. Sezgin, S. Ozkan. – 2013. – Режим доступу до ресурсу: https://www.researchgate.net/publication/261488604_A_systematic_literature_review_on_Health_Recommender_Systems.

13. Bridging Language and Items for Retrieval and Recommendation [Електронний ресурс] / [Y. Hou, J. Li, Z. He та ін.]. – 2024. – Режим доступу до ресурсу: https://arxiv.org/pdf/2403.03952.

Додаток А – Код програми

Код завантаження датасету

```
import datasets

datasets.logging.set_verbosity_error()

from datasets import load_dataset

reviews_dataset = load_dataset("McAuley-Lab/Amazon-Reviews-2023", "raw_review_All_Beauty", trust_remote_code=True)

meta_dataset = load_dataset("McAuley-Lab/Amazon-Reviews-2023", "raw_meta_All_Beauty", trust_remote_code=True)
```

Код препроцесингу даних

```
def rename_reviews(example):
    return {
        "item_id": example["asin"],
        "review_text": example.get("text", ""),
    }

def rename_metadata(example):
    return {
        "item_id": example["parent_asin"],
    }

reviews_dataset = reviews_dataset.map(rename_reviews, remove_columns=["title", "asin", "parent_asin", "timestamp", "helpful_vote", "verified_purchase", "images", "text", "title"])
reviews_dataset = reviews_dataset.filter(lambda x: x["user_id"] is not None and x["item_id"] is not None and x["rating"] is not None)
```

```
    meta_dataset = meta_dataset.map(rename_metadata, remove_columns=["parent_asin", "main_category", "average_rating", "price", "videos", "store", "details", "images", "bought_together", "subtitle", "author", "details", "features", "rating_number"])
    meta_dataset = meta_dataset.filter(lambda x: x["item_id"] is not None)

    reviews_df = reviews_dataset["full"].to_pandas()
    metadata_df = meta_dataset["full"].to_pandas()

    metadata_df = metadata_df.drop_duplicates(subset=["item_id"])

    data_df = pd.merge(reviews_df, metadata_df, on="item_id", how="inner")

    data_df['label'] = (data_df['rating'] >= 4).astype(int)

    data_df['user_id_enc'] = data_df['user_id'].astype('category').cat.codes
    data_df['item_id_enc'] = data_df['item_id'].astype('category').cat.codes

    num_users = data_df['user_id_enc'].max() + 1
    num_items = data_df['item_id_enc'].max() + 1

    def combine_item_features(group):
        title = group['title'].iloc[0] if 'title' in group.columns and len(group) > 0 else ''
            description = group['description'].iloc[0] if 'description' in group.columns and len(group) > 0 else ''

        # Aggregate all categories from all rows
```

```
        all_cats = []
        if 'categories' in group.columns:
            for c_list in group['categories'].dropna():
                if isinstance(c_list, list):
                    all_cats.extend(c_list)
        cat_str = " ".join(all_cats)

        all_desc = []
        if 'description' in group.columns:
            for c_list in group['description'].dropna():
                if isinstance(c_list, list):
                    all_desc.extend(c_list)
        desc_str = " ".join(all_desc)

        # Create the combined text from title, description, and
categories
        combined_text = f"{title} {desc_str} {cat_str}".strip()
        return combined_text

    item_texts                                                  =
data_df.groupby('item_id_enc').apply(combine_item_features)

    vectorizer = TfidfVectorizer(max_features=5000)
    item_features_matrix                                        =
vectorizer.fit_transform(item_texts.values)

    train_df, test_df = train_test_split(data_df, test_size=0.2,
random_state=42)

    user_ids_train = train_df['user_id_enc'].values
    item_ids_train = train_df['item_id_enc'].values
    labels_train = train_df['label'].values
```

```
    user_interactions = data_df.groupby('user_id_enc')['item_id_enc'].apply(list).to_dict()

    user_feature_dim = item_features_matrix.shape[1]

    user_features_matrix = np.zeros((num_users, user_feature_dim), dtype=np.float32)

    for u_id, interacted_items in user_interactions.items():
        if len(interacted_items) > 0:
            user_item_vecs = item_features_matrix[interacted_items]
            user_features_matrix[u_id] = user_item_vecs.mean(axis=0)
        else:
            pass

    user_features_train = user_features_matrix[user_ids_train]
    item_features_train = item_features_matrix[item_ids_train]
```

Код моделі контентної фільтрації

```
    class ContentBasedTwoTowerModel:
        def __init__(self, user_feature_dim, item_feature_dim, embedding_dim=64, learning_rate=0.001):
            self.user_feature_dim = user_feature_dim
            self.item_feature_dim = item_feature_dim
            self.embedding_dim = embedding_dim
            self.learning_rate = learning_rate
            self.model = self.build_model()

        def build_model(self):
            user_features_input = Input(shape=(self.user_feature_dim,), name='user_features_input')
```

```
                                        item_features_input       =
Input(shape=(self.item_feature_dim,), name='item_features_input')

                    user_vector   =   Dense(self.embedding_dim,
activation='relu')(user_features_input)
                    item_vector   =   Dense(self.embedding_dim,
activation='relu')(item_features_input)

            dot_product = Dot(axes=1)([user_vector, item_vector])
            output = Dense(1, activation='sigmoid')(dot_product)

                  model   =   Model(inputs=[user_features_input,
item_features_input], outputs=output)
            model.compile(optimizer=Adam(learning_rate=self.lear
ning_rate), loss='binary_crossentropy', metrics=['accuracy'])
            return model

         def train(self, user_features, item_features, labels,
epochs=1, batch_size=32):
            print("Shapes before training:")
            print("user_features:", user_features.shape)
            print("item_features:", item_features.shape)
            print("labels:", labels.shape)

            self.model.fit(
                x={
                    'user_features_input': user_features,
                    'item_features_input': item_features
                },
                y=labels,
                epochs=epochs,
                batch_size=batch_size,
                validation_split=0.1
            )
```

```
        def recommend(self, user_id, user_features, item_ids,
item_features, top_k=10):
                user_feature_input = np.repeat(user_features,
len(item_ids)+1, axis=0)

            scores = self.model.predict({
                'user_features_input': user_feature_input,
                'item_features_input': item_features
            }, verbose=0).flatten()
               top_indices = np.argpartition(scores, -top_k)[-
top_k:]
            return item_ids[top_indices]
```

Код моделі колаборативної фільтрації

```
    class CollaborativeFilteringTwoTowerModel:
            def __init__(self, num_users, num_items,
user_feature_dim, item_feature_dim, embedding_dim=64,
learning_rate=0.001):
            self.num_users = num_users
            self.num_items = num_items
            self.user_feature_dim = user_feature_dim
            self.item_feature_dim = item_feature_dim
            self.embedding_dim = embedding_dim
            self.learning_rate = learning_rate
            self.model = self.build_model()

        def build_model(self):
                        user_id_input = Input(shape=(1,),
name='user_id_input')

                        item_id_input = Input(shape=(1,),
name='item_id_input')

            # ID embeddings only
```

```
                                        user_id_embedding        =
Embedding(input_dim=self.num_users,
output_dim=self.embedding_dim,
name='user_id_embedding')(user_id_input)
            user_id_embedding = Flatten()(user_id_embedding)

                                        item_id_embedding        =
Embedding(input_dim=self.num_items,
output_dim=self.embedding_dim,
name='item_id_embedding')(item_id_input)
            item_id_embedding = Flatten()(item_id_embedding)

                 dot_product  =  Dot(axes=1)([user_id_embedding,
item_id_embedding])
            output = Dense(1, activation='sigmoid')(dot_product)

            model = Model(inputs=[user_id_input, item_id_input],
outputs=output)
            model.compile(optimizer=Adam(learning_rate=self.lear
ning_rate), loss='binary_crossentropy', metrics=['accuracy'])
            return model

        def train(self, train_data, epochs=10, batch_size=256):
            user_ids, item_ids, labels = train_data
            # Ensure user_ids and item_ids are shaped (N,1)
            if len(user_ids.shape) == 1:
                user_ids = user_ids.reshape(-1,1)
            if len(item_ids.shape) == 1:
                item_ids = item_ids.reshape(-1,1)

            self.model.fit(
                x={
                    'user_id_input': user_ids,
                    'item_id_input': item_ids,
                },
```

```
                y=labels,
                epochs=epochs,
                batch_size=batch_size,
                validation_split=0.1
            )

        def recommend(self, user_id, user_features, item_ids,
item_features, top_k=10):
            # user_id: single integer
            # item_ids: (M,1)
             # Provide dummy user_features and item_features if
not using them
                                            user_array        =
np.array([user_id]*len(item_ids)).reshape(-1,1)

            scores = self.model.predict({
                'user_id_input': user_array,
                'item_id_input': item_ids,
            }, verbose=0).flatten()

               top_indices  =  np.argpartition(scores,  -top_k)[-
top_k:]
        return item_ids[top_indices]
```

Код моделі гібридної фільтрації

```
    class HybridTwoTowerModel:
              def    __init__(self,    num_users,    num_items,
user_feature_dim,       item_feature_dim,       embedding_dim=64,
learning_rate=0.001):
            self.num_users = num_users
            self.num_items = num_items
            self.user_feature_dim = user_feature_dim
            self.item_feature_dim = item_feature_dim
            self.embedding_dim = embedding_dim
```

```
            self.learning_rate = learning_rate
            self.model = self.build_model()

        def build_model(self):
                        user_id_input    =    Input(shape=(1,),
name='user_id_input')
                                    user_features_input       =
Input(shape=(self.user_feature_dim,), name='user_features_input')
                        item_id_input    =    Input(shape=(1,),
name='item_id_input')
                                    item_features_input       =
Input(shape=(self.item_feature_dim,), name='item_features_input')

            # ID embeddings
                                     user_id_embedding        =
Embedding(input_dim=self.num_users,
output_dim=self.embedding_dim,
name='user_id_embedding')(user_id_input)
            user_id_embedding = Flatten()(user_id_embedding)
            print(user_id_embedding.shape)

                                     item_id_embedding        =
Embedding(input_dim=self.num_items,
output_dim=self.embedding_dim,
name='item_id_embedding')(item_id_input)
            item_id_embedding = Flatten()(item_id_embedding)
            print(item_id_embedding.shape)

            # Concatenate ID embeddings with features
                user_concat  =  Concatenate()([user_id_embedding,
user_features_input])
                    user_vector   =   Dense(self.embedding_dim,
activation='relu')(user_concat)
```

```
                item_concat = Concatenate()([item_id_embedding,
item_features_input])
                    item_vector = Dense(self.embedding_dim,
activation='relu')(item_concat)

            dot_product = Dot(axes=1)([user_vector, item_vector])
            output = Dense(1, activation='sigmoid')(dot_product)

                    model = Model(inputs=[user_id_input,
user_features_input, item_id_input, item_features_input],
outputs=output)
            model.compile(optimizer=Adam(learning_rate=self.lear
ning_rate), loss='binary_crossentropy', metrics=['accuracy'])
            return model

        def train(self, train_data, epochs=10, batch_size=256):
              user_ids, user_features, item_ids, item_features,
labels = train_data
            # Ensure user_ids and item_ids are (N,1)
            if len(user_ids.shape) == 1:
                user_ids = user_ids.reshape(-1,1)
            if len(item_ids.shape) == 1:
                item_ids = item_ids.reshape(-1,1)

            self.model.fit(
                x={
                    'user_id_input': user_ids,
                    'user_features_input': user_features,
                    'item_id_input': item_ids,
                    'item_features_input': item_features
                },
                y=labels,
                epochs=epochs,
                batch_size=batch_size,
                validation_split=0.1,
```

```
                verbose=True
            )

        def recommend(self, user_id, user_features, item_ids,
item_features, top_k=10):
            # user_id: single int
            # item_ids: shape (M,1)
            # user_features: (1, user_feature_dim)
            # item_features: (M, item_feature_dim)
                                        user_array        =
np.array([user_id]*len(item_ids)).reshape(-1,1)
                user_features_array  =  np.tile(user_features,
(len(item_ids), 1))

            scores = self.model.predict({
                'user_id_input': user_array,
                'user_features_input': user_features_array,
                'item_id_input': item_ids,
                'item_features_input': item_features
            }, verbose=0).flatten()

              top_indices  =  np.argpartition(scores,  -top_k)[-
top_k:]
            return item_ids[top_indices]
```

Код моделі навчання з підкріпленням

```
class AdvancedRLModel():
    def __init__(
        self,
        num_users,
        num_items,
        user_feature_dim,
        item_feature_dim,
        embedding_dim=64,
```

```
            epsilon=0.1,
            gamma=0.99,
            learning_rate=0.001,
            buffer_size=10000,
            batch_size=64,
            target_update_freq=1000,
            double_q=True
        ):
            """
            Parameters:
            - num_users, num_items: Size of user/item id space.
            - user_feature_dim, item_feature_dim: Dimension of
user/item feature vectors.
            - embedding_dim: Dimension of learned embeddings.
            - epsilon: Exploration rate.
            - gamma: Discount factor for future rewards.
            - learning_rate: Learning rate for the optimizer.
            - buffer_size: Replay buffer max size.
            - batch_size: Minibatch size for updates.
            - target_update_freq: How often to update target
network.
            - double_q: Whether to use Double Q-learning.
            """
            self.num_users = num_users
            self.num_items = num_items
            self.user_feature_dim = user_feature_dim
            self.item_feature_dim = item_feature_dim
            self.embedding_dim = embedding_dim

            self.epsilon = epsilon
            self.gamma = gamma
            self.learning_rate = learning_rate
            self.buffer_size = buffer_size
            self.batch_size = batch_size
            self.target_update_freq = target_update_freq
```

```
            self.double_q = double_q

            # Replay buffer stores tuples of:
            #  (user_id, user_features, item_id, item_features,
reward, next_user_features, done_flag)
            # done_flag indicates whether the episode ended after
this step.
            self.replay_buffer = deque(maxlen=self.buffer_size)

            # Build main and target networks
            self.model = self.build_model()
            self.target_model = clone_model(self.model)

self.target_model.set_weights(self.model.get_weights())

            self.steps = 0

        def build_model(self):
            """
            Builds a Q-network that, given (user, user_features,
item),
            predicts the Q-value. We'll pass item inputs
separately when needed.

            However, to select the best action, we often need to
evaluate multiple items.
            For training, we handle single (user, item) pairs at
a time.
            """
            user_id_input          =          Input(shape=(1,),
name='user_id_input')
            user_features_input                                  =
Input(shape=(self.user_feature_dim,), name='user_features_input')

            user_id_embedding = Embedding(
```

```
                input_dim=self.num_users,
                output_dim=self.embedding_dim,
                name='user_id_embedding'
            )(user_id_input)
            user_id_embedding = Flatten()(user_id_embedding)

            user_vector    =    Concatenate()([user_id_embedding,
user_features_input])
            user_vector        =        Dense(self.embedding_dim,
activation='relu')(user_vector)

            item_id_input           =           Input(shape=(1,),
name='item_id_input')
            item_features_input                                    =
Input(shape=(self.item_feature_dim,), name='item_features_input')

            item_id_embedding = Embedding(
                input_dim=self.num_items,
                output_dim=self.embedding_dim,
                name='item_id_embedding'
            )(item_id_input)
            item_id_embedding = Flatten()(item_id_embedding)

            item_vector    =    Concatenate()([item_id_embedding,
item_features_input])
            item_vector        =        Dense(self.embedding_dim,
activation='relu')(item_vector)

            # Combine user and item vectors to get Q-value
            q_input = Concatenate()([user_vector, item_vector])
            q_value          =          Dense(self.embedding_dim,
activation='relu')(q_input)
            q_value = Dense(1, activation='linear')(q_value)

            model = Model(
```

```
                inputs=[user_id_input,      user_features_input,
item_id_input, item_features_input],
                outputs=q_value
            )

model.compile(optimizer=Adam(learning_rate=self.learning_rate),
loss='mse')
            return model

        def train(self, train_data):
            """
            Pre-train on historical data if available.
            train_data:   (user_ids,   user_features,   item_ids,
item_features, rewards)
            This is optional and depends on whether you have
static data to pretrain on.
            """
            user_ids,  user_features,  item_ids,  item_features,
rewards = train_data
            self.model.fit(
                x={
                    'user_id_input': user_ids,
                    'user_features_input': user_features,
                    'item_id_input': item_ids,
                    'item_features_input': item_features
                },
                y=rewards,
                epochs=5,
                batch_size=self.batch_size,
                validation_split=0.1
            )

        def     recommend(self,     user_id,     user_features,
candidate_item_ids, candidate_item_features, top_k=10):
            """
```

```
            Epsilon-greedy recommendation:
            - With probability epsilon, choose random items.
            - Otherwise, choose items with the highest Q-values.
            """
            if np.random.rand() < self.epsilon:
                # Exploration
                chosen_indices                                    =
np.random.choice(len(candidate_item_ids),              size=top_k,
replace=False)
                recommended_item_ids                              =
candidate_item_ids[chosen_indices]
            else:
                #  Exploitation:  Evaluate  Q-values  for  all
candidate items
                user_array      =      np.array([user_id]      *
len(candidate_item_ids))
                user_features_array   =   np.tile(user_features,
(len(candidate_item_ids), 1))
                q_values = self.model.predict({
                    'user_id_input': user_array,
                    'user_features_input': user_features_array,
                    'item_id_input': candidate_item_ids,
                    'item_features_input':
candidate_item_features
                }, verbose=0).flatten()

                top_indices   =   np.argpartition(q_values,   -
top_k)[-top_k:]
                recommended_item_ids                              =
candidate_item_ids[top_indices]

            return recommended_item_ids

        def  update(self,  user_id,  user_features,  item_id,
item_features, reward, next_user_features, done=False):
```

```
            """
            Store experience and periodically train from replay
buffer.
            For Q-learning updates, we need next state features
and a notion of done (end of episode).
            """
            self.replay_buffer.append((user_id,   user_features,
item_id, item_features, reward, next_user_features, done))
            self.steps += 1

            if len(self.replay_buffer) >= self.batch_size:
                self.train_from_replay()

            # Update target network periodically
            if self.steps % self.target_update_freq == 0:

self.target_model.set_weights(self.model.get_weights())

        def train_from_replay(self):
            batch      =      random.sample(self.replay_buffer,
self.batch_size)

            user_ids,         user_feat_batch,         item_ids,
item_feat_batch,   rewards,   next_user_feat_batch,   dones   =
zip(*batch)

            user_ids = np.array(user_ids)
            user_feat_batch = np.array(user_feat_batch)
            item_ids = np.array(item_ids)
            item_feat_batch = np.array(item_feat_batch)
            rewards = np.array(rewards).reshape(-1, 1)
            next_user_feat_batch                                =
np.array(next_user_feat_batch)
            dones = np.array(dones)
```

```
            # To compute the target Q-values, we need to find max
over next actions.
            # For a large item space, consider sampling a subset
of items or known candidates.
            # Here, we assume a fixed set of candidate items is
available at inference time.
            # For demonstration, let's say we consider all items
(which can be expensive).
            # In practice, sample a subset or maintain a candidate
pool.

            # Get Q-values of next state for all items from main
network
            # and Q-values of next state from the target network
            # This step can be computationally heavy if num_items
is large.
            # For simplicity, let's assume we have a method to
get next state Q-values from target network.

            # Placeholder: Suppose we have a fixed set of
candidate_items for next step
            # In reality, you might sample or have a separate
function to get these Q-values.
            candidate_items = np.arange(self.num_items)  # all
items
            candidate_items = candidate_items.reshape(-1, 1)
            candidate_item_features = np.zeros((self.num_items,
self.item_feature_dim))  # Placeholder features

            # Expand next_user_feat_batch to match candidate
items
            expanded_next_user_ids    =    np.repeat(user_ids,
self.num_items)
            expanded_next_user_feat                           =
np.repeat(next_user_feat_batch, self.num_items, axis=0)
```

```
            expanded_item_ids     =     np.tile(candidate_items,
(self.batch_size, 1))
            expanded_item_feat                                    =
np.tile(candidate_item_features, (self.batch_size, 1))

            # Predict Q-values for next state from main model and
target model
            q_next_main = self.model.predict({
                'user_id_input': expanded_next_user_ids,
                'user_features_input': expanded_next_user_feat,
                'item_id_input': expanded_item_ids,
                'item_features_input': expanded_item_feat
            },               verbose=0).reshape(self.batch_size,
self.num_items)

            q_next_target = self.target_model.predict({
                'user_id_input': expanded_next_user_ids,
                'user_features_input': expanded_next_user_feat,
                'item_id_input': expanded_item_ids,
                'item_features_input': expanded_item_feat
            },               verbose=0).reshape(self.batch_size,
self.num_items)

            if self.double_q:
                # Double Q-Learning:
                # main network selects argmax action
                next_actions = np.argmax(q_next_main, axis=1)
                # target network evaluates the chosen action
                q_next                                            =
q_next_target[np.arange(self.batch_size), next_actions].reshape(-
1, 1)
            else:
                # Regular DQN:
                # Use the target network to find max Q-value for
next state
```

```
                q_next = np.max(q_next_target, axis=1).reshape(-
1, 1)

            # Compute TD targets
            # If done, no future rewards are considered
            td_targets = rewards + (1 - dones.reshape(-1, 1)) *
self.gamma * q_next

            # Current Q-values for chosen actions
            current_q = self.model.predict({
                'user_id_input': user_ids,
                'user_features_input': user_feat_batch,
                'item_id_input': item_ids,
                'item_features_input': item_feat_batch
            }, verbose=0)

            # Train the model to reduce the difference between
current_q and td_targets
            self.model.fit(
                x={
                    'user_id_input': user_ids,
                    'user_features_input': user_feat_batch,
                    'item_id_input': item_ids,
                    'item_features_input': item_feat_batch
                },
                y=td_targets,
                epochs=1,
                batch_size=self.batch_size,
                verbose=0
            )
```

Код тренування моделі навчання з підкріпленням

```
env = RecommendationEnv(
    num_users=num_users,
    num_items=num_items,
```

```
        user_feature_dim=user_feature_dim,
        item_feature_dim=item_feature_dim,
         candidate_item_ids=candidate_item_ids.flatten(),   # Env
uses flat array for actions
        candidate_item_features=candidate_item_features,
        max_steps=10
    )

    # Create the RL model
    model = AdvancedRLModel(
        num_users=num_users,
        num_items=num_items,
        user_feature_dim=user_feature_dim,
        item_feature_dim=item_feature_dim,
        embedding_dim=64,
        epsilon=0.1,         # Exploration rate
        gamma=0.99,          # Discount factor
        learning_rate=0.001, # Learning rate
        buffer_size=10000,
        batch_size=64,
        target_update_freq=1000,
        double_q=True
    )

    # Training parameters
    num_episodes = 1000

    for episode in range(num_episodes):
        # Reset environment
        user_features = env.reset()  # shape: (user_feature_dim,)
          user_id  =  np.random.randint(0,  num_users)   #  Random
user_id assigned for demo
        done = False
        step_count = 0
```

```
        while not done:
            # Current state
            state_user_features = user_features.reshape(1, -1)  # reshape for model input

            # The model expects candidate items shaped as (N,1)
            # and user_features as (1, user_feature_dim)
            # For recommendation, top_k=1 for simplicity
            recommended_item_ids = model.recommend(
                user_id=user_id,
                user_features=state_user_features,
                   item_ids=candidate_item_ids,          # shape (num_items, 1)
                item_features=candidate_item_features,
                top_k=1
            )

            chosen_item_id = recommended_item_ids[0]

            # Convert chosen_item_id to action index used by the env
                                        #        Since        env action_space=Discrete(len(candidate_item_ids)),
            # action = index in candidate_item_ids
              action = np.where(candidate_item_ids.flatten() == chosen_item_id)[0][0]

            # Step through environment
                  next_user_features,  reward,  done,  info  = env.step(action)

            # Convert to required shapes
            chosen_item_id_array = np.array([[chosen_item_id]])
                                    chosen_item_features       = candidate_item_features[action].reshape(1, -1)
```

```
                                    next_user_features_array      =
next_user_features.reshape(1, -1)

                # Update the model: model.update(state, action,
reward, next_state)
            # In update method, we also pass user_id & item_id
            model.update(
                user_id=user_id,
                user_features=state_user_features,
                item_id=chosen_item_id_array,
                item_features=chosen_item_features,
                reward=reward,
                next_user_features=next_user_features_array,
                done=done
            )

            user_features = next_user_features
            step_count += 1

        if (episode + 1) % 100 == 0:
                    print(f"Episode   {episode+1}/{num_episodes}
completed.")

    print("Training completed.")
```

Код оцінки моделей

```
    def precision_at_k(recommended_items, relevant_items, k):
        recommended_k = recommended_items[:k]
        print(f"recommended_k: {recommended_k}")
        print(f"Relevant: {relevant_items}")
        hits = len(set(recommended_k) & set(relevant_items))
        return hits / k

    def recall_at_k(recommended_items, relevant_items, k):
```

```
        recommended_k = recommended_items[:k]
        hits = len(set(recommended_k) & set(relevant_items))
        return hits / len(relevant_items) if len(relevant_items)
> 0 else 0.0
    def recall_at_k(recommended_items, relevant_items, k):
        recommended_k = recommended_items[:k]
        hits = len(set(recommended_k) & set(relevant_items))
        return hits / len(relevant_items) if len(relevant_items)
> 0 else 0.0

    import math

    def ndcg_at_k(recommended_items, relevant_items, k):
        recommended_k = recommended_items[:k]
        dcg = 0.0
        for i, item in enumerate(recommended_k):
            if item in relevant_items:
                dcg += 1 / math.log2(i+2)  # positions are zero-
based in code

        # Ideal DCG
        ideal_hits = min(len(relevant_items), k)
        idcg = 0.0
        for i in range(ideal_hits):
            idcg += 1 / math.log2(i+2)

        return dcg / idcg if idcg > 0 else 0.0

    def evaluate_model(model, test_users, user_relevant_items,
candidate_item_ids, candidate_item_features, user_features_dict,
top_k=10):
        precision_scores = []
        recall_scores = []
        ndcg_scores = []
        for user_id in test_users:
```

```
                relevant_items = user_relevant_items.get(user_id,
set())
            if len(relevant_items) == 0:
                continue
                                        user_features        =
user_features_dict[user_id].reshape(1, -1)
            recommended_items = model.recommend(
                user_id=user_id,
                user_features=user_features,
                item_ids=candidate_item_ids,
                item_features=candidate_item_features,
                top_k=num_items
            )
                                    recommended_top_k        =
recommended_items[:top_k].flatten()
            p = precision_at_k(recommended_top_k, relevant_items,
top_k)
                r = recall_at_k(recommended_top_k, relevant_items,
top_k)
                 n = ndcg_at_k(recommended_top_k, relevant_items,
top_k)

                print(f"Precision@{top_k}={p}, Recall@{top_k}={r},
NDCG@{top_k}={n}")

            precision_scores.append(p)
            recall_scores.append(r)
            ndcg_scores.append(n)

        return {
                  'precision@k': np.mean(precision_scores) if
precision_scores else 0.0,
             'recall@k': np.mean(recall_scores) if recall_scores
else 0.0,
```

```
            'ndcg@k': np.mean(ndcg_scores) if ndcg_scores else
0.0
        }
```