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: «Рекомендаційні системи на основі аналізу послідовності кліків в сфері електронної комерції»

Виконав:
Студент IV курсу, групи КА-22
Марченко Дмитро Сергійович _____

Керівник:
Професор кафедри ММСА, д.т.н.
Недашківська Надія Іванівна _____

Консультант з економічного розділу:
Доцент кафедри ТПЕ, к.е.н.
Рощина Надія Василівна _____

Консультант з нормоконтролю:
Асистент кафедри ММСА, к.т.н.
Канцедал Георгій Олегович _____

Рецензент:
Доцент кафедри СП, к.т.н.
Гіоргізова-Гай В.Ш. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _Марченко Д.С._

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
Марченку Дмитру Сергійовичу

1. Тема роботи «Рекомендаційні системи на основі аналізу послідовності кліків в сфері електронної комерції», керівник роботи Недашківська Надія Іванівна, доктор технічних наук, професор, затверджені наказом по університету від __.__.2026 р. № _____
2. Термін подання студентом роботи «____» червня 2026 р.
3. Вихідні дані до роботи: науково-методична література за темою роботи, публічні датасети з історією взаємодій користувачів з товарами OpenCDP та Retailrocket, програмні засоби мови Python та публічні бібліотеки
4. Зміст роботи: дослідження предметної області; обробка та підготовка даних для навчання моделей; реалізація та аналіз ефективності різних архітектур рекурентних нейронних мереж; функціонально-вартісний аналіз програмного продукту.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): схеми архітектур нейронних мереж та загальної архітектури системи, лендингова сторінка користувацького застосунку, презентація.

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	Затвердження теми бакалаврської дипломної роботи (БДР), опанування ДСТУ 3008:2015	26.02.2026-28.02.2026	виконано
2	Огляд теоретичної інформації по темі роботи	20.03.2026-24.03.2026	виконано
3	Ознайомлення та розбір основних підходів до побудови рекомендаційних систем	27.03.2026-1.04.2026	виконано
4	Проектування систем і розробка програмного забезпечення	4.04.2026-08.05.2026	виконано
5	Оформлення пояснювальної записки (ПЗ), перевірка правильності структури та оформлення	11.05.2026-15.05.2026	виконано
6	Аналіз результатів та порівняння ефективності побудованих моделей	18.05.2026-22.05.2026	виконано
7	Остаточне оформлення ПЗ, підготовка презентації	23.05.2026-10.06.2026	виконано
8	Передзахист БДР	11.06.2026	виконано

Студент _____ Дмитро МАРЧЕНКО

Керівник _____ Надія НЕДАШКІВСЬКА

РЕФЕРАТ

Дипломна робота: 132 с., 16 рис., 15 табл., 27 джерел, 2 додатки.

РЕКОМЕНДАЦІЙНІ СИСТЕМИ, ПОСЛІДОВНІ РЕКОМЕНДАЦІЇ, АНАЛІЗ ПОСЛІДОВНОСТІ КЛІКІВ, ЕЛЕКТРОННА КОМЕРЦІЯ, РЕКУРЕНТНІ НЕЙРОННІ МЕРЕЖІ, LSTM, GRU, SASREC, МЕХАНІЗМ УВАГИ, ГЛИБОКЕ НАВЧАННЯ.

Об'єкт дослідження – процес формування персоналізованих рекомендацій товарів на платформах електронної комерції на основі послідовностей взаємодій користувачів.

Предмет дослідження – моделі та методи побудови послідовних рекомендаційних систем із використанням рекурентних нейронних мереж і механізмів уваги.

Мета роботи – реалізувати та порівняти архітектури рекомендаційних систем, що спираються на аналіз послідовності кліків у сфері e-commerce.

У роботі розглянуто класичні підходи до побудови рекомендаційних систем – колаборативну та контентну фільтрацію, ланцюги Маркова – і послідовні методи на їх противагу. Описано математичний апарат LSTM і GRU, а також механізм уваги як основу архітектури SASRec. Окремо висвітлено навчальний процес: оптимізатор Adam, косинусний відпал швидкості навчання, L2-регуляризацію та метрики якості ранжування HR@K, Precision@K, MAP@K і NDCG@K.

Систему повністю реалізовано на Python з використанням pandas, PyTorch і Polars. Реалізація охоплює попередню обробку наборів даних Retailrocket і OpenCDP, формування послідовностей взаємодій, навчання й збереження ваг моделей, а також веб-застосунок на FastAPI, розгорнутий через Docker. За результатами експериментів при K = 5, 10, 20 найкращі показники на Retailrocket показала модель LSTM4Rec, на OpenCDP – SASRec . Додатково проведено функціонально-вартісний аналіз із обґрунтуванням обраного варіанта реалізації.

Розроблена система підтвердила свою працездатність і придатна до інтеграції в реальні платформи електронної комерції.

ABSTRACT

Diploma thesis: 132 p., 16 fig., 15 tab., 27 references, 2 appendices.

RECOMMENDER SYSTEMS, SEQUENTIAL RECOMMENDATION, CLICKSTREAM ANALYSIS, E-COMMERCE, RECURRENT NEURAL NETWORKS, LSTM, GRU, SASREC, ATTENTION MECHANISM, DEEP LEARNING.

The object of research is the process of generating personalized product recommendations on e-commerce platforms based on user interaction sequences.

The subject of research is the models and methods for building sequential recommender systems using recurrent neural networks and attention mechanisms.

The aim of the work is to implement and compare recommender system architectures that rely on clickstream analysis in the e-commerce domain.

The thesis covers classical recommender system approaches – collaborative and content-based filtering, Markov chains – alongside sequential methods as an alternative. The mathematical foundations of LSTM and GRU are outlined, along with the attention mechanism as the core of the SASRec architecture. The training setup is also described: the Adam optimizer, cosine annealing of the learning rate, L2 regularization, and the ranking quality metrics HR@K, Precision@K, MAP@K, and NDCG@K.

The system was fully implemented in Python with pandas, PyTorch, and Polars. The implementation covers preprocessing of the Retailrocket and OpenCDP datasets, construction of interaction sequences, model training and weight saving, and a FastAPI web application deployed via Docker. Experiments at K = 5, 10, 20 showed that LSTM4Rec performed best on Retailrocket, while SASRec led on OpenCDP. A functional-cost analysis was also carried out to justify the chosen implementation approach.

The resulting system demonstrated its practical viability and is ready for integration into real-world e-commerce platforms.

ЗМІСТ

ВСТУП

Однією з найважливіших і найскладніших задач в електронній комерції, інтернет-кінотеатрах, стрімингових платформах тощо завжди було і залишається формування рекомендацій на основі вподобань користувачів. У сучасному, перенасиченому інформацією світі, залученість користувача стала найціннішим ресурсом, в кожного амбітного продукту постає ціль втримати увагу користувача якомога довше. Істотну роль в цьому грають рекомендаційні системи. Їхня основна роль в сучасному цифровому бізнесі – налагодити зв’язок між користувачем та контентом з максимальним врахуванням особистих смаків, за можливості, настрою та фінансових спроможностей даного користувача. Для одних продуктів це є просто корисною функцією, а для інших – життєвою необхідністю.

В даній роботі буде проаналізовано методи побудови персоналізованих рекомендацій, розглянуто найпоширеніші сучасні архітектури рекомендаційних систем, їх математичний апарат, особливості, переваги та недоліки.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗРОБКИ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ В СФЕРІ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

1.1 Поняття рекомендаційної системи

Рекомендаційна система – це застосунок, що може з достатньою часткою впевненості надавати пропозиції користувачу релевантного контенту [1]. Вони використовуються в онлайн-кінотеатрах на кшталт Megogo чи Netflix, стрімингових музичних платформах, як Spotify чи Last.fm, інтернет-маркетплейсах, як Rozetka, Temu, та в будь-яких інших цифрових продуктах, що мають значну базу свого або представленого користувачами контенту. У всіх схожих продуктів досить швидко виникає проблема інформаційного перевантаження, тобто практичної неможливості або недоцільності перегляду користувачем всього каталогу наявних товарів або послуг, що автоматично призводить до необхідності постачати персоналізовані рекомендації [2]. До того ж, персоналізовані рекомендації формують звичку до продукту, і збільшують залученість, а саме екранний час для стрімингових платформ, відео-хостингів та онлайн-кінотеатрів, або кількість покупок, і, відповідно, дохід з користувача для інтернет-маркетплейсів. Побудова такої системи досить важлива задача для сучасного бізнесу, адже зараз один із основних та водночас найдорожчих ресурсів – це увага користувача, проблему утримання якої коректно спроєктовані рекомендаційні системи досить добре вирішують. Водночас, побудувати таку систему досить складно, і причиною цьому є значна кількість факторів. По-перше, на відміну від задач класифікації, ми не маємо можливості напряму зробити строгий висновок про те, чи є сформована рекомендація релевантною конкретному користувачу і саме в конкретний момент. На додачу до цього, якщо користувач не здійснив на товар, це саме по собі не є негативним сигналом, цілком можливо, що користувачу даний товар

або в поточний момент не є актуальним, або є інший, що задовольняє його потреби та вподобання більше. По-друге, в контексті рекомендаційних систем одна із найсерйозніших і найскладніших проблем – це так звана проблема «холодного старту» [1]. Вона полягає в тому, що у випадку нових, неактивних або анонімних користувачів ми або не маємо даних взагалі, або не маємо в достатньому обсязі, і, відповідно, не можемо надати персоналізовані рекомендації для даного користувача. Аналогічна проблема виявляється у випадку появи нового товару, з яким не було ніякої взаємодії. В такому разі можемо орієнтуватися лише на атрибути самого товару.

1.2 Електронна комерція

Платформи електронної комерції являють собою досить специфічні з точки зору аналізу користувацьких даних цифрові продукти. По-перше, левову частку користувачів складають аноніми, які користуються веб-сайтом без реєстрації. Це складає певну проблему у формуванні довгострокових профілів користувачів, адже відокремлення окремих анонімних користувачів обмежується лише cookies та IP-адресою. По-друге, попит не є стабільним у часі, розподіл кліків міняється під час свят і розпродаж. Також сильний вплив акційних пропозицій – користувач може придбати конкретний товар лише тому, що на нього була знижка, і рекомендувати його поза межами акцій для цього користувача немає сенсу.

Сучасна структура інтернет-маркетплейсів виглядає приблизно таким чином:

1. Ієрархічний список категорій: головні категорії та підкатегорії товарів.
2. Пошуковий рядок.
3. Рекомендації товарів (у випадку нових користувачів неперсоналізовані, на основі переваг схожих користувачів, у випадку користувачів з певною послідовністю взаємодій достатньої довжини – персоналізовані рекомендації).

Також важливий аспект – це користувацькі сесії. Сесія – це центральне поняття для аналізу послідовностей кліків [3]. Вона визначається як неперервна серія взаємодій одного користувача в межах обмеженого часового вікна. Повертаючись до проблеми генерування рекомендацій для анонімних користувачів, для них надійно можемо згенерувати лише в межах поточної сесії. При цьому з'являється додаткова проблема, адже залишається можливість, що один користувач фактично створив кілька паралельних сесій, що спотворює вхідні дані для генерації рекомендацій.

Якщо ж говорити про середній життєвий цикл користувача, то його поведінка ділиться на кілька фаз, які можна зобразити схематично:

Рис. 1.1 Схематичне зображення однієї ітерації життєвого циклу користувача

Основними важливими типами взаємодії користувача з товарами є перегляд їх сторінок, додання товарів до списку бажаного або до кошика і намір про покупку/успішна транзакція [4].

Важливо також розуміти відмінності e-commerce від інших суміжних доменів:

1. Зазвичай повторне споживання одного типу товарів має досить низьку періодичність, на відміну від, наприклад, прослуховування музики.

2. Якість рекомендацій напряму впливає на середній чек і дохід з користувача.

1.3 Огляд існуючих підходів до побудови рекомендаційних систем

На сьогодні існує досить багато підходів до побудови рекомендаційної системи. Кожен із них використовується для різних цілей і працює з різними форматами вхідних даних. Розглянемо деякі з підходів детальніше:

1.3.1 Колаборативна фільтрація (Collaborative Filtering, далі CF)

Колаборативна фільтрація – це один із найстаріших та найбільш досліджених методів формування рекомендацій [2][5]. Його ключова ідея полягає в тому, що користувачі зі схожою історією взаємодій мають схожі вподобання. Memory-based підходи працюють безпосередньо з матрицею взаємодій користувача і не використовують методи машинного навчання. Буває як user-based CF, так і item-based CF. Перший знаходить K найближчих користувачів і рекомендує товари, які вони вподобали, але поточний користувач ще не бачив. Для цього використовуються різноманітні метрики схожості, такі як косинусна подібність, кореляція Пірсона, міра Жакара та інші. Основними проблемами цього методу є погана масштабовність при наявності великого каталогу товарів або користувачів, неможливість враховувати порядок, часові інтервали та інші взаємозв’язки взаємодій між собою. Також цей метод не вирішує проблему холодного старту.

До методів колаборативної фільтрації також належать model-based підходи, зокрема факторизація матриці взаємодій [4]. Це і SVD, і SVD++, і

TimeSVD++ та інші методи, пов’язані з розкладом матриці взаємодій на 2 і більше.

1.3.2 Контентна фільтрація (Content-Based Filtering)

Контентна фільтрація рекомендує товари, які схожі за атрибутами до тих, з якими користувач взаємодіяв раніше. Для обробки категорійних змінних використовуються класичні методи інженерії ознак, також можлива обробка текстових описів та зображень товарів, вони обробляються такими методами як TF-IDF або за допомогою трансформерів на кшталт BERT у випадку тексту та згортковими нейронними мережами у випадку зображень [5][6]. Однією із переваг цього методу є те, що він частково вирішує проблему холодного старту, а саме надає можливість рекомендувати нові товари, з якими користувачі ще не взаємодіяли. З іншої сторони, проблемами цього методу є те, що користувач може отримати лише ті категорії товарів, що вже бачив, також залишається невирішеною проблема генерації рекомендацій для нових користувачів.

1.3.3 Гібридні методи

Гібридні методи поєднують колаборативну та контенту фільтрацію з ціллю покриття і вирішення проблем, які є в обох цих методів окремо [7]. Для цього використовуються різні підходи, наприклад зважене об’єднання, коли оцінки від моделей колаборативної фільтрації (умовно позначимо CF_score) і контентної фільтрації (умовно CB_score) об’єднуються за формулою на кшталт:

$$score = \alpha * CF_{score} + (1 - \alpha) * CB_{score},$$

де α визначається емпіричним шляхом [2]. Комбінування прогнозів моделей CF та текстових подібностей методом стекінгу дозволяє отримати точніші оцінки ніж використання окремих алгоритмів [8]. Також помітне зростання ефективності моделей було досліджено на прикладі побудови ансамблю моделей на основі графової моделі GraphSAGE [9]. Особливим класом гібридних рекомендаційних систем є моделі, що застосовують спільне вивчення властивостей предмета та поведінки користувача на основі тексту огляду товару, наприклад DeepCoNN [10].

1.3.4 Підходи на основі ланцюгів Маркова

Це один із найпростіших в реалізації класів методів, що явно моделють саме послідовності взаємодій [3]. В контексті генерації рекомендацій, їх ідея полягає в тому, щоб на етапі навчання згенерувати матрицю переходів розмірності |I|x||I|, де I – множина наявних товарів, для подальшої генерації рекомендацій на основі кількох останніх товарів, з якими взаємодіяв користувач (у випадку моделі першого порядку). Основна проблема цих методів у тому, що вони мають досить коротку пам’ять і формують рекомендації для наступного товару лише на основі знань про 1-2 попередніх. Це призводить до того, що рідкісні товари зазвичай не потрапляють у видачу рекомендаційної системи. Причиною цьому є збільшення використання пам’яті для зберігання і використання матриці переходів в залежності $O(n^k)$, де k – порядок моделі. В такому випадку, при наявності 10000 товарів, кількість елементів матриці для моделі другого порядку становить 10^8, що робить модель фактично неможливою для використання на практиці. Таким чином, у разі застосування ланцюгів Маркова для генерації рекомендацій, зазвичай використовуються моделі 1-2 порядку і агресивні методи прунінгу (усунення) рідкісних переходів.

1.3.5 Рекурентні нейронні мережі (RNN)

На відміну від простих моделей на основі ланцюгів Маркова, рекурентні нейронні мережі мають значно складнішу архітектуру, що дозволяє збільшити їх пам’ять і, відповідно, покращити якість сесійних рекомендацій.

Найпопулярнішими рекурентними нейронними мережами в побудові рекомендаційних систем є такі моделі як LSTM4Rec (Long-Short Term Memory) та GRU4Rec (Gated Recurrent Unit).

У базових архітектур RNN, так званих Vanilla RNN, існує критична проблема, пов’язана із зниканням або так званим «вибухом» градієнтів [11][12]. Вона полягає в тому, що матриця вхідних зв’язків (як в ланцюгах Маркова) при зворотньому поширенні помилки під час навчання моделі множиться на значну кількість матриць, що при великій кількості шарів моделі зануляє або збільшує градієнти до надмірних значень, що, по-перше, негативно впливає на навчання моделі, і, по-друге, робить такі моделі практично нездатними захоплювати довгострокові залежності в послідовностях при навчанні.

Архітектура LSTM вирішує проблему зникаючого градієнта через гейтовий механізм керування інформаційним потоком [12]. Ключова її ідея полягає в тому, що окрім прихованого стану h_t, як в Vanilla RNN, LSTM також підтримує так званий «клітинний» стан c_t, «стрічку пам’яті», яка може зберігати інформацію протягом довгих послідовностей без деградації градієнта. Гейтовий механізм полягає в тому, що рекурентний модуль LSTM на відміну від Vanilla RNN, складається не з одного, а з кількох шарів, що формують три умовних гейти: гейт забування, який визначає, яку частину кожного блоку інформації слід пропустити далі по мережі; вхідний гейт, який змінює старий стан клітинки c_{t-1} на c_t і вихідний гейт, який приймає рішення про те, яку інформацію з клітинки пам’яті виводити і з якою інформацією з вхідних даних її об’єднувати. Проблема зникаючого градієнта вирішується тим, що для збереження інформації використовується сума, а не добуток,

відповідно, вона може зберігатися довше і рішення про забування старих даних стає фактором, на який вже можемо мати більше впливу шляхом оптимізації ваг у шарах в гейті забування.

GRU – це трохи простіша архітектура, порівняно із LSTM, оскільки в ній, по-перше, об’єднано клітинний та прихований стани і, по-друге, гейти забування і вхідний об’єднані в один – гейт оновлення [13]. Так досягається більша швидкість навчання та інференсу, але незначно зменшується якість [14].

1.3.6 Механізми уваги та Transformer-архітектури

Трансформери, архітектура, що початково запроваджувалася для вирішення задач NLP (обробки природної мови), показують непогані результати також і в задачах побудови рекомендацій на основі послідовностей взаємодій користувачів, оскільки по-перше, є логічним продовженням ідеї RNN і похідних моделей (LSTM, GRU та інші), і, по-друге, їх основна галузь використання структурно схожа, оскільки також полягає в обробці послідовностей, а саме послідовностей токенів [15][16]. Серед цього класу моделей варто зазначити такі архітектури як SASRec та BERT4Rec.

SASRec є однонаправленою transformer-архітектурою, що складається з побудови ембедингів для товарів та позиційних ембедингів, механізму уваги і повнозв’язних шарів нейронної мережі [17]. Слід звернути увагу на важливість використання позиційних ембедингів, що допомагають відповісти на питання відстані між елементами сесії [11][17][18][19].

BERT4Rec також використовує механізм уваги, але на відміну від SASRec є двонаправленою архітектурою і не є ідеальним вибором для задач побудови рекомендацій, оскільки в NLP BERT найкраще вирішує задачу Masked Language Model (для цього і потрібна двонаправлена архітектура), а у випадку рекомендаційних систем в нас стоїть задача передбачити наступний елемент послідовності, а не відновити пропущений [3].

1.4 Висновки до розділу 1

На сьогоднішній день, питання побудови рекомендаційних систем для більшості цифрових продуктів є досить важливим саме по собі, а для таких сфер, як онлайн-кінотеатри або платформи e-commerce, відіграє одну з ключових ролей в утриманні користувачів та їх монетизації. Побудова якісної рекомендаційної системи є нетривіальною задачею через відсутність прямого сигналу про релевантність рекомендацій та проблему холодного старту як для нових користувачів, так і для нових товарів. Особливістю e-commerce як галузі бізнесу з особливою необхідністю використання рекомендаційних систем є те, що переважна частка трафіку належить анонімним користувачам, ідентифікація яких обмежена cookies та IP-адресою, що унеможливлює побудову довгострокових профілів. Попит є нестабільним у часі та схильним до впливу акційних пропозицій і сезонності.

Також було розглянуто і проаналізовано існуючі підходи до побудови рекомендаційних систем, як на основі послідовностей взаємодій користувачів, так і більш класичні, такі як колаборативна та контентна фільтрації. Більшість з цих систем використовують методи машинного навчання для отримання рекомендацій найвищої якості.

РОЗДІЛ 2 ТЕОРЕТИЧНИЙ ОПИС РОБОТИ ПОСЛІДОВНИХ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

2.1 Постановка задачі дослідження

Основна роль послідовних рекомендаційних систем – передбачення наступних елементів (товарів/фільмів/пісень тощо) на основі історичних даних про взаємодії користувача [3]. Головна перевага таких моделей над класичними – врахування і надання значної ваги можливості зміни поведінкових шаблонів користувача з часом. Це дає можливість надавати більш точні, актуальні та відповідні контексту рекомендації.

В математичних означеннях, послідовність дій одного користувача можна представити як багатовимірну впорядковану множину виду $\left\{i_1^{(1)}, i_2^{(2)}, \dots, i_k^{(t-1)}\right\}$, де $i_k^{(t)}$- елемент, з яким користувач взаємодіяв на кроці t. Таким чином, задача прогнозування наступного елементу $i_m^{(t)}$, де 1 < m, k ≤ $|I|$ зводиться до [20]:

$$i_m^{(t)} = \arg\max_{i \in I} P(i | i_1^{(1)}, i_2^{(2)}, \dots, i_k^{(t-1)}).$$

Рекомендаційні системи можна умовно поділити на 2 класи: з використанням класичних алгоритмів та з використання глибокого навчання. В межах даної дипломної роботи детальніше розглянемо другий клас, а саме рекурентні нейронні мережі, оскільки в силу своєї структури вони найкраще підходять для вирішення задач обробки, аналізу та прогнозування числових та текстових послідовностей, побудови рекомендацій на основі послідовностей подій. В даній роботі будуть оглянуті такі моделі рекурентних нейронних мереж як LSTM (Long-Short Term Memory) [12], GRU (Gated Recurrent Units) [13] та SASRec (Self-Attentive Sequential Recommendation) [17].

2.2 Методи попередньої обробки даних

Першим етапом при побудові будь-якої статистичної моделі або моделі машинного навчання є збір навчального набору даних (датасету) та приведення його до формату, що підходить для навчання моделі. У випадку числових змінних зазвичай не потребуються додаткові методи обробки, інколи прибираються викиди, тобто екстремальні значення, що можуть спотворити дані. Для цього використовуються такі методи, як метод міжквартильного розмаху і метод «трьох сигм», у випадку, якщо дані розподілені нормально.

Метод міжквартильного розмаху (IQR) визначає точки даних викидами у випадку, якщо вони виходять за межі інтервалу $[Q1 - 1.5 \times IQR, Q3 + +1.5 \times IQR]$, де $IQR = Q3 - Q1$, Q3 – 75-тий персентиль і Q1 – 25-тий персентиль.

Метод трьох сигм визначає точки даних викидами, якщо вони виходять за межі інтервалу $[\bar{x} - 3\sigma, \bar{x} + 3\sigma]$, де $\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$, $\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})}{n-1}}$

Для обробки текстових даних використовуються ембедінги [11]. Ембедінги – це відображення з множини індексів товарів у щільний вектор (тобто той, де більшість значень ненульові дійсні числа). Для елемента і це можна представити як $\boldsymbol{e}_i = \boldsymbol{E}[i] \in \mathbb{R}^d$, де $E \in \mathbb{R}^{|V| \times d}$ – матриця ембедінгів, що навчається, а |V| - розмір словника. Для задач NLP і побудови рекомендаційних систем не підходять методи на кшталт one-hot encoding, при якому $\boldsymbol{x} \in \{0, 1\}^{|V|}$, адже це призвело б до зберігання в пам'яті і операцій з матрицею розміру $|V| \times d$.

При приведенні послідовності подій до виду, придатного для навчання моделі, використовуємо d = 64 для ембедінгу товарів та d = 16 – для категорій. Ці значення були отримані емпіричним шляхом. Так, фінальний вектор, що йде на вхід моделі для її навчання та інференсу, має вигляд $\boldsymbol{x}_t = = \left[\boldsymbol{e}_t^{item} || \boldsymbol{e}_t^{cat} || \log(1 + \Delta t)\right] \cdot w_t$, де $||$ позначає конкатенацію, $\boldsymbol{e}_t^{item} \in \mathbb{R}^{64}$,

$\boldsymbol{e}_t^{cat} \in \mathbb{R}^{16}, \Delta t$ – часовий проміжок між подіями, а w_t – вага події. Про це детальніше в наступному розділі, але на зараз слід зазначити, що у різних видів взаємодії користувача з товаром мають бути різні ваги, адже, наприклад, покупка товару має значно сильніший сигнал, ніж його перегляд. Логарифмування часової шкали необхідне для того, щоб зменшити розмах значень у цьому вимірі і дати можливість моделі оперувати часовими інтервалами в контексті «мало-багато» замість безпосередніх значень.

Під час навчання застосовується Dropout – підхід, при якому випадкові елементи, в нашому випадку, ембедінгу обнуляються:

$$\widetilde{\boldsymbol{e}_t} = Dropout(\boldsymbol{e}_t, p) = \frac{1}{1-p} \cdot \boldsymbol{e}_t \odot \boldsymbol{m}, \boldsymbol{m}_j \sim Bernoulli(1-p),$$

де p – ймовірність відкидання, а множник $\frac{1}{1-p}$ зберігає очікувану амплітуду;

$\odot -$ поелементне множення [11]

2.3 Розбір архітектур основних моделей послідовних рекомендаційних систем

Тепер розберемо архітектуру LSTM:

Як вже згадувалося, LSTM – це різновид рекурентної нейронної мережі, яка на кожному з кроків містить в собі 3 гейти: забування, вхідний та вихідний [12].

Схематично архітектуру LSTM можна зобразити так:

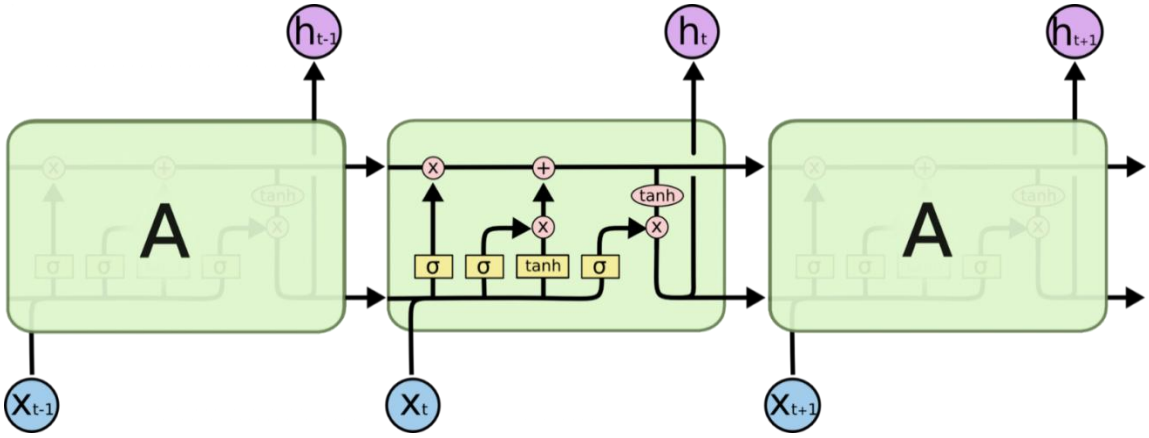


Рис. 2.1 Схематична архітектура LSTM [12]

Формула гейту забування [12]:

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}||\boldsymbol{x}_t] + \boldsymbol{b}_f),$$

де $\boldsymbol{W}_f -$ матриця ваг повнозв'язного шару нейронної мережі;

$\boldsymbol{h}_{t-1} -$ попередній прихований стан;

$\boldsymbol{x}_t$ – поточний вхідний елемент;

$\boldsymbol{b}_f$ – зсув повнозв'язного шару;

$\sigma -$ логістична функція (сигмоїда), $\sigma = \frac{1}{1+e^{-x}}$.

Формула вхідного гейту:

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}||x_t] + \boldsymbol{b}_i),$$

де $\boldsymbol{W}_i -$ матриця ваг повнозв'язного шару нейронної мережі;

$\boldsymbol{h}_{t-1} -$ попередній прихований стан;

$\boldsymbol{x}_t$ – поточний вхідний елемент;

$\boldsymbol{b}_i$ – зсув повнозв'язного шару.

Кандидат стану комірки:

$$\widetilde{\boldsymbol{c}_t} = \tanh(\boldsymbol{W}_c[\boldsymbol{h}_{t-1}||\boldsymbol{x}_t] + \boldsymbol{b}_c),$$

де $\boldsymbol{W}_c -$ матриця ваг повнозв'язного шару нейронної мережі;

$\boldsymbol{h}_{t-1} -$ попередній прихований стан;

$\boldsymbol{x}_t$ – поточний вхідний елемент;

$\boldsymbol{b}_c$ – зсув повнозв'язного шару.

Вихідний вектор значень гейту забування поелементно множиться на попередній стан комірки і сумується з поелементним добутком вихідного вектора вхідного гейту та кандидата стану комірки, так оновлюється стан комірки:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \widetilde{\boldsymbol{c}_t}$$

Формула вихідного гейту:

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o[\boldsymbol{h}_{t-1}||x_t] + \boldsymbol{b}_o),$$

де $\boldsymbol{W}_o -$ матриця ваг повнозв'язного шару нейронної мережі;

$\boldsymbol{h}_{t-1} -$ попередній прихований стан;

$\boldsymbol{x}_t$ – поточний вхідний елемент;

$\boldsymbol{b}_o$ – зсув повнозв'язного шару.

Формула оновлення прихованого стану:

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$

Щодо архітектури GRU – вона схожа до LSTM, але має певні суттєві відмінності, які слід окремо підсвітити. Зокрема, це відсутність окремого стану комірки і наявність двох гейтів замість трьох в LSTM [13]:

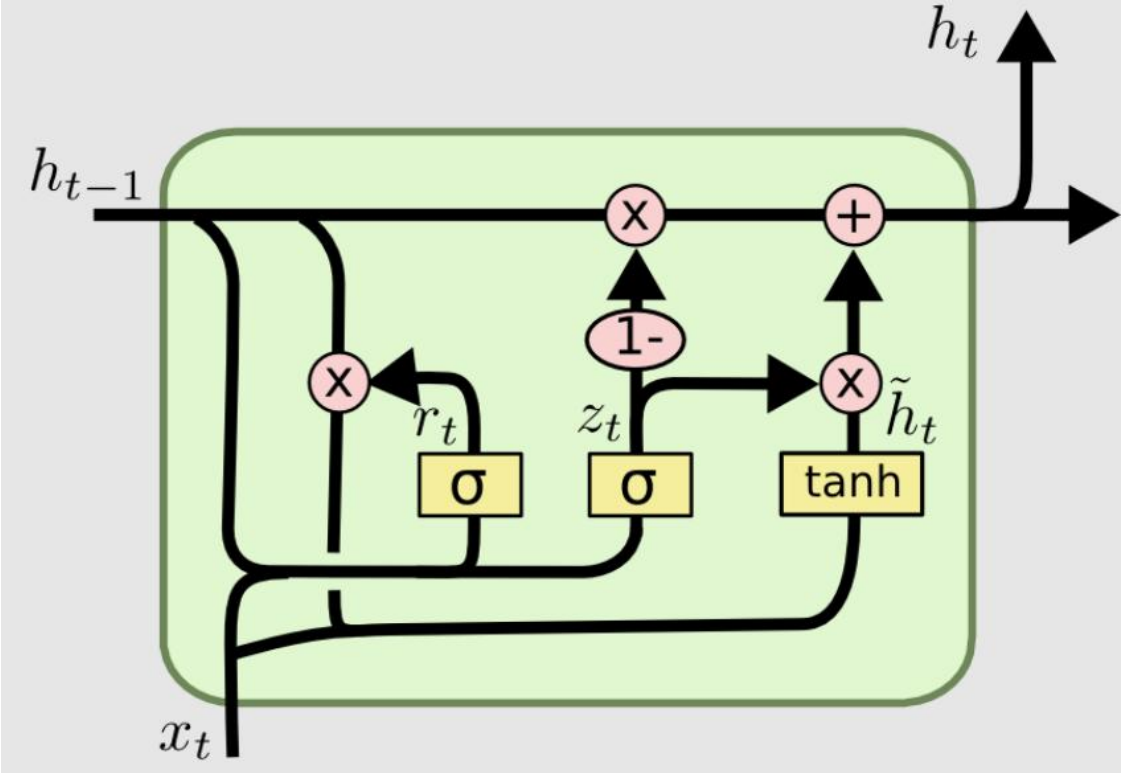


Рис. 2.2 Схематична архітектура GRU [21]

Формула гейту скидання [13][15]:

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r[\boldsymbol{h}_{t-1}||\boldsymbol{x}_t] + \boldsymbol{b}_r)$$

Формула гейту оновлення [13][15]:

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z[\boldsymbol{h}_{t-1}||\boldsymbol{x}_t] + \boldsymbol{b}_z).$$

Кандидат прихованого стану [13][15]:

$$\widetilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_h[(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1})||\boldsymbol{x}_t] + \boldsymbol{b}_h).$$

Оновлення прихованого стану [13][15]:

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \widetilde{\boldsymbol{h}_t}.$$

Стосовно SASRec – головні елементи архітектури цієї мережі – це позиційний ембедінг та механізм уваги, зупинимося на них детальніше [17].

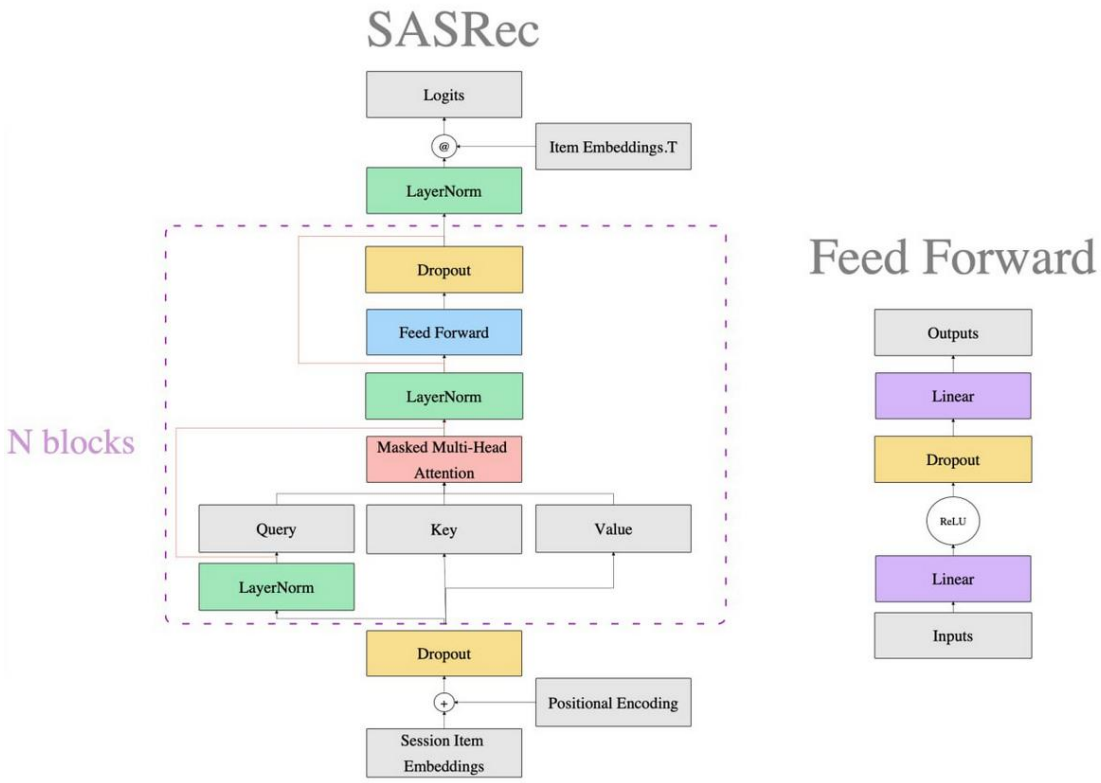


Рис. 2.3 Схематична архітектура SASRec [22]

Перш за все, позиційний ембедінг влаштований таким чином:

$$\boldsymbol{s}_t = \boldsymbol{W}_{proj}\boldsymbol{x}_t + \boldsymbol{P}[t].$$

Тут $\boldsymbol{W}_{proj} \in \mathbb{R}^{d_h \times 81}$, в конкретному випадку 64-вимірного ембедінгу для товарів та 16-вимірного для категорій + часова ознака, а $\boldsymbol{P} \in \mathbb{R}^{T_{max} \times d_h}$ – позиційний ембедінг, що навчається.

Масштабована скалярна увага (Scaled Dot-Product Attention):

$$Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}} + \boldsymbol{M}_{causal}\right)\boldsymbol{V},$$

де $d_k = d_n / n_{heads}$,

Q - тензор запитів, **K** – тензор ключів, **V** – тензор значень;

$\boldsymbol{M}_{causal}[i,j] = \begin{cases} 0, \text{якщо } j \le i \\ -\infty, \text{якщо } j > i \end{cases}$[15].

Каузальна маска являє собою верхньотрикутну матрицю, що забезпечує неможливість моделі враховувати майбутні токени (або у випадку рекомендаційних систем, товари) при навчанні.

Багатоголова увага (Multi-Head Attention):

$$MultiHead(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = [head_1 || \dots || head_h]\boldsymbol{W}^O,$$

де $head_i = Attention\left(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V\right)$;

$\boldsymbol{W}_i^Q, \boldsymbol{W}_i^K, \boldsymbol{W}_i^V \in \mathbb{R}^{d_h \times d_k}$ – відображення для кожної і-ої голови;

d_k – розмірність кожної голови окремо;

$\boldsymbol{W}^O \in \mathbb{R}^{d_h \times d_h}$ – вихідне відображення.

На основі багатоголової уваги базується архітектура трансформер-енкодера, який також використовується в моделі [15]:

$$\boldsymbol{z}_t = \boldsymbol{s}_t + MultiHead\left(LN(\boldsymbol{s}_t), LN(\boldsymbol{s}_t), LN(\boldsymbol{s}_t)\right).$$

На вхід багатоголової уваги йде один і той же нормалізований шар і для запитів, і для ключів, і для пар, звідси і походить назва такого механізму, як self-attention [17].

$$\boldsymbol{o}_t = \boldsymbol{z}_t + FFN\left(LN(\boldsymbol{z}_t)\right),$$

де $FFN(\boldsymbol{x}) = GELU(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$;

$\boldsymbol{W}_1 \in \mathbb{R}^{d_h \times 4d_h}, \boldsymbol{W}_2 \in \mathbb{R}^{4d_h \times d_h}$ у випадку використання 4 голів;

$GELU(x) = x \cdot \Phi(x)$;

$\Phi(x)$ – функція Лапласа [11]

2.4 Функції втрат та навчання моделі

Функції втрат відіграють ключову роль в процесі навчання будь-якої моделі машинного навчання. Їх основна задача – кількісна оцінка помилки між прогнозами моделі та реальними даними. Суть машинного навчання заснована на зменшенні значення функції втрат.

Зважена перехресна ентропія (CrossEntropy) [11]:

Для логітів моделі $\boldsymbol{z} \in \mathbb{R}^{|V|}$ та індексу цільового елемента y формула перехресної ентропії визначається так:

$$\mathcal{L}_{CE} = -\log \frac{\exp(z_y)}{\sum_{j=1}^{|V|} \exp(z_j)}.$$

Після розрахунку перехресної ентропії, вона зважується по зразках за типом події:

$$\mathcal{L} = \sum_{i=1}^{B} w_i \cdot \mathcal{L}_{CE}^{(i)},$$

де B – розмір батчу,

w_i – вага цільової події для зразка i.

Для навчання моделі також необхідно визначити оптимізатор. В нашому випадку це Adam [11][23][24]. Він визначається такими формулами:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Корекція зміщення:

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t}, \widehat{v_t} = \frac{v_t}{1-\beta_2^t}$$

Оновлення параметрів відбувається за наступною формулою:

$$\theta_t = \theta_{t-1} - \eta \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon},$$

де θ – значення параметра на цьому кроці;

η – швидкість навчання;

$\epsilon = 10^{-8}$ – довільне мале число для запобігання ділення на нуль.

Для затухання ваг з ціллю запобігання перенавчанню, використовується L2-регуляризація [11]. Вона додає штраф до функції втрат, що не дає градієнтам «вибухати»:

$$\mathcal{L}_{reg} = \mathcal{L} + \frac{\lambda}{2}||\theta||^2,$$

де λ – коефіцієнт регуляризації.

Також застосовується косинусний відпал швидкості навчання [24]. Його основна мета та ціль – забезпечити більш плавне навчання моделі на пізніх епохах, поступово зменшуючи швидкість навчання для точнішої збіжності:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\frac{t\pi}{T}\right),$$

де η_{min} – встановлена мінімальна допустима швидкість навчання;

T - загальна кількість епох.

Також при навчанні моделей можна додати обрізку градієнтів [11]:

$$\boldsymbol{g} \leftarrow \boldsymbol{g} \cdot \frac{c}{\max\left(c, \left|\left|\boldsymbol{g}\right|\right|_2\right)},$$

де c – певне порогове значення для обрізки, зазвичай $c = 1$.

2.5 Метрики оцінювання якості роботи моделі

Як вже було зазначено раніше, в контексті рекомендаційних систем ми не маємо явних сигналів, що вказують на релевантність запропонованого користувачеві товару, тому для задач ранжування використовуються інші метрики, ніж в задачах класифікації, де мітки класів чіткі та для навчальних даних відомі [3].

HR@K (Hit Rate):

$$HR@K = \sum_{i=1}^{N} 1[y_i \in TopK_i]$$

Метрика Hit rate відповідає на запитання «Яка частка користувачів отримала правильну рекомендацію серед перших K позицій?» [3]. Так, якщо HR@10 = 0.3, то це означає, що для 30% тестових сесій модель включила релевантний наступний товар у свої топ-10 рекомендацій. У випадку K = 1, ця метрика є бінарною – чи влучила модель з правильною відповіддю, чи ні. Дана метрика не враховує позицію, на якій знаходиться правильний елемент.

Precision@K:

$$Precision@K = \frac{1}{N} \sum_{i=1}^{N} \frac{1[y_i \in TopK_i]}{K}$$

Дана метрика відповідає на питання «Яка частина рекомендованих елементів є релевантною?». При прогнозуванні одного наступного елемента $Precision@K = \frac{HR@K}{K}$. Ця метрика є більш інформативною у випадку прогнозування кількох наступних елементів.

Recall@K (у випадку одного наступного елемента):

$$Recall@K = \frac{1}{N}\sum_{i=1}^{N}\frac{|\{y_i\} \cap TopK_i|}{|\{y_i\}|} = HR@K$$

Ця метрика відповідає на запитання «Яку частку серед всіх релевантних елементів модель знайшла серед K рекомендацій?». У випадку прогнозування одного наступного елемента Recall@K стає тотожним HR@K.

MRR@K:

$$MRR@K = \frac{1}{N}\sum_{i=1}^{N}\begin{cases}\frac{1}{rank_i(y_i)}, \text{якщо } rank_i(y_i) \le K \\ 0, \text{якщо } rank_i(y_i) > K\end{cases}$$

Ця метрика відповідає на питання, наскільки високо модель ранжує правильний елемент [3]. На відміну від бінарної природи метрики HR@K, MRR@K також враховує позицію влучення.

Введемо:

$$AP@K = Precision@r,$$

якщо прогнозується один елемент.

Ця метрика логічно веде до MAP (Mean Average Precision), ідея якої полягає в порівнянні середнього значення Precision по всіх зразках. При

передбаченні одного елемента, Average Precision зводиться до $\frac{1}{rank}$ для кожного із запропонованих, що є еквівалентним формулі MRR@K.

NDCG@K

$$NDCG@K = \frac{1}{N}\sum_{i=1}^{N}\begin{cases}\frac{1}{\log_2(rank_i(y_i)+1)}, \text{якщо } rank_i(y_i) \le K \\ 0, \text{якщо } rank_i(y_i) > K\end{cases}$$

Ця метрика відповідає на питання про якість ранжування елементів, враховуючи логарифмічний штраф за нижчу позицію релевантного елементу [3].

2.6 Висновки до розділу 2

В даному розділі було розглянуто математичні основи існуючих методів рекомендації на основі аналізу послідовностей взаємодій користувачів з товарами в контексті рекомендаційних систем для галузі e-commerce. Було розглянуто основні методи обробки сирих даних про взаємодії користувачів з товарами та перетворення їх в готові для обробки моделями послідовності. Було запропоновано до побудови та порівняння моделі LSTM4Rec, GRU4Rec та SASRec. Також було описано та проаналізовано метрики порівняння якості моделей, а також обґрунтовано їх вибір для подальшого застосування.

РОЗДІЛ 3 ПРОЕКТУВАННЯ, РЕАЛІЗАЦІЯ І ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

3.1 Архітектура системи

Перш за все, необхідно визначити архітектуру готової рекомендаційної системи. Нехай існує фронтенд-складова, певний клієнтський інтерфейс, що емулює взаємодії користувача з товарами. Подалі будемо називати її лендинг сторінкою. При натисканні кнопки «Згенерувати рекомендації» в клієнтському інтерфейсі на сервер, на якому розгорнута модель, мусить надсилатися HTTP-запит, що містить в своєму тілі дані про всі взаємодії користувача, тип бажаної моделі для генерації рекомендацій та бажану кількість рекомендацій. Сервер, на якому завантажена модель із заздалегідь збереженими параметрами після навчання, приводить тіло запиту до формату, придатного для обробки моделлю. Відбувається інференс моделі, отримуємо результати – і сервер формує відповідь, в якій міститься назва типу моделі, кількість використаних івентів (взаємодій) та самі рекомендації. Після отримання статусу 200 OK, клієнтський скрипт отримує відповідь від серверу у форматі JSON, яка оброблюється і відображається користувачу як HTML-елементи.

3.2 Обґрунтування вибору технологій

Вибір технологій буде таким

1. Для роботи з моделями (їх навчання та інференс) буде використана мова Python і бібліотека PyTorch [24].

2. Для обробки вхідних даних моделі на етапі навчання використана мова Python і бібліотека pandas.

3. Для клієнтського застосунку (лендингу) використано мову верстки HTML, мову стилів CSS та мову програмування JavaScript.

4. Веб-застосунок (сервер) побудований мовою Python на основі FastAPI, розгорнутий разом із збереженими вагами моделей, файлами лендингу (index.html та style.css) за допомогою Docker, сервер запускається за допомогою uvicorn.

Зобразимо схематичну діаграму архітектури системи:

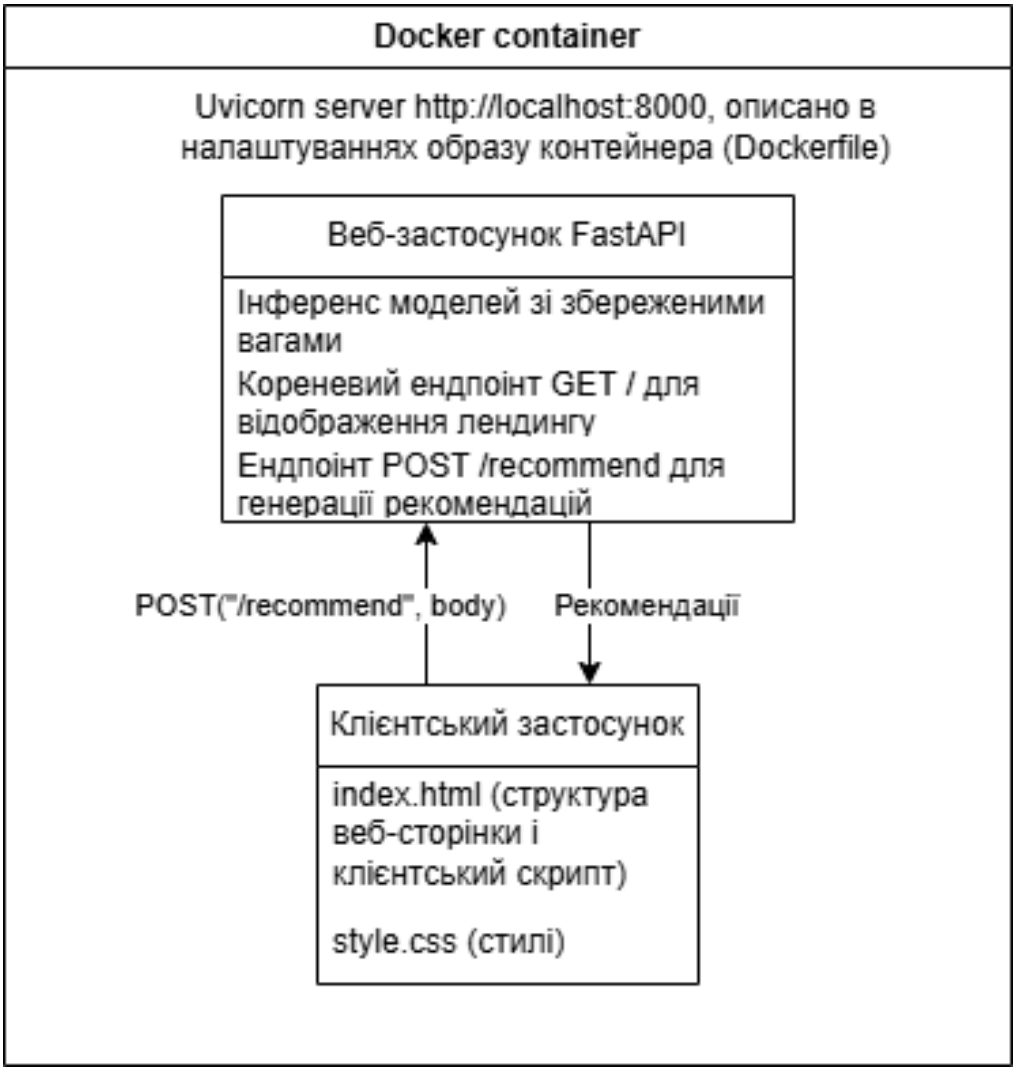


Рис. 3.1 Архітектурна діаграма системи (додано приклад POST-запиту для генерації рекомендацій)

В якості даних для навчання моделі оберемо відкритий датасет від Retailrocket – Retailrocket recommender system dataset [25]. Він складається з 4

файлів: events.csv, category_tree.csv, item_properties_part1.csv та item_properties_part2.csv

Табл. 3.1 Структура файлу events.csv

Назва колонки	Тип даних	Опис колонки
timestamp	int64	Момент часу, в який сталася взаємодія в UNIX форматі (кількість мілісекунд з 01.01.1970)
visitorid	int	Унікальний ідентифікатор користувача
event	string	Один із трьох можливих типів взаємодій: view (перегляд), addtocart (додання в кошик), transaction (покупка)
itemid	int	Унікальний ідентифікатор товару
transactionid	int	Унікальний ідентифікатор транзакції (ненульове значення тільки для event = transaction)

Табл. 3.2 Структура файлу category_tree.csv

Назва колонки	Тип даних	Опис колонки
categoryid	int	Унікальний ідентифікатор категорії
parentid	int	Ідентифікатор батьківської категорії. Якщо батьківська категорія відсутня – нульове значення.

Табл. 3.3 Структура файлів item_properties_part1.csv та item_properties_part2.csv

Назва колонки	Тип даних	Опис колонки
timestamp	int64	Момент часу вивантаження даних про властивості товарів в UNIX форматі
itemid	int	Унікальний ідентифікатор товару
property	string	Визначає властивості товару, також з цього поля виявляється categoryid товару
value	string	Значення властивості товару, для property = categoryid означає відповідно categoryid товару

3.3 Побудова, навчання та експорт ваг моделей

Спершу необхідно провести EDA (Exploratory Data Analysis) та визначити, які ознаки можемо сформувати для остаточного навчання моделі. Очевидно, в першу чергу це event. Потрібно також відзначити, що їх сигнали відрізняються за важливістю для формування рекомендацій. Позначимо для view значення ваг 1, для addtocart – 5 та для transaction – 10 як для найсильнішого сигналу [4]:

EVENT_WEIGHTS = {"view": 1.0, "addtocart": 5.0, "transaction": 10.0}.

Для подальшого використання часової ознаки, потрібно трансформувати UNIX timestamp формат в різницю в секундах від «поточного моменту», яким визначимо останню дату івенту в наборі даних. Надалі отриману часову шкалу логарифмуємо для остаточної побудови навчальних та тестувальних послідовностей.

Для більш стабільного навчання моделей бажано прибрати рідкісні товари, які майже не користуються попитом. Якщо з товаром було хоча б 5 взаємодій, можемо враховувати його при навчанні моделей. Будуємо словник товарів, нумеруючи їх від 1. Нумерація детермінована за рахунок сортування ідентифікаторів товарів за зростанням. Номер 0 залишимо для паддінгу (відступу).

Потім тривіальним чином визначаємо категорію товару, якщо така є. Аналогічно до товарів, будуємо словник категорій. Можемо перейти до побудови фінальних послідовностей. Групуючи по visitorid, формуємо список кортежів з товарами, взаємодій з ними, вагами взаємодій і різницями в часі між послідовними взаємодіями для кожного користувача. Прибираємо користувачів, чиї послідовності коротше за 3 взаємодії, такі дані не підійдуть для навчання через їх недостатність. Для користувачів з довгими послідовностями взаємодій залишаємо MAX_SEQ останніх. MAX_SEQ – це настроюваний гіперпараметр на етапі навчання, в контексті цієї роботи він дорівнює 50. В результаті всіх препроцесингів отримуємо оброблений датасет, який можна ділити на навчальну та тестову вибірки:

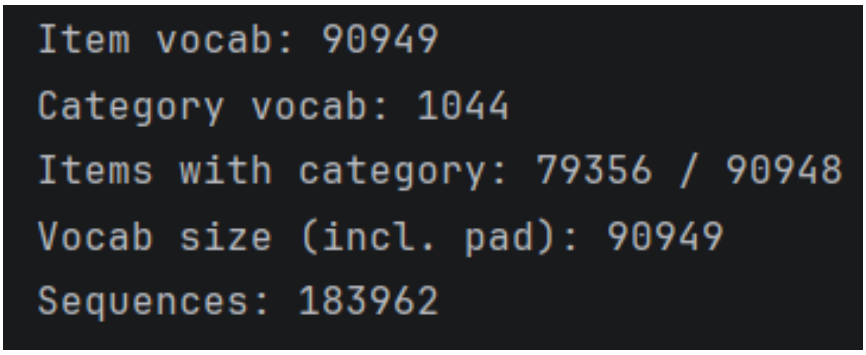


Рис. 3.2 Розміри словників товарів, категорій, кількість товарів з ненульовими категоріями та загальна кількість послідовностей

```
[(55828, 718, 1.0, 0.0),
 (69710, 155, 1.0, 0.04726444496896534),
 (12957, 203, 1.0, 0.04377333297835455)]
```

Рис. 3.3 Приклад послідовності взаємодій для одного користувача

Семантика послідовності на рис. 7 – користувач переглянув три товари: 55828, 69710 та 12957 з відповідними категоріями 718, 155 та 203. Логарифмування часової шкали на цьому етапі ще не відбувається.

Поділ на навчальну та тестову вибірки відбувається в пропорції 80/20. Тестова вибірка являє собою останні 20% послідовності початкового обробленого датасету, так визначаємо цільові товари.

Потім відбувається безпосередньо побудова моделей. Вони представлені як 3 класи: LSTMRec, GRURec та SASRec. Всі класи наслідують клас PyTorch nn.Module, для моделей пишемо власні конструктори класів та перевантажуємо метод forward(). Конструктори класів LSTMRec та GRURec приймають на вхід позиційні аргументи num_items, num_cats та ключові аргументи item_embed_dim (розмірність ембедингу для товарів, стандартно 64), cat_embed_dim (розмірність ембедингу для категорій, стандартно 16), hidden_dim (розмірність прихованого шару), num_layers (кількість шарів LSTM/GRU відповідно) а також dropout та embed_dropout для визначення ймовірності обнулення параметрів для LSTM/GRU та ембедингів відповідно. В перевантаженому методі forward(), який приймає на вхід розпаковані кортежі зі списку взаємодій користувачів відбувається логарифмування часової ознаки. Інша частина методу forward – це прямий прохід по шару LSTM/GRU. Конструктор SASRec приймає на вхід ті ж самі аргументи, додатково з n_heads, що відповідає за кількість голів уваги та max_len, що відповідає за максимальну довжину вхідної послідовності. Суть роботи SASRec описана в попередньому розділі. Зовнішня функція compute_loss

використовується для розрахунку функції втрат моделей, як і згадувалося раніше, використовується зважена крос-ентропія.

Навчання відбувається на GPU за допомогою CUDA, в нашому випадку в ролі GPU виступає відеокарта NVIDIA GeForce RTX 5070 [24]. При старті навчання ініціалізуємо оптимізатор (Adam з швидкістю навчання 10^{-3} та коефіцієнтом регуляризації 10^{-5}) та планувальник навчання (CosineAnnealingLR з кількістю епох 10 та $\eta_{min} = 10^{-5}$).

```
================================================
  Training LSTM
================================================
  Epoch 1/10  loss=11.6352  lr=0.000976
  Epoch 2/10  loss=9.3576  lr=0.000905
  Epoch 3/10  loss=8.8663  lr=0.000796
  Epoch 4/10  loss=8.5455  lr=0.000658
  Epoch 5/10  loss=8.2872  lr=0.000505
  Epoch 6/10  loss=8.0582  lr=0.000352
  Epoch 7/10  loss=7.8813  lr=0.000214
  Epoch 8/10  loss=7.7196  lr=0.000105
  Epoch 9/10  loss=7.6454  lr=0.000034
  Epoch 10/10  loss=7.5915  lr=0.000010
```

Рис. 3.4 Процес навчання моделі LSTMRec

```
================================================
  Training GRU
================================================
  Epoch 1/10  loss=10.8594  lr=0.000976
  Epoch 2/10  loss=9.3382  lr=0.000905
  Epoch 3/10  loss=8.9672  lr=0.000796
  Epoch 4/10  loss=8.6948  lr=0.000658
  Epoch 5/10  loss=8.4333  lr=0.000505
  Epoch 6/10  loss=8.2009  lr=0.000352
  Epoch 7/10  loss=8.0289  lr=0.000214
  Epoch 8/10  loss=7.8993  lr=0.000105
  Epoch 9/10  loss=7.8178  lr=0.000034
  Epoch 10/10  loss=7.7619  lr=0.000010
```

Рис. 3.5 Процес навчання моделі GRURec

```
Epoch 1/10  loss=12.1536  lr=0.000976
Epoch 2/10  loss=9.7176  lr=0.000905
Epoch 3/10  loss=9.3550  lr=0.000796
Epoch 4/10  loss=9.0800  lr=0.000658
Epoch 5/10  loss=8.8380  lr=0.000505
Epoch 6/10  loss=8.5864  lr=0.000352
Epoch 7/10  loss=8.3645  lr=0.000214
Epoch 8/10  loss=8.1755  lr=0.000105
Epoch 9/10  loss=8.0680  lr=0.000034
Epoch 10/10  loss=7.9812  lr=0.000010
```

Рис. 3.6 Процес навчання моделі SASRec

Розрахуємо метрики, які показують моделі на тестовій вибірці після навчання:

Табл. 3.4 Порівняння побудованих моделей (5 рекомендацій)

Модель/метрика	HR@5	Precision@5	MAP@5	NDCG@5
LSTM	0.3384	0.0677	0.2977	0.308
GRU	0.3351	0.067	0.2951	0.3053
SASRec	0.3074	0.0615	0.2689	0.2787

Табл. 3.5 Порівняння побудованих моделей (10 рекомендацій)

Модель/метрика	HR@10	Precision@10	MAP@10	NDCG@10
LSTM	0.355	0.0355	0.3	0.3134
GRU	0.35	0.035	0.2972	0.3101
SASRec	0.3236	0.0324	0.2711	0.2839

Табл. 3.6 Порівняння побудованих моделей (20 рекомендацій)

Модель/метрика	HR@20	Precision@20	MAP@20	NDCG@20
LSTM	0.3691	0.0185	0.3009	0.317
GRU	0.3648	0.0182	0.2982	0.3139
SASRec	0.3382	0.0169	0.2721	0.2876

При порівнянні моделей на 5, 10 та 20 рекомендацій, можемо побачити, що значення метрик трохи кращі у LSTM, в порівнянні з GRU, що не є дивним, адже їх архітектури досить схожі. При цьому, SASRec показує трохи гірші результати.

Після навчання та валідації моделей, можемо зберегти значення їх ваг за допомогою метода torch.save() у форматі .pt для зручності подальшого інференсу на сервері.

3.4 Проєктування та реалізація програмного інтерфейсу

Маючи збережені ваги моделей, можемо побудувати веб-застосунок на базі FastAPI, розгорнути його за допомогою Docker та запустити за допомогою uvicorn. Для можливості інференсу моделей, в файлі app.py імпортуємо всі бібліотеки, що були використані для побудови моделей та ініціалізуємо їх. Також імпортуємо бібліотеки FastAPI та pydantic для побудови веб-застосунку. Напишемо функції prepare_sequence(), яка конвертує вхідні івенти з JSON-формату у вид, готовий для інференсу та get_recommendations(), яка й виконує інференс на моделях, ваги яких були попередньо завантажені з відповідних .pt файлів за допомогою методу load_state_dict(). Девайсом інференсу цього разу виступить CPU.

Тепер можемо ініціалізувати безпосередньо сам FastAPI веб-застосунок. Після цього монтуємо статичні файли (файл стилів style.css) та оголошуємо класи сутностей, з якими будемо працювати. Це EventItem, RecommendRequest, RecommendItem та RecommendResponse. Всі ці класи наслідують pydantic.BaseModel і використовуються для валідації коректності вхідних HTTP-запитів. Тепер можна сформувати ендпоінти, на які клієнтським скриптом будуть відправлятися HTTP-запити. Найбільший інтерес мають GET /, за яким користувач отримує саму HTML-сторінка зі стилями та POST /recommend, який запускає на сервері інференс моделей та повертає відповідь у вигляді рекомендацій.

Побудова Docker-образу відбувається за допомогою відповідного Dockerfile:

```
FROM python:3.13-slim
LABEL authors="Марченко"

WORKDIR /app
COPY . .
RUN pip install --no-cache-dir -r requirements.txt
RUN useradd -m appuser
USER appuser
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```

Рис. 3.7 Dockerfile проєкту

Його основна роль полягає в створенні віртуальної машини за допомогою Docker на базі ядра Linux, на якому встановлений інтерпретатор Python 3.13 та завантаженні всіх файлів проєкту в робочу директорії цієї машини, інсталяції всіх залежностей (використаних бібліотек) та запуску localhost-серверу на порті 8000 командою uvicorn app:app --host 0.0.0.0 –port 8000. Така машина називається контейнером. IP-адреса 0.0.0.0 використовується оскільки адреса 127.0.0.1 є локальною для самого контейнера, і користувач на пристрої, на якому запущений даний контейнер не зможе в такому разі мати доступ до серверу. Файл requirements.txt генерується командою pip freeze > requirements.txt у вікні терміналу віртуального середовища Python. Фінальний вигляд клієнтського застосунку такий:

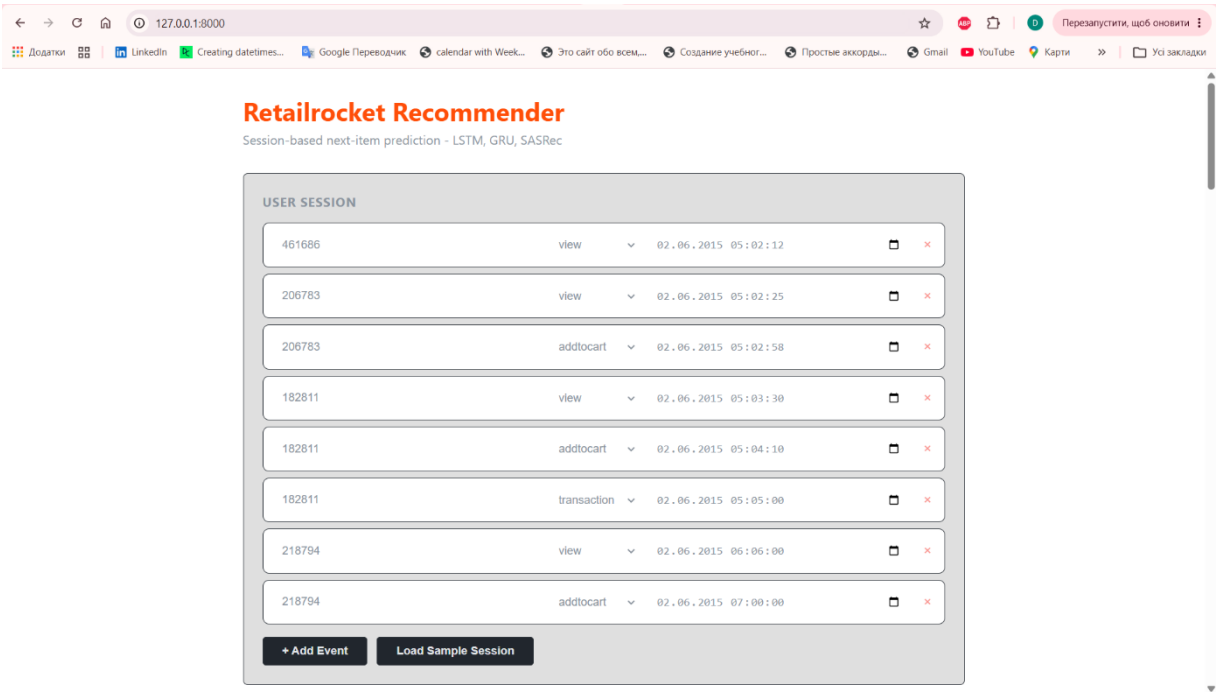


Рис. 3.8 Інтерфейс для введення івентів (емуляція дій одного користувача)

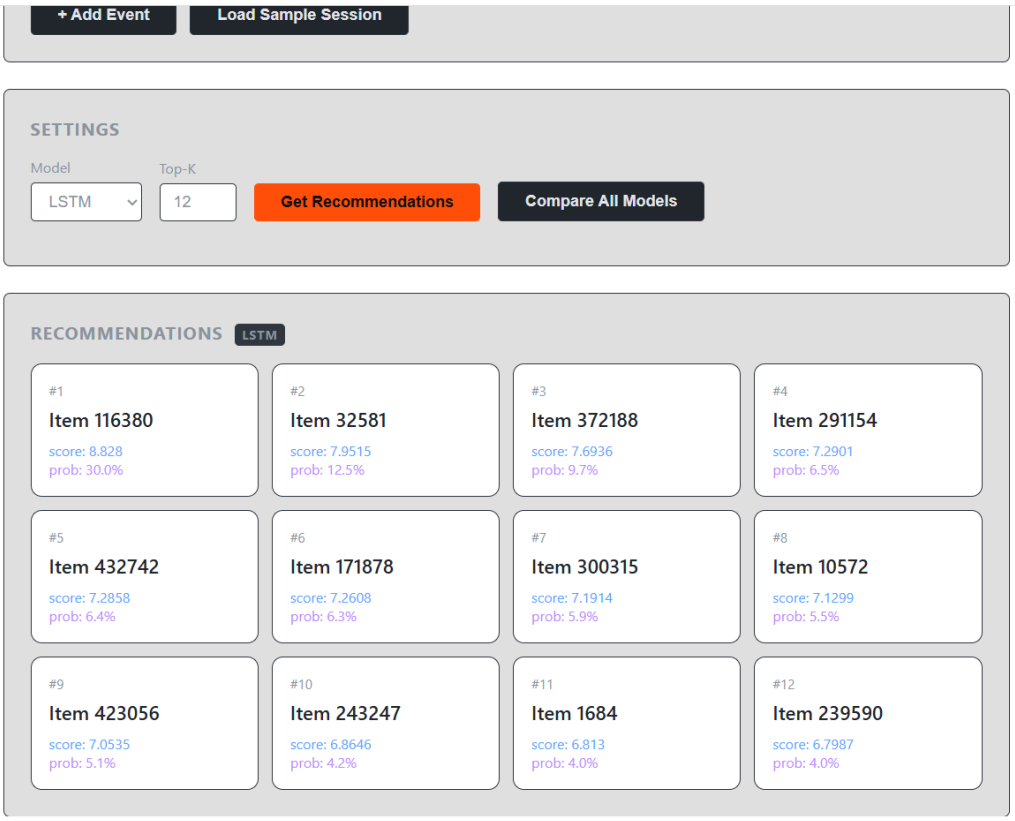


Рис.3.9 Згенеровані за допомогою LSTM рекомендації, їх score та ймовірності

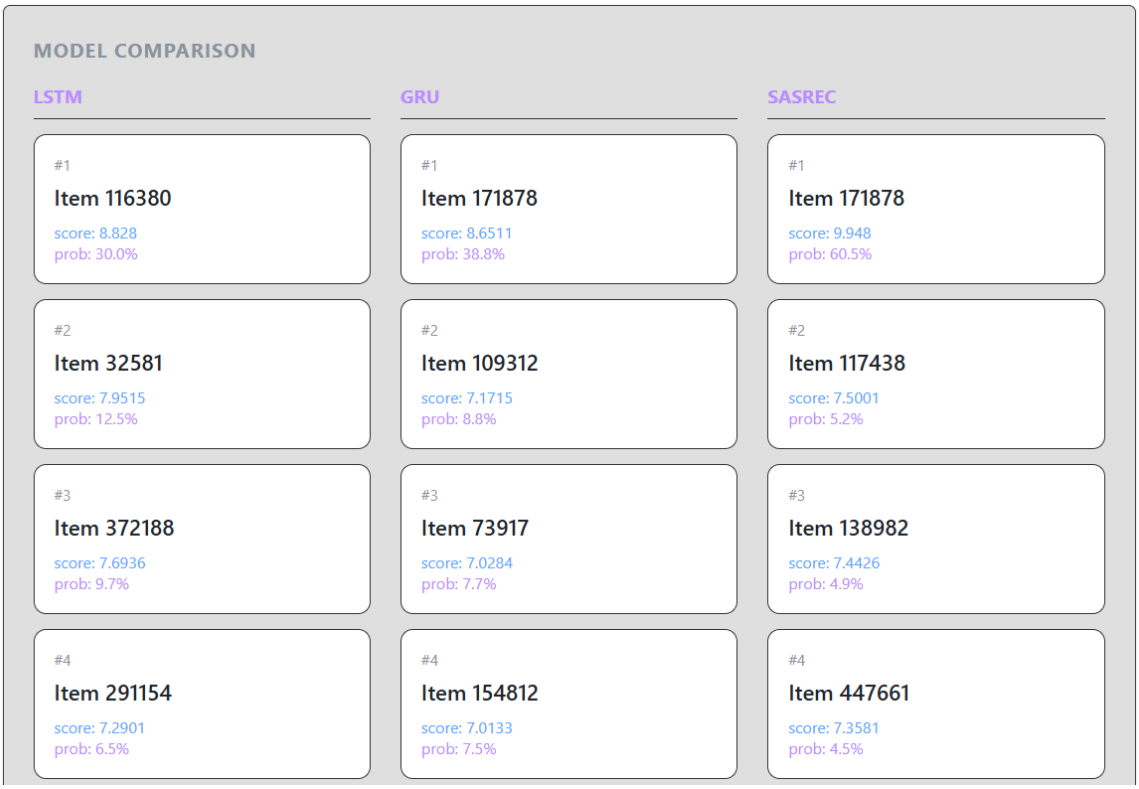


Рис. 3.10 Порівняння моделей і результатів їх видачі

3.5 Порівняння ефективності моделі на іншому наборі даних

Для більшої наочності сформованих рекомендацій використаємо відкритий датасет OpenCDP. Він містить в собі два файли: 2019-Oct.csv та 2019-Nov.csv, що мають ідентичну структуру (табл. 3.7) [26].

Табл. 3.7 Структура файлів 2019-Oct.csv та 2019-Nov.csv

Назва колонки	Тип даних	Опис колонки
event_time	datetime64	Дата й час взаємодії (часова зона UTC)
event_type	string	Один із чотирьох можливих типів взаємодій: view (перегляд), cart (додання в кошик), purchase (покупка), remove_from_cart (видалення з кошику)

Продовження табл. 3.7

product_id	int	Унікальний ідентифікатор товару
category_id	int64	Унікальний ідентифікатор категорії товару
category_code	str	Категорія товару
brand	str	Бренд товару
price	float	Ціна товару у доларах
user_id	int	Унікальний ідентифікатор користувача
user_session	int	Унікальний ідентифікатор користувацької сесії

У порівнянні з попередньо використаним Retailrocket, він набагато менш анонімізований і дає можливість користувачу робити висновки про бренд, категорію та ціну запропонованого товару, анонімізовані лише назви товарів. При цьому даний датасет має значно більший розмір, сумарно в обох файлах >109M рядків, тому для прийнятних часу роботи з датасетом та обсягів використаної пам’яті, використаємо бібліотеку Polars, що добре підходить для обробки великих обсягів даних у Python [27].

Безпосередня логіка обробки вхідних ідентична до такої у випадку попереднього датасету. Основна відмінність полягає в появі нових ознак (category_code, brand, price). Ціна, аналогічно до часової ознаки, логарифмується. З category_code будемо брати лише першу частину, певну «загальну» категорію. Бренд враховується в тому випадку, якщо є хоча б 10 взаємодій з товарами цього бренду. При побудові моделей залишаємо ті ж параметри, але вхідні розмірності тепер враховують бренд після ембедингу (розмірність 16) та цінову ознаку. Тобто якщо вхідна розмірність для моделі у

випадку попереднього датасету була 81 (64+16+1), тепер вона дорівнює 98 (64+16+16+1+1). Навчання моделей відбувається аналогічно.

Після навчання моделей порівняємо значення їх метрик:

Табл. 3.8 Порівняння побудованих моделей (5 рекомендацій)

Модель/метрика	HR@5	Precision@5	MAP@5	NDCG@5
LSTM	0.4494	0.0899	0.3775	0.3955
GRU	0.4502	0.09	0.3787	0.3965
SASRec	0.4554	0.0911	0.3896	0.4060

Табл. 3.9 Порівняння побудованих моделей (10 рекомендацій)

Модель/метрика	HR@10	Precision@10	MAP@10	NDCG@10
LSTM	0.4999	0.05	0.3843	0.4118
GRU	0.5008	0.0501	0.3855	0.4129
SASRec	0.5045	0.0505	0.3961	0.4219

Табл. 3.10 Порівняння побудованих моделей (20 рекомендацій)

Модель/метрика	HR@20	Precision@20	MAP@20	NDCG@20
LSTM	0.5518	0.0276	0.3879	0.4249
GRU	0.5526	0.0276	0.3891	0.426
SASRec	0.5566	0.0278	0.3998	0.435

Видно, що порівняно з датасетом Retailrocket, моделі, навчені на датасеті OpenCDP показують набагато кращі значення метрик, але при цьому час навчання значно зростає. Для прикладу, на першому датасеті швидкість навчання SASRec впродовж 10 епох становила ~136 секунд, а на другому – близько 3322 секунди. При цьому на даному датасеті SASRec показує себе найкращим чином, хоча на попередньому її метрики були найгіршими.

Веб-застосунок та користувацький застосунок не потребують значних змін. Оскільки тепер є можливість відобразити більше інформації про товар, а саме його категорію, бренд та ціну, додаємо в клас RecommendResponse нові поля: brand типу str, category типу str та price_visual типу float.

Dockerfile та процес деплою готового застосунку змін не зазнає, необхідно лише оновити файл залежностей requirements.txt для врахування бібліотеки Polars.

Користувацький інтерфейс для вводу також не зазнав змін:

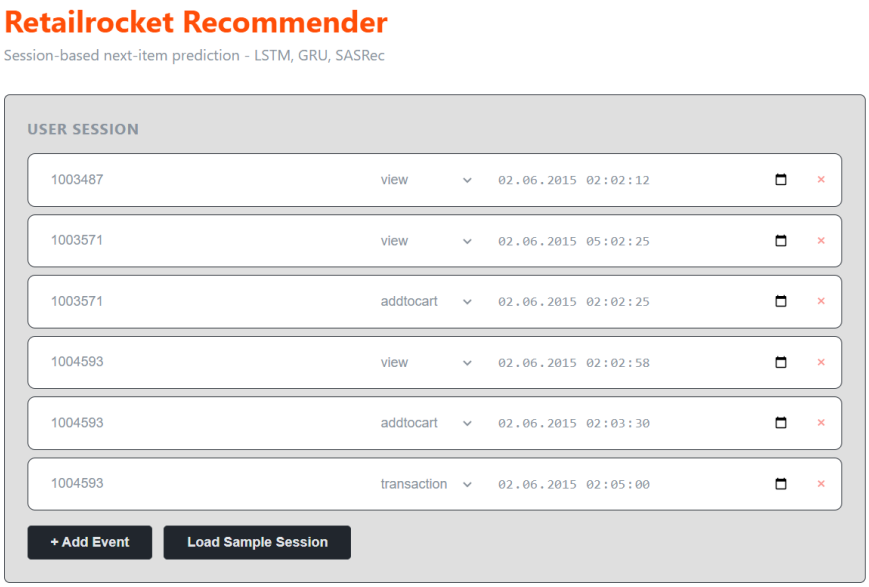


Рис. 3.11 Інтерфейс вводу

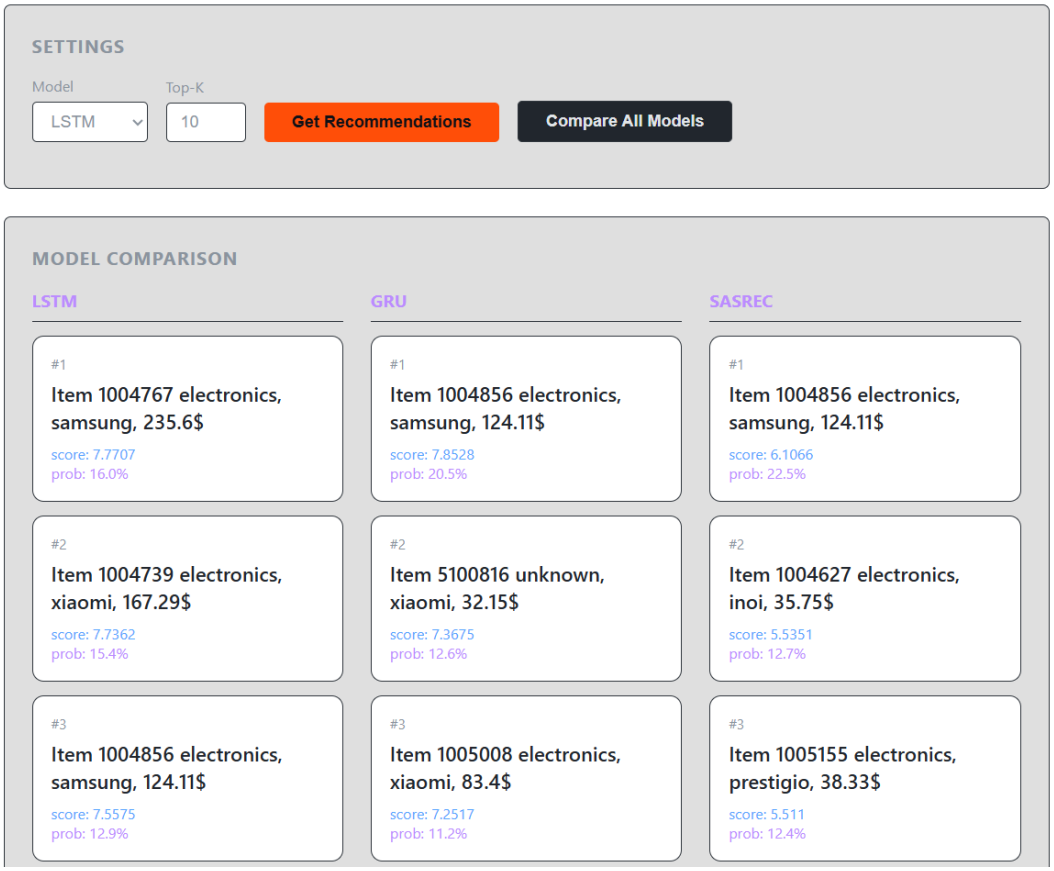


Рис. 3.12 Приклад згенерованих рекомендацій

3.6 Висновок до розділу 3

В даному розділі було створено рекомендаційну систему, що надає змогу формувати персоналізовані рекомендації на базі однієї з трьох моделей на вибір користувача: LSTM4Rec, GRU4Rec або SASRec. Було розглянуто загальну архітектуру системи, прийняті рішення щодо вибору мов та технологій для реалізації продукту, а також деталі формування вхідного датасету, навчальної та тестової вибірок. Проведено порівняння моделей та сформовано висновки про їх ефективність при навчанні на різних наборах даних.

РОЗДІЛ 4 ЕКОНОМІЧНИЙ АНАЛІЗ ТА ОЦІНКА ВАРТОСТІ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться техніко-економічне обґрунтування розробки системи рекомендації товарів з урахуванням інтересів та побажань користувачів інтернет-магазину. Сучасні рекомендаційні системи відіграють одну з ключових ролей у роботі цифрових платформ, оскільки вони безпосередньо впливають на рівень утримання користувачів, тривалість взаємодії з сервісом, якість користувацького досвіду та комерційну ефективність будь-яких платформ, особливо e-commerce. Саме тому при створенні програмного продукту важливим є не лише досягнення високої точності рекомендацій, а й забезпечення економічної доцільності розробки, впровадження та подальшої експлуатації системи.

У дипломній роботі програмний продукт розглядається як складна інтелектуальна система аналізу даних, що поєднує методики глибокого навчання на основі рекурентних нейронних мереж та механізми уваги. Для оцінювання ефективності обраних рішень використовується метод функціонально-вартісного аналізу (ФВА), який дозволяє дослідити функціональні складові системи, визначити їх важливість та співвіднести із витратами на реалізацію.

Застосування ФВА у даній роботі дозволяє: визначити ключові функції рекомендаційної системи; проаналізувати альтернативні варіанти реалізації окремих компонентів; оцінити технічний рівень програмного продукту; мінімізувати витрати на розробку без втрати функціональності; обґрунтувати вибір технологій та програмних інструментів.

У процесі аналізу розглядаються основні компоненти системи: алгоритмічне ядро рекомендаційної системи, механізми обробки даних, методи побудови рекомендацій та засоби оцінювання ефективності моделей.

Після визначення альтернативних варіантів реалізації проводиться порівняння технічних параметрів і розрахунок рівня якості програмного продукту.

4.1 Постановка задачі проєктування

Метою даного етапу є створення програмної системи рекомендації товарів, здатної автоматизувати процес підбору товарів для користувача на основі аналізу історії взаємодій користувачів з товарами на платформі.

Програмний продукт повинен забезпечувати

1. Формування персоналізованих рекомендацій товарів.
2. Обробку великих масивів неявного зворотного зв’язку (Implicit Feedback).
3. Реалізацію декількох алгоритмів рекомендацій для їх подальшого порівняння.
4. Можливість побудови гібридного ансамблю рекомендацій.
5. Аналіз ефективності моделей за допомогою метрик HR@K, Precision@K, MAP@K та NDCG@K.
6. Масштабованість та можливість подальшого розширення функціоналу.

Основними функціональними вимогами до програмного продукту є

1. Алгоритмічна база: реалізація рекурентних нейронних мереж та механізму уваги.
2. Підсистема обробки даних: система повинна підтримувати вміст POST-запитів з даними у JSON-форматі для подальшого інференсу.
3. Аналітична частина: програмний продукт повинен забезпечувати розрахунок HR@K, Precision@K, MAP@K та NDCG@K, порівняння результатів різних моделей, оцінювання ефективності механізму уваги та рекурентних нейронних мереж.

4. Ергономічність та продуктивність: система повинна забезпечувати прийнятний час обчислень, помірне використання оперативної пам’яті, простоту масштабування, можливість подальшої інтеграції у веб-сервіси.

4.2 Обґрунтування функцій програмного продукту

На основі аналізу вимог до системи рекомендації товарів було виділено основні функції, які визначають технічну реалізацію та вартість розробки.

Головна функція F0 – створення програмної системи рекомендацій товарів. На основі цієї функції виділено наступні підфункції

1. F1 – Вибір мови програмування:
    а) Python;
    б) C++ .
2. F2 – Реалізація алгоритмічного ядра:
    а) Використання готових бібліотек обробки даних та машинного навчання (pandas, PyTorch);
    б) Власна реалізація алгоритмів глибокого навчання.
3. F3 – Побудова рекомендаційної моделі:
    а) Класичні методи колаборативної фільтрації;
    б) Підхід з використанням рекурентних нейронних мереж.
4. F4 – Аналіз та візуалізація результатів:
    а) Графічна візуалізація результатів (лендинг-сторінка з демонстрацією рекомендацій на основі взаємодій користувача);
    б) Текстова звітність та CSV-логи.

Для кожної з функцій було обрано альтернативні варіанти реалізації , які можуть бути використані при проектуванні системи. Вони представлені на морфологічній карті.

На основі морфологічної карти проведено аналіз переваг і недоліків кожного варіанту за допомогою позитивно-негативної матриці.

Табл. 4.1 Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F1 – вибір мови програмування	a) Python	Велика кількість підтримуваних бібліотек глибокого навчання, легкість написання коду	Порівняно мала швидкість та споживання значної кількості оперативної пам’яті
	b) C++	Велика швидкість виконання та більші допустимі обсяги даних	Складність реалізації та менша кількість бібліотек глибокого навчання
F2 – реалізація алгоритмічного ядра	а) Використання готових бібліотек обробки даних та машинного навчання (pandas, PyTorch)	Швидкість розробки	Менша інтерпретовність
	б) Власна реалізація алгоритмів	Повний контроль над алгоритмами і	Нераціональна трудомісткість розробки

	глибокого навчання	максимальна інтерпретовність	
F3 – побудова рекомендаційної моделі	а) Класичні методи колаборативної фільтрації	Простота реалізації та інтерпретації	Нижча точність
	б) Підхід з використанням рекурентних нейронних мереж	Вища точність та надійність	Більша складність системи та трудомісткі розрахунки при навчанні
F4 – аналіз та візуалізація результатів	а) Графічна візуалізація результатів	Більша інтерпретовність результатів роботи	Додаткові витрати часу на розробку підсистеми
	б) Текстова звітність та CSV-логи	Простота реалізації	Неінформативна для кінцевого користувача

Для F1 перевагу отримала мова Python, оскільки вона забезпечує легку і відносно швидку реалізацію, завдяки наявності необхідних бібліотек. Вона також добре підходить для наглядного виводу результатів. Дозволяє значно пришвидшити розробку рекомендаційної системи порівняно з низькорівневими мовами. Також, виходимо з припущення, що обсяги трафіку недостатньо великі, щоб спричинити відчутне latency для користувачів при розробці системи мовою Python. При цьому для особливо великих технологічних компаній на кшталт Netflix або Amazon зазвичай використовуються власні рішення, побудовані за допомогою мов C++ або Rust. Тому варто також розглянути цей варіант.

Для F2 було обрано готові бібліотеки, що забезпечить високу надійність алгоритму, та відповідність усім стандартам обчислень та безпеки.

Для F3 підхід з використанням рекурентних нейронних мереж є більш ефективним для надання рекомендацій, тому й був реалізований в межах даної дипломної роботи.

Серед варіантів F4 було обрано графічну візуалізацію за допомогою побудови клієнтського веб-застосунку. Це забезпечить наочність і зрозумілість результатів, ціною невеликої кількості обчислень і нескладних архітектурних рішень.

На основі проведеного аналізу було сформовано два варіанти реалізації системи:

Варіант 1: F1(а) – F2(а) – F3(б) – F4(а).

Варіант 2: F1(б) – F2(а) – F3(б) – F4(а).

Оскільки в дипломній роботі основний акцент зроблено на дослідженні наявних методів побудови рекомендацій на основі послідовностей взаємодій, підтримка високих навантажень не є критичною, то прийнято рішення надати перевагу першому варіанту як найзручнішому з точки зору наукового дослідження.

4.3 Обґрунтування системи параметрів програмного продукту

Для кількісного оцінювання та порівняння обраних варіантів реалізації системи необхідно визначити систему техніко-економічних параметрів. При цьому вони повинні максимально повно відображати специфіку методів рекомендацій, а також враховувати витрати на розробку.

X1 – Точність рекомендацій: характеризує здатність системи формувати релевантні рекомендації користувачам. Одиниця виміру: %. Основні метрики: HR@K, MRR@K та NDCG@K.

X2 – Швидкість обчислень: відображає час, необхідний для навчання моделей та генерації рекомендацій. Одиниця виміру: секунди.

X3 – Витрати оперативної пам’яті: характеризує ресурсомісткість алгоритмів при роботі з великими вхідними наборами даних. Одиниця виміру: МБ.

X4 – Час розробки системи: відображає трудомісткість реалізації програмного продукту. Одиниця виміру: людиногодини.

4.4 Аналіз експертного оцінювання параметрів

Для визначення вагомості параметрів було залучено експертну групу з 7 осіб, до складу якої увійшли фахівці у галузі системного аналізу, машинного навчання та розробки програмного забезпечення.

Кожен експерт незалежно виконав ранжування параметрів за ступенем важливості, результати ранжування продемонстровано в таблиці 4.2.

Табл. 4.2 Результати експертного ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Точність рекомендацій	%	2	1	1	1	2	2	1	10	-7.5	56.25
X2	Швидкість розрахунків	сек	1	2	2	3	1	1	2	12	-5.5	30.25
X3	Витрати пам’яті	Мб	4	4	3	2	3	4	4	24	6.5	42.25
X4	Час розробки	люд-год	3	3	4	4	4	3	3	24	6.5	42.25
	Сума		10	10	10	10	10	10	10	70	0	123

Найважливішими було обрано точність рекомендацій, швидкість розрахунків та витрати пам’яті.

Перевіримо достовірність експертних оцінок:

а) загальна сума рангів:

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

де N – число експертів;

n – кількість параметрів.

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5.$$

в) відхилення суми рангів кожного параметра від середньої:

$$\Delta_i = R_i - T.$$

$$\Delta_1 = 16 - 17{,}5 = -4{,}5;$$

$$\Delta_2 = 14 - 17{,}5 = -4{,}5;$$

$$\Delta_3 = 20 - 17{,}5 = 5{,}5;$$

$$\Delta_4 = 20 - 17{,}5 = 3{,}5;$$

Сума відхилень: $-1.5 - 3{,}5 + 2{,}5 + 2{,}5 = 0$.

г) загальна сума квадратів відхилень:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 171.$$

д) коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 171}{7^2(4^3 - 4)} = 0{,}698 > W_k = 0{,}67.$$

Отримане значення коефіцієнта узгодженості перевищує нормативне, тому результати ранжування вважаються достовірними і придатними для подальших розрахунків. На основі попарного порівняння параметрів були визначені коефіцієнти вагомості:

Ступінь переваги і-го параметра над j-тим визначається числовим значенням a_{ij} та по формулі:

$$a_{ij} = \{1.5 \text{ при } X_i > X_j, 1.0 \text{ при } X_i = X_j, 0.5 \text{ при } X_i < X_j\}.$$

Результати попарного порівняння винесено у таблицю 4.3.

Табл. 4.3. Коефіцієнти вагомості параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
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	<	<	>	>	>	<	<	<	0.5

З отриманих числових оцінок переваги складемо матрицю $A = ||a_{ij}||$ та розрахуємо вагомість $K_{\text{ві}}$:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На

другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i{}'}{\sum_{i=1}^{n} b_i{}'}\ , b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Табл. 4.4 Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітерація		Друга ітерація	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	$b_i{}'$	$K'_{\text{ві}}$
X1	1	1.5	1.5	1.5	5.5	0.3438	21.25	0.3603
X2	0.5	1	1.5	1.5	4.5	0.2812	16.25	0.2706
X3	0.5	0.5	1	0.5	2.5	0.1562	9.25	0.1568
X4	0.5	0.5	1.5	1	3.5	0.2188	12.25	0.2076
Всього:					16	1	59	1

Отримані результати підтверджують, що при проектуванні рекомендаційної системи пріоритетним фактором є якість рекомендацій та швидкодія алгоритмів.

Різниця між першою та другою ітераціями менша за 2%, тому результати другої ітерації задовольняють нашим потребам. Приймемо їх як остаточні.

4.5 Аналіз рівня якості варіантів реалізації функції

Для оцінювання рівня якості кожного варіанту реалізації використовується коефіцієнт технічного рівня:

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j}\, B_{i,j},$$

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

B_i – бальна оцінка і-го параметра.

Бальні оцінки визначаються на основі абсолютних значень параметрів для кожного варіанту.

Варіант 1: точність рекомендацій (значення HR@K) – 36%, час навчання однієї моделі – 140 секунд, витрати пам'яті – 2048 МБ, час розробки – 45 люд-год.

Варіант 2: точність рекомендацій – 40%, час навчання – 80 секунд, витрати пам'яті - >4096 МБ, час розробки – 100 люд-год.

Табл. 4.5 Розрахунок показників рівня якості варіантів реалзіації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F_1	А – готові бібліотеки Python	X1	36	90	0.3603	32.43
		X2	140	96	0.2706	25.98
		X3	2048	79	0.1568	12.39
		X4	45	93	0.2076	19.31
F_2	Б – C++	X1	40	98	0.3603	35.31
		X2	80	78	0.2706	21.11
		X3	4096	75	0.1568	11.76
		X4	100	62	0.2076	12.87

За формулою визначаємо рівень якості кожного варіанту:

$$K_{K1} = 32.43 + 25.98 + 12.39 + 19.31 = 90.11;$$
$$K_{K2} = 35.31 + 21.11 + 11.76 + 12.87 = 81.05$$

Після розрахунку інтегрального коефіцієнта якості видно, що варіант 1 має вищий коефіцієнт рівня якості. Тому для подальшої реалізації оберемо саме його (використання Python та його готових бібліотек).

4.6 Економічний аналіз варіантів розробки програмного продукту

Для визначення вартості розробки рекомендаційної системи необхідно оцінити трудомісткість виконання основних етапів проектування.

Основні етапи розробки

1) аналіз предметної області;

2)проектування архітектури системи;

3) реалізація алгоритмів рекомендацій, навчання моделей;

4) побудова клієнтського застосунку;

5) тестування та порівняння ефективності моделей;

6) підготовка документації.

Загальна трудомісткість обчислюється за формулою:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М},$$

де T_P – базова трудомісткість;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації ($K_{СК} = 1$);

$K_М$ – коефіцієнт рівня мови програмування ($K_М = 0.85$ для Python, 1.3 для C++);

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм ($K_{СТ} = 0.8$);

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення ($K_{СТ.М} = 1$)

Для першого завдання: $T_P = 40$:

$$T_1 = 40 \cdot 1.51 \cdot 1 \cdot 0.85 \cdot 0.8 \cdot 1 = 41.072 \text{ людино-годин}$$

Для другого завдання: $T_P = 60$:

$\mathrm{T}_2 = 60 \cdot 1.31 \cdot 1 \cdot 1.3 \cdot 0.8 \cdot 1 = 81.744$ людино-годин

Загальна трудомісткість для кожного варіанту в людино-годинах:

$\mathrm{T}_\mathrm{I} = 41.072 + 81.744 = 122.816$ людино-годин

$\mathrm{T}_\mathrm{II} = 41.072 + 81.744 + 5.2 = 128.016$ людино-годин

У розробці беруть участь розробник з місячним окладом 20000 грн та науковий керівник з окладом 35000 грн. Середня погодинна ставка, рахується за формулою:

$$\mathrm{СЧ} = \frac{M}{T_m \cdot t} \text{ грн},$$

де M – місячний оклад працівників;

T_m – кількість робочих днів на місяць;

t – кількість робочих годин в день.

$$C_\mathrm{ч} = \frac{20000 + 35000}{2 \cdot 21.1 \cdot 8} = 162.91 \text{ грн/год}$$

Заробітна плата вираховується по даній формулі:

$$\mathrm{ЗП} = \mathrm{C_ч} \cdot \mathrm{T_i} \cdot \mathrm{К_д},$$

де $\mathrm{C_ч}$ – величина погодинної оплати праці програміста;

$\mathrm{T_i}$ – трудомісткість відповідного завдання;

$\mathrm{К_д}$ – норматив, який враховує додаткову заробітну плату.

Заробітна плата з урахуванням додаткової ($\mathrm{К_д} = 1.2$):

I. $С_{ЗП} = 162.91 \cdot 717.04 \cdot 1.2 = 140176.55$ грн.

II. $С_{ЗП} = 162.91 \cdot 758.64 \cdot 1.2 = 148310.13$ грн.

Відрахування на єдиний соціальний внесок (ЄСВ, 22%):

I. $С_{ВІД} = 140176.55 \cdot 0.22 = 30838.84$ грн.

II. $С_{ВІД} = 148310.13 \cdot 0.22 = 32628.23$ грн.

Визначимо витрати на оплату однієї машино-години. ЕОМ обслуговує розробника з окладом 20000 грн, коефіцієнт зайнятості КЗ = 0.2:

$$С_{Г} = 12 \cdot 20000 \cdot 0.2 = 48000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП}(ЕОМ) = 48000 \cdot (1 + 0.2) = 57600 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД}(ЕОМ) = 57600 \cdot 0.22 = 12672 \text{ грн.}$$

Амортизаційні відрахування при нормі амортизації 25% та вартості відеокарти, на якій проводилося навчання моделей, 40000 грн:

$$С_{А} = К_{ТМ} \cdot К_{А} \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 40000 = 11500 \text{ грн,}$$

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

$К_{А}$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 40000 \cdot 0.05 = 2300 \text{ грн},$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний річний фонд часу роботи ПК:

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t_З \cdot К_В = (365 - 104 - 11 - 8) \cdot 8 \cdot 0.85 = \\ = 1645.6 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році;

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

$Д_Р$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

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

Витрати на електроенергію (тариф 13.64 грн/кВТ·год, потужність 0.2 кВТ):

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1645.6 \cdot 0.2 \cdot 0.2 \cdot 13.64 = 897.84 \text{ грн},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$ – коефіцієнтом зайнятості приладу;

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

Накладні витрати:

$$С_Н = Ц_{ПР} \cdot 0.67 = 40000 \cdot 0.67 = 26800 \text{ грн}.$$

Річні експлуатаційні витрати:

$$C_{ЕКС} = C_{ЗП} + C_{ВІД} + C_{А} + C_{Р} + C_{ЕЛ} + C_{Н} =$$
$$= 57600 + 12672 + 11500 + 2300 + 897.84 + 26800 =$$
$$= 111769.84 \text{ грн.}$$

Собівартість однієї машино-години:

$$C_{М-Г} = C_{ЕКС}/T_{ЕФ} = 111769.84/1645.6 = 67.92 \text{ грн/год.}$$

Витрати на машинний час:

$$C_{М} = C_{М-Г} \cdot T,$$
$$\text{I. } C_{М} = 67.92 \cdot 122.816 = 8341.66 \text{ грн.}$$
$$\text{II. } C_{М} = 67.92 \cdot 128.016 = 8694.85 \text{ грн.}$$

Накладні витрати (67% від заробітної плати):

$$C_{Н} = C_{ЗП} \cdot 0.67,$$
$$\text{I. } C_{Н} = 140176.55 \cdot 0.67 = 93918.29 \text{ грн.}$$
$$\text{II. } C_{Н} = 148310.13 \cdot 0.67 = 99367.79 \text{ грн.}$$

Повна вартість розробки ПП:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_{М} + C_{Н},$$
$$\text{I. } C_{ПП} = 140176.55 + 30838.84 + 8341.66 + 93918.29 = 273275.34 \text{ грн.}$$
$$\text{II. } C_{ПП} = 148310.13 + 32628.23 + 8694.85 + 99367.79 = 289001 \text{ грн.}$$

4.7 Висновки до розділу 4

У даному розділі було проведено техніко-економічний аналіз системи рекомендацій музики на основі уподобань користувачів.

У результаті функціонально-вартісного аналізу були визначені основні функції програмного продукту, розглянуті альтернативні варіанти реалізації та виконано їх порівняння за технічними та економічними параметрами.

Було проведено оцінку технічного рівню: розрахунок показників рівня якості показав перевагу Варіанту 1 (K1 = 90,11 проти K2 = 81,05), що зумовлено високою швидкістю розробки та простотою використання стандартних бібліотечних рішень Python.

Економічний розрахунок показав, що повна вартість розробки обраного варіанту при окладах 20 000 грн та 35 000 грн для розробника та керівника відповідно склала 273275.34 грн, що є на 15 725,66 грн менше, ніж вартість розробки системи з нуля.

За критерієм техніко-економічного рівня кращим визначено Варіант 1. Обрана архітектура забезпечує необхідну точність обчислень при прийнятних часових та фінансових затратах. Саме цей варіант (Python, pandas, PyTorch) було прийнято до реалізації у практичній частині дипломного проєкту.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну науково-практичну задачу розробки, реалізації та порівняння ефективності різних моделей рекомендаційних систем на основі аналізу послідовностей кліків. За результатами проведеного комплексу теоретичних та експериментальних досліджень отримано такі основні висновки.

Проведено аналіз сучасного стану та архітектурних особливостей рекомендаційних систем у галузі e-commerce. Встановлено, що задача рекомендації товару в e-commerce зводиться до прогнозування наступного елементу вхідної послідовності взаємодій користувача з товарами, для чого найкраще підходять SRS, тобто Sequential Recommender Systems, до класу яких входять в тому числі LSTM4Rec, GRU4Rec та SASRec.

Формалізовано математичний апарат рекурентних нейронних мереж, таких як LSTM та GRU, а також механізму уваги, що є ключовим елементом SASRec. Також було пояснено принцип роботи окремих механізмів навчання моделей, такі як оптимізатор Adam, косинусний відпал та L2-регуляризація. Пояснено суть та обґрунтовано використання метрик точності HR@K, Precision@K, MAP@K та NDCG@K.

Здійснено програмну реалізацію повної архітектури рекомендаційної системи мовою Python із використанням бібліотек pandas, PyTorch та Polars. Було розроблено програмні модулі для попередньої обробки даних, тобто наборів даних Retailrocket та OpenCDP, а саме реалізовано переведення категорійних змінних в числовий формат, очистку даних та формування послідовностей користувацьких взаємодій на основі сирої таблиці взаємодій. Реалізовано розділення обробленого набору даних на навчальну та тестову вибірки для проведення коректної валідації. Ваги навчених моделей були експортовані для подальшого використання в побудованому користувацькому веб-сервісі.

Було проведено серію експериментів для порівняння ефективності моделей на різних наборах даних за допомогою розрахунку метрик HR@K, Precision@K, MAP@K та NDCG@K для K = 5, K = 10 та K = 20. Сформовано висновки про високу ефективність моделі LSTM4Rec для датасету Retailrocket та SASRec для датасету OpenCDP.

Проведено техніко-економічний аналіз системи рекомендації товарів на основі аналізу послідовностей взаємодій. У результаті функціонально-вартісного аналізу були визначені основні функції програмного продукту, розглянуті альтернативні варіанти реалізації та виконано їх порівняння за технічними та економічними параметрами. Розроблена в межах кваліфікаційної роботи система є найбільш ефективною з точки зору фінансових та часових витрат.

Таким чином, розроблена система генерації рекомендацій підтвердила свою теоретичну та практичну спроможність та може бути використаний для інтеграції в реальні платформи e-commerce з ціллю побудови релевантних персоналізованих рекомендацій товарів для збільшення середнього чеку на користувача, збільшення утримання користувачів та покращення користувацького досвіду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Falk K. Practical Recommender Systems. Manning Publications, 2019. 432 p. URL: https://github.com/TheSuranaverse/Resources/blob/main/Practical%20Recommender%20Systems%20(2019%2C%20Manning%20Publications)%20Kim%20Falk%20-%20-%20libgen.lc.pdf (date of access: 05.06.2026)

2. Ricci F., Werthner H. Introduction to the Special Issue: Recommender Systems. *International Journal of Electronic Commerce*. 2006. Vol. 11, no. 2. P. 5–9. DOI: 10.2753/jec1086-4415110200.

3. Sequential Recommender Systems: Challenges, Progress and Prospects / S. Wang et al. *Twenty-Eighth International Joint Conference on Artificial Intelligence {IJCAI-19}*, Macao, China, 10–16 August 2019. California, 2019. DOI: 10.24963/ijcai.2019/883.

4. Hu Y., Koren Y., Volinsky C. Collaborative Filtering for Implicit Feedback Datasets. *2008 Eighth IEEE International Conference on Data Mining (ICDM)*, Pisa, Italy, 15–19 December 2008. 2008. DOI: 10.1109/icdm.2008.22.

5. Aggarwal C. C. Recommender Systems: The Textbook. Springer, 2018. 519 p.

6. Melville P., Sindhwani V. Recommender Systems. *Encyclopedia of Machine Learning and Data Mining*. Boston, MA, 2017. P. 1056–1066. DOI: 10.1007/978-1-4899-7687-1_964.

7. Nedashkovskaya N., Androsov D. Multicriteria evaluation of recommender systems using fuzzy analytic hierarchy process, *Research Square*, 2023, 17 p., DOI: 10.21203/rs.3.rs-3228086/v1.

8. Артеменко Є. В., Недашківська Н. І. Рекомендаційні системи з використанням текстів оглядів користувачів та ансамблювання на основі стекінгу. Системні науки та інформатика: збірник доповідей IV Всеукр. наук.-практ. конф. «Системні науки та інформатика», 01–05 грудня 2025 року, Київ [Електронне видання]. – К., НН ІПСА КПІ ім. Ігоря Сікорського, 2025. – С. 76-

80. URL: http://mmsa.kpi.ua/sites/default/files/systemni_nauky_ta_informatyka_2025.pdf (дата звернення: 05.06.2026)

9. Бичков Д. В., Недашківська Н. І. Розробка рекомендаційної системи на основі моделі GraphSAGE графових нейронних мереж. Системні науки та інформатика: збірник доповідей IV Всеукр. наук.-практ. конф. «Системні науки та інформатика», 01–05 грудня 2025 року, Київ [Електронне видання]. – К., НН ІПСА КПІ ім. Ігоря Сікорського, 2025. – С. 20-27. URL: http://mmsa.kpi.ua/sites/default/files/systemni_nauky_ta_informatyka_2025.pdf (дата звернення: 05.06.2026)

10. Кравченко О.В., Недашківська Н.І. Рекомендаційна система на основі оглядів користувачів і продуктів // Системні науки та інформатика: збірник доповідей I Всеукраїнської науково-практичної конференції «Системні науки та інформатика», 22–29 листопада 2022 року, Київ. – К., НН ІПСА КПІ ім. Ігоря Сікорського, 2022., стор. 404 – 409. URL: http://mmsa.kpi.ua/conferences/2290 (дата звернення: 05.06.2026)

11. Zhang A., Lipton Z. C. Dive into Deep Learning. Cambridge University Press, 2023. URL: https://d2l.ai (date of access: 05.06.2026)

12. Understanding LSTM Networks. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (date of access: 05.06.2026)

13. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation / K. Cho et al. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar. Stroudsburg, PA, USA, 2014. DOI: 10.3115/v1/d14-1179.

14. Hidasi B., Karatzoglou A. Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. *CIKM '18: The 27th ACM International Conference on Information and Knowledge Management*, Torino Italy. New York, NY, USA, 2018. DOI: 10.1145/3269206.3271761.

15. Bahdanau et al. Neural Machine Translation by Jointly Learning to Align and Translate" URL: https://arxiv.org/abs/1409.0473 (date of access: 05.06.2026).

16. Androsov D., Nedashkovskaya N. The hybrid sequential recommender system synthesis method based on attention mechanism with automatic knowledge graph construction // *System Research and Information Technologies*. 2025, No.1, p. 7 – 18. DOI: 10.20535/SRIT.2308-8893.2025.1.01.

17. Kang W.-C., McAuley J. Self-Attentive Sequential Recommendation. *2018 IEEE International Conference on Data Mining (ICDM)*, Singapore, 17–20 November 2018. 2018. DOI: 10.1109/icdm.2018.00035.

18. Androsov D., Nedashkovskaya N. Simple, fast and scalable recommendation systems via external knowledge distillation // *Radio Electronics, Computer Science, Control*. 2025. № 3, p. 126 – 137, DOI: 10.15588/1607-3274-2025-3-12.

19. Nedashkovskaya N., Androsov D. System approach to multicriteria evaluation of session-based and sequential recommendation systems // *KPI Science News*. 2025, № 4, p. 46 – 54. DOI: 10.20535/kpisn.2025.4.343329.

20. Androsov D., Nedashkovskaya N. Simple, fast and scalable recommendation systems via external knowledge distillation // *Radio Electronics, Computer Science, Control*. 2025. № 3, p. 126 – 137, DOI: 10.15588/1607-3274-2025-3-12.

21. Xiong W. A time-series model of GRUs based on attention mechanism for short-term load forecasting. *IEEE Access*. 2024. P. 1. DOI: 10.1109/access.2024.3443876.

22. RecSys Transformer Models Tutorial. *RecTools documentation*. URL: https://rectools.readthedocs.io/en/stable/examples/tutorials/transformers_tutorial.html (date of access: 05.06.2026).

23. PyTorch: An Imperative Style, High-Performance Deep Learning Library URL: https://arxiv.org/pdf/1912.01703. (date of access: 05.06.2026).

24. Недашківська Н.І. Методи і моделі інтелектуального аналізу даних: Практикум [Електронний ресурс]: навч. посіб. для студ. спеціальності 122

«Комп'ютерні науки», освітньої програми «Системи і методи штучного інтелекту» / Н. І. Недашківська; КПІ ім. Ігоря Сікорського. – Електронні текстові дані. – Київ : КПІ ім. Ігоря Сікорського, 2019. – 70 с. URL: https://ela.kpi.ua/handle/123456789/53764 (дата звернення: 05.06.2026).

25. Retailrocket recommender system dataset. URL: https://www.kaggle.com/datasets/retailrocket/ecommerce-dataset (date of access: 05.06.2026)

26. OpenCDP eCommerce behavior data. URL: https://www.kaggle.com/datasets/mkechinov/ecommerce-behavior-data-from-multi-category-store (date of access: 05.06.2026)

27. Polars documentation. URL: https://docs.pola.rs/ (date of access: 05.06.2026)

ДОДАТОК А. Лістинг програми

Лістинг файлу models.py:

```
import itertools

import polars as pl
import numpy as np
import pandas as pd

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence, pack_padded_sequence, pad_packed_sequence

EVENT_WEIGHTS = {"view": 1.0, "addtocart": 5.0, "transaction": 10.0}
TIME_DECAY_HALF_LIFE_DAYS = 90

df = pd.read_csv("events.csv")

df = df[df["event"].isin(EVENT_WEIGHTS.keys())].copy()
df["weight"] = df["event"].map(EVENT_WEIGHTS)
df.sort_values(["visitorid", "timestamp"], inplace=True)

df["ts_seconds"] = df["timestamp"] / 1000.0
max_ts = df["ts_seconds"].max()
df["days_ago"] = (max_ts - df["ts_seconds"]) / 86400.0



w_min = df["weight"].min()
w_max = df["weight"].max()
df["weight"] = 1.0 + (df["weight"] - w_min) / (w_max - w_min) * 9.0
```

```
item_counts = df["itemid"].value_counts()
valid_items = set(item_counts[item_counts >= 5].index)
df = df[df["itemid"].isin(valid_items)]

# build item vocabulary  (0 = padding)
unique_items = sorted(df["itemid"].unique())
item2idx = {int(item): idx + 1 for idx, item in enumerate(unique_items)}
num_items = len(item2idx) + 1  # +1 for padding idx 0

df["item_idx"] = df["itemid"].map(item2idx)

props1 = pd.read_csv("item_properties_part1.csv")
props2 = pd.read_csv("item_properties_part2.csv")
props = pd.concat([props1, props2], ignore_index=True)
cat_rows = props[props["property"] == "categoryid"][["itemid",
"value"]].drop_duplicates("itemid")
cat_rows.columns = ["itemid", "categoryid"]
cat_rows["categoryid"] = pd.to_numeric(cat_rows["categoryid"], errors="coerce")
cat_rows = cat_rows.dropna(subset=["categoryid"])
cat_rows["categoryid"] = cat_rows["categoryid"].astype(int)

cat_rows = cat_rows[cat_rows["itemid"].isin(item2idx)]

unique_cats = sorted(cat_rows["categoryid"].unique())
cat2idx = {cat: idx + 1 for idx, cat in enumerate(unique_cats)}
num_cats = len(unique_cats) + 1

item_to_cat_idx  = {}
for _, row in cat_rows.iterrows():
    if row["itemid"] in item2idx:
        item_to_cat_idx[item2idx[row["itemid"]]] = cat2idx.get(int(row["categoryid"]), 0)

df["cat_idx"] = df["item_idx"].map(item_to_cat_idx).fillna(0).astype(int)

print(f"Item vocab: {num_items}")
```

```
print(f"Category vocab: {num_cats}")
print(f"Items with category: {len(item_to_cat_idx)} / {num_items - 1}")

def build_sequence(g):
    items = g["item_idx"].tolist()
    cats = g["cat_idx"].tolist()
    weights = g["weight"].tolist()
    ts = g["ts_seconds"].tolist()
    deltas = [0.0] + [(ts[i] - ts[i-1]) / 3600.0 for i in range(1, len(ts))]
    return list(zip(items, cats, weights, deltas))

grouped = (
    df.groupby("visitorid")
    .apply(build_sequence)
    .reset_index()
)
grouped.columns = ["visitorid", "sequence"]

grouped = grouped[grouped["sequence"].apply(len) >= 3]

MAX_SEQ = 50
grouped["sequence"] = grouped["sequence"].apply(lambda s: s[-MAX_SEQ:])

print(f"Vocab size (incl. pad): {num_items}")
print(f"Sequences: {len(grouped)}")

class SessionDataset(Dataset):
    def __init__(self, sequences):
        self.input_items = []
        self.input_cats = []
        self.input_weights = []
        self.input_deltas = []
        self.targets = []
        self.target_weights = []
```

```
        for seq in sequences:
            items, cats, weights, deltas = zip(*seq)
            self.input_items.append(torch.tensor(items[:-1], dtype=torch.long))
            self.input_cats.append(torch.tensor(cats[:-1], dtype=torch.long))
            self.input_weights.append(torch.tensor(weights[:-1], dtype=torch.float))
            self.input_deltas.append(torch.tensor(deltas[:-1], dtype=torch.float))
            self.targets.append(items[-1])
            self.target_weights.append(weights[-1])

        self.targets = torch.tensor(self.targets, dtype=torch.long)
        self.target_weights = torch.tensor(self.target_weights, dtype=torch.float)

    def __len__(self):
        return len(self.targets)

    def __getitem__(self, idx):
        return (
            self.input_items[idx],
            self.input_cats[idx],
            self.input_weights[idx],
            self.input_deltas[idx],
            self.targets[idx],
            self.target_weights[idx],
        )

def collate_fn(batch):
    items, cats, weights, deltas, targets, target_w = zip(*batch)
    lengths = torch.tensor([len(x) for x in items])
    padded_items = pad_sequence(items, batch_first=True, padding_value=0)
    padded_cats = pad_sequence(cats, batch_first=True, padding_value=0)
    padded_weights = pad_sequence(weights, batch_first=True, padding_value=0.0)
    padded_deltas = pad_sequence(deltas, batch_first=True, padding_value=0.0)
    return padded_items, padded_cats, padded_weights, padded_deltas, lengths,
torch.stack(targets), torch.stack(target_w)
```

```
seqs = grouped["sequence"].tolist()
np.random.seed(42)
np.random.shuffle(seqs)
split = int(0.8 * len(seqs))

train_ds = SessionDataset(seqs[:split])
test_ds = SessionDataset(seqs[split:])

train_loader = DataLoader(train_ds, batch_size=256, shuffle=True, collate_fn=collate_fn)
test_loader = DataLoader(test_ds, batch_size=512, shuffle=False, collate_fn=collate_fn)

class Attention(nn.Module):
    def __init__(self, hidden_dim):
        super().__init__()
        self.W1 = nn.Linear(hidden_dim, hidden_dim, bias=False)
        self.W2 = nn.Linear(hidden_dim, hidden_dim, bias=False)
        self.v = nn.Linear(hidden_dim, 1, bias=False)

    def forward(self, hiddens, last_hidden, mask):
        energy = self.v(torch.tanh(self.W1(hiddens) + self.W2(last_hidden.unsqueeze(1))))
        energy = energy.squeeze(-1)                # (B, T)
        energy = energy.masked_fill(~mask, -1e9)
        weights = F.softmax(energy, dim=1)         # (B, T)
        context = (hiddens * weights.unsqueeze(-1)).sum(dim=1)  # (B, H)
        return context, weights

class LSTMRec(nn.Module):
    def __init__(self, num_items, num_cats, item_embed_dim=64, cat_embed_dim=16,
hidden_dim=128, num_layers=2, dropout=0.3, embed_dropout=0.2, use_attention=False):
        super().__init__()
        self.item_embedding = nn.Embedding(num_items, item_embed_dim,
padding_idx=0)
        self.cat_embedding = nn.Embedding(num_cats, cat_embed_dim, padding_idx=0)
        self.embed_drop = nn.Dropout(embed_dropout)
```

```
        self.use_attention = use_attention

        lstm_input_dim = item_embed_dim + cat_embed_dim + 1

        self.lstm = nn.LSTM(
            lstm_input_dim, hidden_dim,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout
        )

        if use_attention:
            self.attention = Attention(hidden_dim)
            fc_input_dim = hidden_dim * 2
        else:
            fc_input_dim = hidden_dim

        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(fc_input_dim, item_embed_dim)

        self.output_bias = nn.Parameter(torch.zeros(num_items))

    def forward(self, items, cats, weights, deltas, lengths):
        item_emb = self.embed_drop(self.item_embedding(items))
        cat_emb = self.embed_drop(self.cat_embedding(cats))
        combined = torch.cat((item_emb, cat_emb), dim=-1)
        combined *= weights.unsqueeze(-1)

        time_feat = torch.log1p(deltas).unsqueeze(-1)
        lstm_input = torch.cat([combined, time_feat], dim=-1)

        packed = nn.utils.rnn.pack_padded_sequence(
            lstm_input, lengths.cpu(), batch_first=True, enforce_sorted=False
        )
        rnn_out, (h_n, _) = self.lstm(packed)
```

```
        last_hidden = h_n[-1]

        if self.use_attention:
            rnn_out, _ = pad_packed_sequence(rnn_out, batch_first=True)
            B, T = items.size()
            mask = torch.arange(T, device=items.device).unsqueeze(0) < lengths.unsqueeze(1).to(items.device)
            context, _ = self.attention(rnn_out, last_hidden, mask)
            merged = torch.cat([context, last_hidden], dim=-1)
        else:
            merged = last_hidden

        merged = self.dropout(merged)
        proj = self.fc(merged)

        logits = torch.matmul(proj, self.item_embedding.weight.t())
        logits += self.output_bias
        return logits

class GRURec(nn.Module):
    def __init__(self, num_items, num_cats,
                 item_embed_dim=64, cat_embed_dim=16,
                 hidden_dim=128, num_layers=2, dropout=0.3, embed_dropout=0.2, use_attention=False):
        super().__init__()
        self.item_embed_dim = item_embed_dim
        self.item_embedding = nn.Embedding(num_items, item_embed_dim, padding_idx=0)
        self.cat_embedding = nn.Embedding(num_cats, cat_embed_dim, padding_idx=0)
        self.use_attention = use_attention
        self.embed_drop = nn.Dropout(embed_dropout)
        self.hidden_dim = hidden_dim

        input_dim = item_embed_dim + cat_embed_dim + 1
```

```
        self.gru = nn.GRU(
            input_dim, hidden_dim,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout,
        )

        if use_attention:
            self.attention = Attention(hidden_dim)
            fc_input_dim = hidden_dim * 2
        else:
            fc_input_dim = hidden_dim

        self.layer_norm = nn.LayerNorm(hidden_dim)
        self.dropout = nn.Dropout(dropout)

        self.fc = nn.Linear(fc_input_dim, item_embed_dim)
        self.output_bias = nn.Parameter(torch.zeros(num_items))

    def forward(self, items, cats, weights, deltas, lengths):
        item_emb = self.embed_drop(self.item_embedding(items))
        cat_emb = self.embed_drop(self.cat_embedding(cats))
        combined = torch.cat([item_emb, cat_emb], dim=-1)
        combined = combined * weights.unsqueeze(-1)

        time_feat = torch.log1p(deltas).unsqueeze(-1)
        raw_input = torch.cat([combined, time_feat], dim=-1)

        packed = pack_padded_sequence(raw_input, lengths.cpu(), batch_first=True,
enforce_sorted=False)
        rnn_out, h_n = self.gru(packed)
        rnn_out, _ = nn.utils.rnn.pad_packed_sequence(rnn_out, batch_first=True)

        rnn_out = self.layer_norm(rnn_out)
```

```
        last_hidden = h_n[-1]

        if self.use_attention:
            T_out = rnn_out.size(1)
            mask = torch.arange(T_out, device=items.device).unsqueeze(0) < lengths.unsqueeze(1).to(items.device)
            context, _ = self.attention(rnn_out, last_hidden, mask)
            merged = torch.cat([context, last_hidden], dim=-1)
        else:
            merged = last_hidden

        merged = self.dropout(merged)
        proj = self.fc(merged)

        logits = F.linear(proj, self.item_embedding.weight, self.output_bias)
        return logits

class SASRec(nn.Module):
    def __init__(self, num_items, num_cats,
                 item_embed_dim=64, cat_embed_dim=16,
                 hidden_dim=128, n_heads=4, n_layers=2,
                 dropout=0.3, embed_dropout=0.2, max_len=50):
        super().__init__()
        self.item_embed_dim = item_embed_dim
        self.hidden_dim = hidden_dim
        self.max_len = max_len

        self.item_embedding = nn.Embedding(num_items, item_embed_dim, padding_idx=0)
        self.cat_embedding = nn.Embedding(num_cats, cat_embed_dim, padding_idx=0)
        self.embed_drop = nn.Dropout(embed_dropout)

        input_dim = item_embed_dim + cat_embed_dim + 1
        self.input_proj = nn.Linear(input_dim, hidden_dim)
```

```
        self.pos_embedding = nn.Embedding(max_len, hidden_dim)

        encoder_layer = nn.TransformerEncoderLayer(
            d_model=hidden_dim,
            nhead=n_heads,
            dim_feedforward=hidden_dim * 4,
            dropout=dropout,
            activation="gelu",
            batch_first=True,
            norm_first=True,
        )
        self.transformer = nn.TransformerEncoder(encoder_layer, num_layers=n_layers)
        self.layer_norm = nn.LayerNorm(hidden_dim)

        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(hidden_dim, item_embed_dim)
        self.output_bias = nn.Parameter(torch.zeros(num_items))

        print(f"  [SASRec] {n_layers} layers, {n_heads} heads, hidden={hidden_dim}")

    def forward(self, items, cats, weights, deltas, lengths):
        B, T = items.size()

        item_emb = self.embed_drop(self.item_embedding(items))
        cat_emb = self.embed_drop(self.cat_embedding(cats))
        combined = torch.cat([item_emb, cat_emb], dim=-1)
        combined = combined * weights.unsqueeze(-1)
        time_feat = torch.log1p(deltas).unsqueeze(-1)
        features = torch.cat([combined, time_feat], dim=-1)

        hidden = self.input_proj(features)
        positions = torch.arange(T, device=items.device).unsqueeze(0).expand(B, -1)
        hidden = hidden + self.pos_embedding(positions)
        hidden = self.layer_norm(hidden)
```

```
        causal_mask = torch.triu(
            torch.ones(T, T, device=items.device, dtype=torch.bool), diagonal=1
        )
        padding_mask = torch.arange(T, device=items.device).unsqueeze(0) >=
lengths.unsqueeze(1).to(items.device)

        out = self.transformer(
            hidden,
            mask=causal_mask,
            src_key_padding_mask=padding_mask,
        )

        last_idx = (lengths - 1).long().to(items.device)
        last_hidden = out[torch.arange(B, device=items.device), last_idx]

        proj = self.fc(self.dropout(last_hidden))

        logits = torch.matmul(proj, self.item_embedding.weight.t())
        logits = logits + self.output_bias
        return logits


def compute_loss(logits, targets, target_weights):
    per_sample = F.cross_entropy(logits, targets, reduction="none")
    return (per_sample * target_weights).mean()

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

EPOCHS = 10

def train_model(model, name):
    print(f"\n{'='*50}")
    print(f"  Training {name}")
    print(f"{'='*50}")
```

```
    model = model.to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3, weight_decay=1e-5)
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,
T_max=EPOCHS, eta_min=1e-5)

    for epoch in range(1, EPOCHS + 1):
        model.train()
        total_loss = 0
        n_samples = 0
        for batch in train_loader:
            p_items = batch[0].to(device)
            p_cats = batch[1].to(device)
            p_weights = batch[2].to(device)
            p_deltas = batch[3].to(device)
            lengths = batch[4].cpu()
            targets = batch[5].to(device)
            target_w = batch[6].to(device)

            logits = model(p_items, p_cats, p_weights, p_deltas, lengths)
            loss = compute_loss(logits, targets, target_w)

            optimizer.zero_grad()
            loss.backward()
            nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
            optimizer.step()

            total_loss += loss.item() * targets.size(0)
            n_samples += targets.size(0)

        scheduler.step()
        lr = optimizer.param_groups[0]["lr"]
        print(f"  Epoch {epoch}/{EPOCHS}  loss={total_loss/n_samples:.4f}  lr={lr:.6f}")

    return model
```

```
lstm_model = train_model(LSTMRec(num_items, num_cats), "LSTM")
gru_model = train_model(GRURec(num_items, num_cats), "GRU")
sas_model = train_model(SASRec(num_items, num_cats), "SAS + Attention")

def evaluate(model, loader, ks=(5, 10, 20)):
    model.eval()
    hits = {k: 0 for k in ks}
    mrr = {k: 0.0 for k in ks}
    ndcg = {k: 0.0 for k in ks}
    total = 0

    with torch.no_grad():
        for padded_items, padded_cats, padded_weights, padded_deltas, lengths, targets, _ in
loader:
            padded_items = padded_items.to(device)
            padded_cats = padded_cats.to(device)
            padded_weights = padded_weights.to(device)
            padded_deltas = padded_deltas.to(device)
            targets = targets.to(device)

            logits = model(padded_items, padded_cats, padded_weights, padded_deltas,
lengths)

            for k in ks:
                topk = logits.topk(k, dim=1).indices
                match = (topk == targets.unsqueeze(1))

                has_hit = match.any(dim=1)
                hit_rank = match.float().argmax(dim=1) + 1
                hit_rank[~has_hit] = 0

                hits[k] += has_hit.sum().item()

                rr = torch.zeros_like(hit_rank, dtype=torch.float)
                rr[has_hit] = 1.0 / hit_rank[has_hit].float()
```

```
                mrr[k] += rr.sum().item()

                dcg = torch.zeros_like(hit_rank, dtype=torch.float)
                dcg[has_hit] = 1.0 / torch.log2(hit_rank[has_hit].float() + 1)
                ndcg[k] += dcg.sum().item()

            total += targets.size(0)

    print("\n── Test Metrics ──")
    for k in ks:
        hr = hits[k] / total
        print(f"  @{k:>2}  HR={hr:.4f}  \tPrecision={hr/k:.4f}  \tRecall={hr:.4f}
\tMAP={mrr[k]/total:.4f}  \tMRR@{k:>2} = {mrr[k]/total:.4f}
\tNDCG={ndcg[k]/total:.4f}")


evaluate(lstm_model, test_loader)
evaluate(gru_model, test_loader)
evaluate(sas_model, test_loader)

import pickle

artifacts = {
    "item2idx": item2idx,
    "idx2item": {idx: item for item, idx in item2idx.items()},
    "cat2idx": cat2idx,
    "item_to_cat_idx": item_to_cat_idx,
    "num_items": num_items,
    "num_cats": num_cats,
    "event_weights": EVENT_WEIGHTS,
    "max_seq": MAX_SEQ
}

with open("./artifacts/artifacts.pkl", "wb") as f:
    pickle.dump(artifacts, f)
```

```
print("Saved artifacts")

torch.save(lstm_model.state_dict(), "artifacts/lstm_model.pt")
torch.save(gru_model.state_dict(), "artifacts/gru_model.pt")
torch.save(sas_model.state_dict(), "artifacts/sas_model.pt")

df_1 = pl.read_csv("2019-Oct.csv")
df_2 = pl.read_csv("2019-Nov.csv")
df = pl.concat([df_1, df_2])

EVENT_WEIGHTS = {"view": 1.0, "cart": 5.0, "purchase": 10.0}

df = df.filter(pl.col("event_type").is_in(list(EVENT_WEIGHTS.keys())))
df = df.with_columns(pl.col("event_type").replace_strict(EVENT_WEIGHTS).alias("weight"))

df = df.with_columns(pl.col("event_time").str.to_datetime("%Y-%m-%d %H:%M:%S UTC").dt.epoch("s").cast(pl.Float64).alias("ts_seconds"))

df = df.sort(["user_id", "ts_seconds"])

item_counts = df.group_by("product_id").len()
valid_items = item_counts.filter(pl.col("len") >= 5)["product_id"]
df = df.filter(pl.col("product_id").is_in(valid_items))

unique_items = df["product_id"].unique().sort()
item2idx = {int(item): idx + 1 for idx, item in enumerate(unique_items)}
num_items = len(item2idx) + 1
df = df.with_columns(pl.col("product_id").replace_strict(item2idx, default=0).alias("item_idx"))

df = df.with_columns(pl.col("category_code").fill_null("unknown").str.split(".").list[0].alias("category_top"))
```

```
unique_cats = df["category_code"].unique().sort()
cat2idx = {cat: idx + 1 for idx, cat in enumerate(unique_cats)}
num_cats = len(cat2idx) + 1
df = df.with_columns(pl.col("category_top").replace_strict(cat2idx, default=0).alias("cat_idx"))

df = df.with_columns(pl.col("brand").fill_null("unknown"))
brand_counts = df.group_by("brand").len()
valid_brands = brand_counts.filter(pl.col("len") >= 10)["brand"]
df = df.with_columns(pl.when(pl.col("brand").is_in(valid_brands)).then(pl.col("brand")).otherwise(pl.lit("unknown")).alias("brand_clean"))

unique_brands = df["brand_clean"].unique().sort()
brand2idx = {b: idx + 1 for idx, b in enumerate(unique_brands)}
num_brands = len(brand2idx) + 1
df = df.with_columns(pl.col("brand_clean").replace_strict(brand2idx, default=0).alias("brand_idx"))

df = df.with_columns(pl.col("price").fill_null(0.0).clip(lower_bound=0).alias("price"))
df = df.with_columns(pl.col("price").log1p().alias("log_price"))
price_max = df["log_price"].max()
df = df.with_columns(pl.when(pl.col("log_price") > 0).then(pl.col("log_price") / price_max).otherwise(pl.lit(0.0)).alias("price_norm"))

df = df.with_columns(
    (pl.col("ts_seconds") - pl.col("ts_seconds").shift(1).over("user_id")) \
    .fill_null(0.0)
    .truediv(3600.0)
    .alias("delta_hours")
  )
print(f"Item vocab: {num_items}")
print(f"Category vocab: {num_cats}")
print(f"Brand vocab: {num_brands}")
```

```
print(f"Price range: {df['price'].min():.2f} to {df['price'].max():.2f}")

item_map = (
    df.select(["item_idx", "cat_idx", "category_top", "brand_idx", "brand_clean", "price",
"price_norm"])
    .unique(subset=["item_idx"], keep="last")
)

item_to_cat_idx = dict(item_map.select("item_idx", "cat_idx").iter_rows())
item_to_cat_visual = dict(item_map.select("item_idx", "category_top").iter_rows())
item_to_brand_idx = dict(item_map.select("item_idx", "brand_idx").iter_rows())
item_to_brand_visual = dict(item_map.select("item_idx", "brand_clean").iter_rows())
item_to_price = dict(item_map.select("item_idx", "price_norm").iter_rows())
item_to_price_visual = dict(item_map.select("item_idx", "price").iter_rows())

MAX_SEQ = 50

seq_df = df.group_by('user_id').agg([
    pl.struct("item_idx", "cat_idx", "brand_idx", "price_norm", "weight",
"delta_hours").alias("sequence"),
])

seq_df = (
    seq_df.filter(pl.col("sequence").list.len() >= 3)
    .with_columns(
        pl.col("sequence").list.tail(MAX_SEQ)
    )
)

del df, item_map

class SessionDataset(Dataset):
    def __init__(self, seq_df):
        sequences = seq_df.clone().with_columns([
            pl.col("sequence").list.eval(pl.element().struct.field('item_idx')).alias("items"),
```

```
            pl.col("sequence").list.eval(pl.element().struct.field('cat_idx')).alias("cats"),
            pl.col("sequence").list.eval(pl.element().struct.field('brand_idx')).alias("brands"),
            pl.col("sequence").list.eval(pl.element().struct.field('price_norm')).alias("prices"),
            pl.col("sequence").list.eval(pl.element().struct.field('weight')).alias("weights"),
            pl.col("sequence").list.eval(pl.element().struct.field('delta_hours')).alias("deltas")]
        )

        self.input_items = sequences['items'].list.head(pl.col('items').list.len() - 1)
        self.input_cats = sequences['cats'].list.head(pl.col('cats').list.len() - 1)
        self.input_brands = sequences['brands'].list.head(pl.col('brands').list.len() - 1)
        self.input_prices = sequences['prices'].list.head(pl.col('prices').list.len() - 1)
        self.input_weights = sequences['weights'].list.head(pl.col('weights').list.len() - 1)
        self.input_deltas = sequences['deltas'].list.head(pl.col('deltas').list.len() - 1)
        self.targets = sequences['items'].list.last()
        self.target_weights = sequences['weights'].list.last()

    def __len__(self):
        return len(self.targets)

    def __getitem__(self, idx):
        return (
            torch.tensor(self.input_items[idx],   dtype=torch.long),
            torch.tensor(self.input_cats[idx],    dtype=torch.long),
            torch.tensor(self.input_brands[idx],  dtype=torch.long),
            torch.tensor(self.input_prices[idx],  dtype=torch.float),
            torch.tensor(self.input_weights[idx], dtype=torch.float),
            torch.tensor(self.input_deltas[idx],  dtype=torch.float),
            torch.tensor(self.targets[idx],       dtype=torch.long),
            torch.tensor(self.target_weights[idx],dtype=torch.float)
        )

def collate_fn(batch):
    items, cats, brands, prices, weights, deltas, targets, target_weights = zip(*batch)

    lengths = torch.tensor([len(x) for x in items])
```

```
    return (
        pad_sequence(items,   batch_first=True, padding_value=0),
        pad_sequence(cats,    batch_first=True, padding_value=0),
        pad_sequence(brands,  batch_first=True, padding_value=0),
        pad_sequence(prices,  batch_first=True, padding_value=0.0),
        pad_sequence(weights, batch_first=True, padding_value=0.0),
        pad_sequence(deltas,  batch_first=True, padding_value=0.0),
        lengths,
        torch.stack(targets),
        torch.stack(target_weights),
    )


seq_df = seq_df.sample(n=len(seq_df), shuffle=True, seed=42)
split = int(0.8 * len(seq_df))

train_ds = SessionDataset(seq_df[:split])
test_ds = SessionDataset(seq_df[split:])

train_loader = DataLoader(train_ds, batch_size=256, shuffle=True, collate_fn=collate_fn,
num_workers=0, pin_memory=True)
test_loader = DataLoader(test_ds, batch_size=512, shuffle=False, collate_fn=collate_fn,
num_workers=0, pin_memory=True)

del seq_df

class Attention(nn.Module):
    def __init__(self, hidden_dim):
        super().__init__()
        self.W1 = nn.Linear(hidden_dim, hidden_dim, bias=False)
        self.W2 = nn.Linear(hidden_dim, hidden_dim, bias=False)
        self.v = nn.Linear(hidden_dim, 1, bias=False)

    def forward(self, hiddens, last_hidden, mask):
```

```
        energy = self.v(torch.tanh(self.W1(hiddens) + self.W2(last_hidden.unsqueeze(1))))
        energy = energy.squeeze(-1)
        energy = energy.masked_fill(~mask, -1e9)
        weights = F.softmax(energy, dim=1)
        context = (hiddens * weights.unsqueeze(-1)).sum(dim=1)
        return context, weights

class LSTMRec(nn.Module):
    def __init__(self, num_items, num_cats, num_brands, item_embed_dim=64, cat_embed_dim=16, brand_embed_dim=16, hidden_dim=128, num_layers=2, dropout=0.3, embed_dropout=0.2, use_attention=False):
        super().__init__()
        self.item_embedding = nn.Embedding(num_items, item_embed_dim, padding_idx=0)
        self.cat_embedding = nn.Embedding(num_cats, cat_embed_dim, padding_idx=0)
        self.brand_embedding = nn.Embedding(num_brands, brand_embed_dim, padding_idx=0)
        self.embed_drop = nn.Dropout(embed_dropout)
        self.use_attention = use_attention

        lstm_input_dim = item_embed_dim + cat_embed_dim + brand_embed_dim + 2

        self.lstm = nn.LSTM(
            lstm_input_dim, hidden_dim,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout
        )

        if use_attention:
            self.attention = Attention(hidden_dim)
            fc_input_dim = hidden_dim * 2
        else:
            fc_input_dim = hidden_dim
```

```
        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(fc_input_dim, item_embed_dim)

        self.output_bias = nn.Parameter(torch.zeros(num_items))

    def forward(self, items, cats, brands, prices, weights, deltas, lengths):
        item_emb = self.embed_drop(self.item_embedding(items))
        cat_emb = self.embed_drop(self.cat_embedding(cats))
        brand_emb = self.embed_drop(self.brand_embedding(brands))
        combined = torch.cat((item_emb, cat_emb, brand_emb), dim=-1)
        combined *= weights.unsqueeze(-1)

        time_feat = deltas.unsqueeze(-1)
        price_feat = prices.unsqueeze(-1)
        lstm_input = torch.cat([combined, time_feat, price_feat], dim=-1)

        packed = nn.utils.rnn.pack_padded_sequence(
            lstm_input, lengths.cpu(), batch_first=True, enforce_sorted=False
        )
        rnn_out, (h_n, _) = self.lstm(packed)
        last_hidden = h_n[-1]

        if self.use_attention:
            rnn_out, _ = pad_packed_sequence(rnn_out, batch_first=True)
            B, T = items.size()
            mask = torch.arange(T, device=items.device).unsqueeze(0) <
lengths.unsqueeze(1).to(items.device)
            context, _ = self.attention(rnn_out, last_hidden, mask)
            merged = torch.cat([context, last_hidden], dim=-1)
        else:
            merged = last_hidden

        merged = self.dropout(merged)
        proj = self.fc(merged)
```

```
        logits = torch.matmul(proj, self.item_embedding.weight.t())
        logits += self.output_bias
        return logits

class GRURec(nn.Module):
    def __init__(self, num_items, num_cats, num_brands,
                 item_embed_dim=64, cat_embed_dim=16, brand_embed_dim=16,
                 hidden_dim=128, num_layers=2, dropout=0.3, embed_dropout=0.2,
use_attention=False):
        super().__init__()
        self.item_embed_dim = item_embed_dim
        self.item_embedding = nn.Embedding(num_items, item_embed_dim,
padding_idx=0)
        self.cat_embedding = nn.Embedding(num_cats, cat_embed_dim, padding_idx=0)
        self.brand_embedding = nn.Embedding(num_brands, brand_embed_dim,
padding_idx=0)
        self.use_attention = use_attention
        self.embed_drop = nn.Dropout(embed_dropout)
        self.hidden_dim = hidden_dim

        input_dim = item_embed_dim + cat_embed_dim + brand_embed_dim + 2

        self.gru = nn.GRU(
            input_dim, hidden_dim,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout,
        )

        if use_attention:
            self.attention = Attention(hidden_dim)
            fc_input_dim = hidden_dim * 2
        else:
            fc_input_dim = hidden_dim
```

```
        self.layer_norm = nn.LayerNorm(hidden_dim)
        self.dropout = nn.Dropout(dropout)

        self.fc = nn.Linear(fc_input_dim, item_embed_dim)
        self.output_bias = nn.Parameter(torch.zeros(num_items))

    def forward(self, items, cats, brands, prices, weights, deltas, lengths):
        item_emb = self.embed_drop(self.item_embedding(items))
        cat_emb = self.embed_drop(self.cat_embedding(cats))
        brand_emb = self.embed_drop(self.brand_embedding(brands))
        combined = torch.cat([item_emb, cat_emb, brand_emb], dim=-1)
        combined *= weights.unsqueeze(-1)

        time_feat = deltas.unsqueeze(-1)
        price_feat = prices.unsqueeze(-1)
        raw_input = torch.cat([combined, time_feat, price_feat], dim=-1)

        packed = pack_padded_sequence(raw_input, lengths.cpu(), batch_first=True,
enforce_sorted=False)
        rnn_out, h_n = self.gru(packed)
        rnn_out, _ = nn.utils.rnn.pad_packed_sequence(rnn_out, batch_first=True)

        rnn_out = self.layer_norm(rnn_out)

        last_hidden = h_n[-1]

        if self.use_attention:
            T_out = rnn_out.size(1)
            mask = torch.arange(T_out, device=items.device).unsqueeze(0) <
lengths.unsqueeze(1).to(items.device)
            context, _ = self.attention(rnn_out, last_hidden, mask)
            merged = torch.cat([context, last_hidden], dim=-1)
        else:
            merged = last_hidden
```

```
        merged = self.dropout(merged)
        proj = self.fc(merged)

        logits = F.linear(proj, self.item_embedding.weight, self.output_bias)
        return logits

class SASRec(nn.Module):
    def __init__(self, num_items, num_cats, num_brands,
                 item_embed_dim=64, cat_embed_dim=16, brand_embed_dim=16,
                 hidden_dim=128, n_heads=4, n_layers=2,
                 dropout=0.3, embed_dropout=0.2, max_len=50):
        super().__init__()
        self.item_embed_dim = item_embed_dim
        self.hidden_dim = hidden_dim
        self.max_len = max_len

        self.item_embedding = nn.Embedding(num_items, item_embed_dim,
padding_idx=0)
        self.cat_embedding = nn.Embedding(num_cats, cat_embed_dim, padding_idx=0)
        self.brand_embedding = nn.Embedding(num_brands, brand_embed_dim,
padding_idx=0)
        self.embed_drop = nn.Dropout(embed_dropout)

        input_dim = item_embed_dim + cat_embed_dim + brand_embed_dim + 2
        self.input_proj = nn.Linear(input_dim, hidden_dim)

        self.pos_embedding = nn.Embedding(max_len, hidden_dim)

        encoder_layer = nn.TransformerEncoderLayer(
            d_model=hidden_dim,
            nhead=n_heads,
            dim_feedforward=hidden_dim * 4,
            dropout=dropout,
            activation="gelu",
```

```
            batch_first=True,
            norm_first=True,
        )
        self.transformer = nn.TransformerEncoder(encoder_layer, num_layers=n_layers)
        self.layer_norm = nn.LayerNorm(hidden_dim)

        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(hidden_dim, item_embed_dim)
        self.output_bias = nn.Parameter(torch.zeros(num_items))

        print(f"  [SASRec] {n_layers} layers, {n_heads} heads, hidden={hidden_dim}")

    def forward(self, items, cats, brands, prices, weights, deltas, lengths):
        B, T = items.size()

        item_emb = self.embed_drop(self.item_embedding(items))
        cat_emb = self.embed_drop(self.cat_embedding(cats))
        brand_emb = self.embed_drop(self.brand_embedding(brands))
        combined = torch.cat([item_emb, cat_emb, brand_emb], dim=-1)
        combined *= weights.unsqueeze(-1)
        time_feat = deltas.unsqueeze(-1)
        price_feat = prices.unsqueeze(-1)
        features = torch.cat([combined, time_feat, price_feat], dim=-1)

        hidden = self.input_proj(features)
        positions = torch.arange(T, device=items.device).unsqueeze(0).expand(B, -1)
        hidden = hidden + self.pos_embedding(positions)
        hidden = self.layer_norm(hidden)

        causal_mask = torch.triu(
            torch.ones(T, T, device=items.device, dtype=torch.bool), diagonal=1
        )
        padding_mask = torch.arange(T, device=items.device).unsqueeze(0) >=
lengths.unsqueeze(1).to(items.device)
```

```
            out = self.transformer(
                hidden,
                mask=causal_mask,
                src_key_padding_mask=padding_mask,
            )

            last_idx = (lengths - 1).long().to(items.device)
            last_hidden = out[torch.arange(B, device=items.device), last_idx]

            proj = self.fc(self.dropout(last_hidden))

            logits = torch.matmul(proj, self.item_embedding.weight.t())
            logits = logits + self.output_bias
            return logits


        def compute_loss(logits, targets, target_weights):
            per_sample = F.cross_entropy(logits, targets, reduction="none")
    return (per_sample * target_weights).mean()
```

Лістинг файлу app.py:

```
import pickle
from typing import Optional
from collections import defaultdict

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.nn.utils.rnn import pad_sequence, pack_padded_sequence,
pad_packed_sequence
from fastapi import FastAPI, HTTPException
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from pydantic import BaseModel
```

```
class Attention(nn.Module):
    def __init__(self, hidden_dim):
        super().__init__()
        self.W1 = nn.Linear(hidden_dim, hidden_dim, bias=False)
        self.W2 = nn.Linear(hidden_dim, hidden_dim, bias=False)
        self.v = nn.Linear(hidden_dim, 1, bias=False)

    def forward(self, hiddens, last_hidden, mask):
        energy = self.v(torch.tanh(self.W1(hiddens) + self.W2(last_hidden.unsqueeze(1))))
        energy = energy.squeeze(-1)
        energy = energy.masked_fill(~mask, -1e9)
        weights = F.softmax(energy, dim=1)
        context = (hiddens * weights.unsqueeze(-1)).sum(dim=1)
        return context, weights


class LSTMRec(nn.Module):
    def __init__(self, num_items, num_cats, item_embed_dim=64, cat_embed_dim=16,
hidden_dim=128, num_layers=2, dropout=0.3, embed_dropout=0.2, use_attention=False):
        super().__init__()
        self.item_embedding = nn.Embedding(num_items, item_embed_dim,
padding_idx=0)
        self.cat_embedding = nn.Embedding(num_cats, cat_embed_dim, padding_idx=0)
        self.embed_drop = nn.Dropout(embed_dropout)
        self.use_attention = use_attention

        lstm_input_dim = item_embed_dim + cat_embed_dim + 1


        self.lstm = nn.LSTM(
            lstm_input_dim, hidden_dim,
            num_layers=num_layers,
            batch_first=True,
```

```
            dropout=dropout
        )

        if use_attention:
            self.attention = Attention(hidden_dim)
            fc_input_dim = hidden_dim * 2
        else:
            fc_input_dim = hidden_dim

        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(fc_input_dim, item_embed_dim)

        self.output_bias = nn.Parameter(torch.zeros(num_items))

    def forward(self, items, cats, weights, deltas, lengths):
        item_emb = self.embed_drop(self.item_embedding(items))
        cat_emb = self.embed_drop(self.cat_embedding(cats))
        combined = torch.cat((item_emb, cat_emb), dim=-1)
        combined *= weights.unsqueeze(-1)

        time_feat = torch.log1p(deltas).unsqueeze(-1)
        lstm_input = torch.cat([combined, time_feat], dim=-1)

        packed = nn.utils.rnn.pack_padded_sequence(
            lstm_input, lengths.cpu(), batch_first=True, enforce_sorted=False
        )
        rnn_out, (h_n, _) = self.lstm(packed)
        last_hidden = h_n[-1]

        if self.use_attention:
            rnn_out, _ = pad_packed_sequence(rnn_out, batch_first=True)
            B, T = items.size()
            mask = torch.arange(T, device=items.device).unsqueeze(0) <
lengths.unsqueeze(1).to(items.device)
            context, _ = self.attention(rnn_out, last_hidden, mask)
```

```
            merged = torch.cat([context, last_hidden], dim=-1)
        else:
            merged = last_hidden

        merged = self.dropout(merged)
        proj = self.fc(merged)


        logits = torch.matmul(proj, self.item_embedding.weight.t())
        logits += self.output_bias
        return logits

class LSTMRec_opencdp(nn.Module):
    def __init__(self, num_items, num_cats, num_brands, item_embed_dim=64, cat_embed_dim=16, brand_embed_dim=16, hidden_dim=128, num_layers=2, dropout=0.3, embed_dropout=0.2, use_attention=False):
        super().__init__()
        self.item_embedding = nn.Embedding(num_items, item_embed_dim, padding_idx=0)
        self.cat_embedding = nn.Embedding(num_cats, cat_embed_dim, padding_idx=0)
        self.brand_embedding = nn.Embedding(num_brands, brand_embed_dim, padding_idx=0)
        self.embed_drop = nn.Dropout(embed_dropout)
        self.use_attention = use_attention

        lstm_input_dim = item_embed_dim + cat_embed_dim + brand_embed_dim + 2

        self.lstm = nn.LSTM(
            lstm_input_dim, hidden_dim,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout
        )

        if use_attention:
```

```
            self.attention = Attention(hidden_dim)
            fc_input_dim = hidden_dim * 2
        else:
            fc_input_dim = hidden_dim

        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(fc_input_dim, item_embed_dim)

        self.output_bias = nn.Parameter(torch.zeros(num_items))

    def forward(self, items, cats, brands, prices, weights, deltas, lengths):
        item_emb = self.embed_drop(self.item_embedding(items))
        cat_emb = self.embed_drop(self.cat_embedding(cats))
        brand_emb = self.embed_drop(self.brand_embedding(brands))

        combined = torch.cat((item_emb, cat_emb, brand_emb), dim=-1)
        combined *= weights.unsqueeze(-1)

        price_feat = prices.unsqueeze(-1)
        time_feat = deltas.unsqueeze(-1)
        lstm_input = torch.cat([combined, price_feat, time_feat], dim=-1)

        packed = nn.utils.rnn.pack_padded_sequence(
            lstm_input, lengths.cpu(), batch_first=True, enforce_sorted=False
        )
        rnn_out, (h_n, _) = self.lstm(packed)
        last_hidden = h_n[-1]

        if self.use_attention:
            rnn_out, _ = pad_packed_sequence(rnn_out, batch_first=True)
            B, T = items.size()
            mask = torch.arange(T, device=items.device).unsqueeze(0) <
lengths.unsqueeze(1).to(items.device)
            context, _ = self.attention(rnn_out, last_hidden, mask)
            merged = torch.cat([context, last_hidden], dim=-1)
```

```
            else:
                merged = last_hidden

            merged = self.dropout(merged)
            proj = self.fc(merged)

            logits = torch.matmul(proj, self.item_embedding.weight.t())
            logits += self.output_bias
            return logits

        class GRURec(nn.Module):
            def __init__(self, num_items, num_cats,
                         item_embed_dim=64, cat_embed_dim=16,
                         hidden_dim=128, num_layers=2, dropout=0.3, embed_dropout=0.2,
use_attention=False):
            super().__init__()
            self.item_embed_dim = item_embed_dim
            self.item_embedding = nn.Embedding(num_items, item_embed_dim,
padding_idx=0)
            self.cat_embedding = nn.Embedding(num_cats, cat_embed_dim, padding_idx=0)
            self.use_attention = use_attention
            self.embed_drop = nn.Dropout(embed_dropout)
            self.hidden_dim = hidden_dim

            input_dim = item_embed_dim + cat_embed_dim + 1

            self.gru = nn.GRU(
                input_dim, hidden_dim,
                num_layers=num_layers,
                batch_first=True,
                dropout=dropout,
            )

            if use_attention:
                self.attention = Attention(hidden_dim)
```

```
            fc_input_dim = hidden_dim * 2
        else:
            fc_input_dim = hidden_dim

        self.layer_norm = nn.LayerNorm(hidden_dim)
        self.dropout = nn.Dropout(dropout)

        self.fc = nn.Linear(fc_input_dim, item_embed_dim)
        self.output_bias = nn.Parameter(torch.zeros(num_items))

    def forward(self, items, cats, weights, deltas, lengths):
        item_emb = self.embed_drop(self.item_embedding(items))
        cat_emb = self.embed_drop(self.cat_embedding(cats))
        combined = torch.cat([item_emb, cat_emb], dim=-1)
        combined = combined * weights.unsqueeze(-1)

        time_feat = torch.log1p(deltas).unsqueeze(-1)
        raw_input = torch.cat([combined, time_feat], dim=-1)

        packed = pack_padded_sequence(raw_input, lengths.cpu(), batch_first=True,
enforce_sorted=False)
        rnn_out, h_n = self.gru(packed)
        rnn_out, _ = nn.utils.rnn.pad_packed_sequence(rnn_out, batch_first=True)


        rnn_out = self.layer_norm(rnn_out)

        last_hidden = h_n[-1]

        if self.use_attention:
            T_out = rnn_out.size(1)
            mask = torch.arange(T_out, device=items.device).unsqueeze(0) <
lengths.unsqueeze(1).to(items.device)
            context, _ = self.attention(rnn_out, last_hidden, mask)
            merged = torch.cat([context, last_hidden], dim=-1)
```

```
            else:
                merged = last_hidden

            merged = self.dropout(merged)
            proj = self.fc(merged)

            logits = F.linear(proj, self.item_embedding.weight, self.output_bias)
            return logits

        class GRURec_opencdp(nn.Module):
            def __init__(self, num_items, num_cats, num_brands,
                         item_embed_dim=64, cat_embed_dim=16, brand_embed_dim=16,
                         hidden_dim=128, num_layers=2, dropout=0.3, embed_dropout=0.2,
use_attention=False):
            super().__init__()
            self.item_embed_dim = item_embed_dim
            self.item_embedding = nn.Embedding(num_items, item_embed_dim,
padding_idx=0)
            self.cat_embedding = nn.Embedding(num_cats, cat_embed_dim, padding_idx=0)
            self.brand_embedding = nn.Embedding(num_brands, brand_embed_dim,
padding_idx=0)
            self.use_attention = use_attention
            self.embed_drop = nn.Dropout(embed_dropout)
            self.hidden_dim = hidden_dim

            input_dim = item_embed_dim + cat_embed_dim + brand_embed_dim + 2

            self.gru = nn.GRU(
                input_dim, hidden_dim,
                num_layers=num_layers,
                batch_first=True,
                dropout=dropout,
            )

            if use_attention:
```

```
            self.attention = Attention(hidden_dim)
            fc_input_dim = hidden_dim * 2
        else:
            fc_input_dim = hidden_dim

        self.layer_norm = nn.LayerNorm(hidden_dim)
        self.dropout = nn.Dropout(dropout)

        self.fc = nn.Linear(fc_input_dim, item_embed_dim)
        self.output_bias = nn.Parameter(torch.zeros(num_items))

    def forward(self, items, cats, brands, prices, weights, deltas, lengths):
        item_emb = self.embed_drop(self.item_embedding(items))
        cat_emb = self.embed_drop(self.cat_embedding(cats))
        brand_emb = self.embed_drop(self.brand_embedding(brands))
        combined = torch.cat([item_emb, cat_emb, brand_emb], dim=-1)
        combined = combined * weights.unsqueeze(-1)

        price_feat = prices.unsqueeze(-1)
        time_feat = deltas.unsqueeze(-1)
        raw_input = torch.cat([combined, price_feat, time_feat], dim=-1)

        packed = pack_padded_sequence(raw_input, lengths.cpu(), batch_first=True,
enforce_sorted=False)
        rnn_out, h_n = self.gru(packed)
        rnn_out, _ = nn.utils.rnn.pad_packed_sequence(rnn_out, batch_first=True)


        rnn_out = self.layer_norm(rnn_out)

        last_hidden = h_n[-1]

        if self.use_attention:
            T_out = rnn_out.size(1)
```

```
                mask = torch.arange(T_out, device=items.device).unsqueeze(0) < lengths.unsqueeze(1).to(items.device)
                context, _ = self.attention(rnn_out, last_hidden, mask)
                merged = torch.cat([context, last_hidden], dim=-1)
            else:
                merged = last_hidden

            merged = self.dropout(merged)
            proj = self.fc(merged)

            logits = F.linear(proj, self.item_embedding.weight, self.output_bias)
            return logits

        class SASRec(nn.Module):
            def __init__(self, num_items, num_cats,
                         item_embed_dim=64, cat_embed_dim=16,
                         hidden_dim=128, n_heads=4, n_layers=2,
                         dropout=0.3, embed_dropout=0.2, max_len=50):
                super().__init__()
                self.item_embed_dim = item_embed_dim
                self.hidden_dim = hidden_dim
                self.max_len = max_len

                self.item_embedding = nn.Embedding(num_items, item_embed_dim, padding_idx=0)
                self.cat_embedding = nn.Embedding(num_cats, cat_embed_dim, padding_idx=0)
                self.embed_drop = nn.Dropout(embed_dropout)

                input_dim = item_embed_dim + cat_embed_dim + 1
                self.input_proj = nn.Linear(input_dim, hidden_dim)

                self.pos_embedding = nn.Embedding(max_len, hidden_dim)

                encoder_layer = nn.TransformerEncoderLayer(
                    d_model=hidden_dim,
```

```
            nhead=n_heads,
            dim_feedforward=hidden_dim * 4,
            dropout=dropout,
            activation="gelu",
            batch_first=True,
            norm_first=True,
        )
        self.transformer = nn.TransformerEncoder(encoder_layer, num_layers=n_layers)
        self.layer_norm = nn.LayerNorm(hidden_dim)

        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(hidden_dim, item_embed_dim)
        self.output_bias = nn.Parameter(torch.zeros(num_items))

        print(f"  [SASRec] {n_layers} layers, {n_heads} heads, hidden={hidden_dim}")

    def forward(self, items, cats, weights, deltas, lengths):
        B, T = items.size()

        item_emb = self.embed_drop(self.item_embedding(items))
        cat_emb = self.embed_drop(self.cat_embedding(cats))
        combined = torch.cat([item_emb, cat_emb], dim=-1)
        combined = combined * weights.unsqueeze(-1)
        time_feat = torch.log1p(deltas).unsqueeze(-1)
        features = torch.cat([combined, time_feat], dim=-1)

        hidden = self.input_proj(features)
        positions = torch.arange(T, device=items.device).unsqueeze(0).expand(B, -1)
        hidden = hidden + self.pos_embedding(positions)
        hidden = self.layer_norm(hidden)

        causal_mask = torch.triu(
            torch.ones(T, T, device=items.device, dtype=torch.bool), diagonal=1
        )
```

```
            padding_mask = torch.arange(T, device=items.device).unsqueeze(0) >= lengths.unsqueeze(1).to(items.device)

            out = self.transformer(
                hidden,
                mask=causal_mask,
                src_key_padding_mask=padding_mask,
            )

            last_idx = (lengths - 1).long().to(items.device)
            last_hidden = out[torch.arange(B, device=items.device), last_idx]

            proj = self.fc(self.dropout(last_hidden))

            logits = torch.matmul(proj, self.item_embedding.weight.t())
            logits = logits + self.output_bias
            return logits

        class SASRec_opencdp(nn.Module):
            def __init__(self, num_items, num_cats, num_brands,
                         item_embed_dim=64, cat_embed_dim=16, brand_embed_dim=16,
                         hidden_dim=128, n_heads=4, n_layers=2,
                         dropout=0.3, embed_dropout=0.2, max_len=50):
                super().__init__()
                self.item_embed_dim = item_embed_dim
                self.hidden_dim = hidden_dim
                self.max_len = max_len

                self.item_embedding = nn.Embedding(num_items, item_embed_dim, padding_idx=0)
                self.cat_embedding = nn.Embedding(num_cats, cat_embed_dim, padding_idx=0)
                self.brand_embedding = nn.Embedding(num_brands, brand_embed_dim, padding_idx=0)
                self.embed_drop = nn.Dropout(embed_dropout)
```

```
        input_dim = item_embed_dim + cat_embed_dim + brand_embed_dim + 2
        self.input_proj = nn.Linear(input_dim, hidden_dim)

        self.pos_embedding = nn.Embedding(max_len, hidden_dim)

        encoder_layer = nn.TransformerEncoderLayer(
            d_model=hidden_dim,
            nhead=n_heads,
            dim_feedforward=hidden_dim * 4,
            dropout=dropout,
            activation="gelu",
            batch_first=True,
            norm_first=True,
        )
        self.transformer = nn.TransformerEncoder(encoder_layer, num_layers=n_layers)
        self.layer_norm = nn.LayerNorm(hidden_dim)

        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(hidden_dim, item_embed_dim)
        self.output_bias = nn.Parameter(torch.zeros(num_items))

        print(f"  [SASRec] {n_layers} layers, {n_heads} heads, hidden={hidden_dim}")

    def forward(self, items, cats, brands, prices, weights, deltas, lengths):
        B, T = items.size()

        item_emb = self.embed_drop(self.item_embedding(items))
        cat_emb = self.embed_drop(self.cat_embedding(cats))
        brand_emb = self.embed_drop(self.brand_embedding(brands))

        combined = torch.cat([item_emb, cat_emb, brand_emb], dim=-1)
        combined = combined * weights.unsqueeze(-1)
        price_feat = prices.unsqueeze(-1)
        time_feat = deltas.unsqueeze(-1)
        features = torch.cat([combined, price_feat, time_feat], dim=-1)
```

```
            hidden = self.input_proj(features)
            positions = torch.arange(T, device=items.device).unsqueeze(0).expand(B, -1)
            hidden = hidden + self.pos_embedding(positions)
            hidden = self.layer_norm(hidden)

            causal_mask = torch.triu(
                torch.ones(T, T, device=items.device, dtype=torch.bool), diagonal=1
            )
            padding_mask = torch.arange(T, device=items.device).unsqueeze(0) >=
lengths.unsqueeze(1).to(items.device)

            out = self.transformer(
                hidden,
                mask=causal_mask,
                src_key_padding_mask=padding_mask,
            )

            last_idx = (lengths - 1).long().to(items.device)
            last_hidden = out[torch.arange(B, device=items.device), last_idx]

            proj = self.fc(self.dropout(last_hidden))

            logits = torch.matmul(proj, self.item_embedding.weight.t())
            logits = logits + self.output_bias
            return logits

        with open("artifacts/artifacts_opencdp.pkl", "rb") as f:
            artifacts = pickle.load(f)

        item2idx = artifacts["item2idx"]
        idx2item = artifacts["idx2item"]
        cat2idx = artifacts["cat2idx"]
        item_to_cat_idx = artifacts["item_to_cat_idx"]
        item_to_cat_visual = artifacts["item_to_cat_visual"]
```

```
        item_to_brand_idx = artifacts["item_to_brand_idx"]
        item_to_brand_visual = artifacts["item_to_brand_visual"]
        item_to_price = artifacts["item_to_price"]
        item_to_price_visual = artifacts["item_to_price_visual"]
        num_items = artifacts["num_items"]
        num_cats = artifacts["num_cats"]
        num_brands = artifacts.get("num_brands", 0)
        EVENT_WEIGHTS = artifacts["event_weights"]
        MAX_SEQ = artifacts["max_seq"]

        device = torch.device("cpu")

        USE_ATTENTION = False

        MODELS = {}

        for name, cls, path in [
            ("lstm", LSTMRec_opencdp, "artifacts/lstm_model_opencdp.pt"),
            ("gru", GRURec_opencdp, "artifacts/gru_model_opencdp.pt"),
            ("sasrec", SASRec_opencdp, "artifacts/sas_model_opencdp.pt"),
        ]:
            try:
                if cls == SASRec_opencdp:
                    model = cls(num_items, num_cats, num_brands)
                else:
                    model = cls(num_items, num_cats, num_brands,
use_attention=USE_ATTENTION)
                model.load_state_dict(torch.load(path, map_location=device))
                model.eval()
                MODELS[name] = model
                print(f"Loaded {name} from {path}")
            except FileNotFoundError:
                print(f"Warning: {path} not found, skipping {name}")

        if not MODELS:
```

```
    raise RuntimeError("No models loaded. Run training and save_models first.")

def prepare_sequence(events: list[dict]) -> tuple:
    """
    Convert raw event dicts to model-ready tensors.

    Each event dict: {"itemid": int, "event": str, "timestamp": float}
    timestamp is in seconds (or ms – we handle both).
    """
    processed_items = []
    processed_cats = []
    processed_brands = []
    processed_prices = []
    processed_weights = []
    processed_deltas = []
    prev_ts = None

    for e in events:
        item_id = int(e["itemid"])
        event_type = e.get("event", "view")
        ts = float(e["timestamp"])

        if ts > 1e12:
            ts = ts / 1000.0

        item_idx = item2idx.get(item_id, 0)
        cat_idx = item_to_cat_idx.get(item_idx, 0)
        brand_idx = item_to_brand_idx.get(item_idx, 0)
        price = item_to_price.get(item_idx, 0.0)
        weight = EVENT_WEIGHTS.get(event_type, 1.0)
        delta = 0.0 if prev_ts is None else (ts - prev_ts) / 3600.0

        if item_idx > 0:
            processed_items.append(item_idx)
            processed_cats.append(cat_idx)
```

```
                processed_brands.append(brand_idx)
                processed_prices.append(price)
                processed_weights.append(weight)
                processed_deltas.append(delta)

            prev_ts = ts

        n = min(len(processed_items), MAX_SEQ)
        processed_items = processed_items[-n:]
        processed_cats = processed_cats[-n:]
        processed_brands = processed_brands[-n:]
        processed_prices = processed_prices[-n:]
        processed_weights = processed_weights[-n:]
        processed_deltas = processed_deltas[-n:]

        return (
            torch.tensor(processed_items, dtype=torch.long).unsqueeze(0),
            torch.tensor(processed_cats, dtype=torch.long).unsqueeze(0),
            torch.tensor(processed_brands, dtype=torch.long).unsqueeze(0),
            torch.tensor(processed_prices, dtype=torch.float).unsqueeze(0),
            torch.tensor(processed_weights, dtype=torch.float).unsqueeze(0),
            torch.tensor(deltas, dtype=torch.float).unsqueeze(0),
            torch.tensor([n]),
        )


    def get_recommendations(model_name: str, events: list[dict],
                            top_k: int = 10) -> list[dict]:
        model = MODELS.get(model_name)
        if model is None:
            raise ValueError(f"Unknown model: {model_name}. Available:
{list(MODELS.keys())}")

        tensors = prepare_sequence(events)
        if tensors is None:
```

```
    return []

t_items, t_cats, t_brands, t_prices, t_weights, t_deltas, lengths = tensors
with torch.no_grad():
    logits = model(t_items, t_cats, t_brands, t_prices, t_weights, t_deltas, lengths)

logits[0, 0] = -float("inf")

seen = set(t_items[0].tolist())
for idx in seen:
    logits[0, idx] = -float("inf")

scores, indices = logits.topk(top_k, dim=1)
probs = F.softmax(scores, dim=1)

results = []
for rank in range(top_k):
    idx = indices[0, rank].item()
    results.append({
        "rank": rank + 1,
        "item_id": idx2item.get(idx, -1),
        "category_id": item_to_cat_idx.get(idx, "0"),
        "category": item_to_cat_visual.get(idx, "0"),
        "brand_id": item_to_brand_idx.get(idx, "0"),
        "brand": item_to_brand_visual.get(idx, "0"),
        "price_visual": item_to_price_visual.get(idx, "0"),
        "price": item_to_price.get(idx, 0.0),
        "item_idx": idx,
        "score": round(scores[0, rank].item(), 4),
        "probability": round(probs[0, rank].item(), 4),
    })

return results
```

```
app = FastAPI(
    title="Retailrocket Recommender API",
    description="Session-based recommendation using LSTM, GRU, and SASRec models.",
    version="1.0.0",
)
app.mount("/static", StaticFiles(directory="./landing/static"), name="static")


class EventItem(BaseModel):
    itemid: int
    event: str = "view"
    timestamp: float

    class Config:
        json_schema_extra = {
            "example": {"itemid": 123456, "event": "view", "timestamp": 1.4335e9}
        }

class RecommendRequest(BaseModel):
    events: list[EventItem]
    model: str = "lstm"
    top_k: int = 10
    rule_boost: float = 0.0

    class Config:
        json_schema_extra = {
            "example": {
                "events": [
                    {"itemid": 461686, "event": "view", "timestamp": 1433221332},
                    {"itemid": 206783, "event": "view", "timestamp": 1433221345},
                    {"itemid": 206783, "event": "addtocart", "timestamp": 1433221378},
                ],
                "model": "lstm",
                "top_k": 10,
```

```
            "rule_boost": 0.3,
        }
    }


class RecommendItem(BaseModel):
    rank: int
    item_id: int
    category_id: int
    category: str
    brand_id: int
    brand: str
    price_visual: float
    price: float
    item_idx: int
    score: float
    probability: float


class RecommendResponse(BaseModel):
    model: str
    num_events: int
    recommendations: list[RecommendItem]

@app.get("/", response_class=HTMLResponse)
def landing_page():
    with open("./landing/index.html", "r") as f:
        return HTMLResponse(f.read())

@app.get("/health")
def health():
    return {"status": "ok", "models_loaded": list(MODELS.keys())}


@app.get("/models")
```

```
def list_models():
    return {
        "available": list(MODELS.keys()),
        "default": "lstm",
        "num_items": num_items,
        "num_categories": num_cats,
    }


@app.post("/recommend", response_model=RecommendResponse)
def recommend(req: RecommendRequest):
    if req.model not in MODELS:
        raise HTTPException(
            status_code=400,
            detail=f"Unknown model '{req.model}'. Available: {list(MODELS.keys())}",
        )

    if not req.events:
        raise HTTPException(status_code=400, detail="No events provided.")

    if req.top_k < 1 or req.top_k > 100:
        raise HTTPException(status_code=400, detail="top_k must be between 1 and 100.")

    events = [e.model_dump() for e in req.events]
    recs = get_recommendations(req.model, events, req.top_k)

    return RecommendResponse(
        model=req.model,
        num_events=len(req.events),
        recommendations=recs,
    )
```

Лістинг файлу index.html:

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Retailrocket Recommender</title>
<link rel="stylesheet" href="static/style.css">
</head>
<body>
<div class="container">
  <h1>Retailrocket Recommender</h1>
  <p class="subtitle">Session-based next-item prediction - LSTM, GRU, SASRec</p>

  <!-- Session Input -->
  <div class="panel">
    <h2>User Session</h2>
    <div class="events-list" id="eventsList"></div>
    <div style="display: flex; gap: 0.75rem; flex-wrap: wrap;">
      <button class="btn btn-secondary" id="addEventBtn">+ Add Event</button>
      <button class="btn btn-secondary" id="loadSampleBtn">Load Sample
Session</button>
    </div>
  </div>

  <!-- Controls -->
  <div class="panel">
    <h2>Settings</h2>
    <div class="controls">
      <div class="field">
        <label>Model</label>
        <select id="modelSelect">
          <option value="lstm">LSTM</option>
          <option value="gru">GRU</option>
          <option value="sasrec">SASRec</option>
        </select>
      </div>
```

```
      <div class="field">
        <label>Top-K</label>
        <input type="number" id="topK" value="10" min="1" max="50"
style="width:70px">
      </div>
<!--      <div class="field">-->
<!--        <label>Rule Boost</label>-->
<!--        <input type="number" id="ruleBoost" value="0" min="0" max="2" step="0.1"
style="width:80px">-->
<!--      </div>-->
      <button class="btn btn-primary" id="recBtn">Get Recommendations</button>
      <button class="btn btn-secondary" id="compareBtn">Compare All Models</button>
    </div>
  </div>

  <!-- Status -->
  <div class="status" id="status"></div>

  <!-- Results -->
  <div class="panel" id="resultsPanel" style="display:none;">
    <h2 id="resultsTitle">Recommendations</h2>
    <div id="resultsArea"></div>
  </div>
</div>
<script>
function addEvent(itemid = '', event = 'view', timestamp = '') {
  const list = document.getElementById('eventsList');
  const row = document.createElement('div');
  row.className = 'event-row';
  row.innerHTML = `
    <input type="number" placeholder="Item ID" value="${itemid}" class="ev-item">
    <select class="ev-type">
      <option value="view" ${event === 'view' ? 'selected' : ''}>view</option>
      <option value="addtocart" ${event === 'addtocart' ? 'selected' : ''}>addtocart</option>
```

```
    <option value="transaction" ${event === 'transaction' ? 'selected' : ''}>transaction</option>
   </select>
   <input type="datetime-local" step="1" placeholder="Timestamp" value="${timestamp}" class="ev-ts">
   <button class="remove-btn">&times;</button>
  `;
  row.querySelector('.remove-btn').addEventListener('click', () => {row.remove()});
  list.appendChild(row);
}

function loadSample() {
  document.getElementById('eventsList').innerHTML = '';
  const sample = [
    [1003487, 'view', new Date(1433210532 * 1000).toISOString().replace(/Z$/, '')],
    [1003571, 'view', new Date(1433221345 * 1000).toISOString().replace(/Z$/, '')],
    [1003571, 'addtocart', new Date(1433210545 * 1000).toISOString().replace(/Z$/, '')],
    [1004593, 'view', new Date(1433210578 * 1000).toISOString().replace(/Z$/, '')],
    [1004593, 'addtocart', new Date(1433210610 * 1000).toISOString().replace(/Z$/, '')],
    [1004593, 'transaction', new Date(1433210700 * 1000).toISOString().replace(/Z$/, '')],
  ];
  sample.forEach(([id, ev, ts]) => addEvent(id, ev, ts));
}

function collectEvents() {
  const rows = document.querySelectorAll('.event-row');
  const events = [];
  rows.forEach(row => {
    const itemid = parseInt(row.querySelector('.ev-item').value);
    const event = row.querySelector('.ev-type').value;
    const ts = parseFloat(new Date(row.querySelector('.ev-ts').value).getTime() / 1000);
    if (!isNaN(itemid) && !isNaN(ts)) {
      events.push({ itemid, event, timestamp: ts });
    }
  });
```

```
  return events;
}

function setStatus(msg, type) {
  const el = document.getElementById('status');
  el.textContent = msg;
  el.className = 'status' + (type ? ' status-' + type : '');
}

function clearStatus() {
  const el = document.getElementById('status');
  el.className = 'status';
}

function renderCards(recs) {
  return recs.map(r => `
    <div class="rec-card">
      <div class="rec-rank">#${r.rank}</div>
      <div class="rec-item">Item ${r.item_id}\n${r.category}, ${r.brand},
${r.price_visual}$</div>
      <div class="rec-meta">
        <span class="rec-score">score: ${r.score}</span><br>
        <span class="rec-prob">prob: ${(r.probability * 100).toFixed(1)}%</span>
      </div>
    </div>
  `).join('');
}

async function getRecommendations() {
  const events = collectEvents();
  if (events.length === 0) {
    setStatus('Add at least one event to the session.', 'error');
    return;
  }
```

```
  const model = document.getElementById('modelSelect').value;
  const topK = parseInt(document.getElementById('topK').value);

  setStatus('Loading recommendations...', 'loading');
  document.getElementById('recBtn').disabled = true;

  try {
    const resp = await fetch(`/recommend`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ "events": events, "model": model, "top_k": topK }),
    });

    if (!resp.ok) {
      const err = await resp.json();
      throw new Error(err.detail || 'Request failed');
    }

    const data = await resp.json();
    clearStatus();

    const panel = document.getElementById('resultsPanel');
    const title = document.getElementById('resultsTitle');
    const area = document.getElementById('resultsArea');

    panel.style.display = 'block';
    title.innerHTML = "Recommendations <span class=\"tag\">" + model.toUpperCase() +
"</span>";
    area.innerHTML = "<div class=\"results-grid\">" +
renderCards(data.recommendations) + "</div>";
  } catch (e) {
    setStatus(e.message, 'error');
  } finally {
    document.getElementById('recBtn').disabled = false;
  }
```

```
}

async function compareAll() {
  const events = collectEvents();
  if (events.length === 0) {
    setStatus('Add at least one event to the session.', 'error');
    return;
  }

const topK = parseInt(document.getElementById('topK').value);
  const models = ['lstm', 'gru', 'sasrec'];

  setStatus('Comparing all models...', 'loading');

  try {
    const results = await Promise.all(
      models.map(m =>
        fetch(`/recommend`, {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({ events, model: m, top_k: topK }),
        }).then(r => r.ok ? r.json() : r.json().then(e => { throw new Error(e.detail); }))
      )
    );

    clearStatus();

    const panel = document.getElementById('resultsPanel');
    const title = document.getElementById('resultsTitle');
    const area = document.getElementById('resultsArea');

    panel.style.display = 'block';
    title.textContent = 'Model Comparison';

    area.innerHTML = `<div class="compare-grid">
```

```
        ${results.map((data, i) => `
          <div class="compare-col">
            <h3>${models[i].toUpperCase()}</h3>
            <div class="results-grid" style="grid-template-columns:1fr;">
              ${renderCards(data.recommendations)}
            </div>
          </div>
        `).join('')}
      </div>`;
    } catch (e) {
      setStatus(e.message, 'error');
    }
  }

  // attach event listeners
  document.getElementById('addEventBtn').addEventListener('click', addEvent);
  document.getElementById('loadSampleBtn').addEventListener('click', loadSample);
  document.getElementById('recBtn').addEventListener('click', getRecommendations);
  document.getElementById('compareBtn').addEventListener('click', compareAll);

  // start with sample loaded
  loadSample();
  </script>
  </body>
</html>
```

ДОДАТОК Б. Графічні матеріали

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
Кафедра математичних методів системного аналізу
Дипломна робота на здобуття бакалавра
За освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»

Рекомендаційні системи на основі аналізу послідовності кліків в сфері електронної комерції

Виконав: Студент IV курсу, групи КА-22
Марченко Дмитро Сергійович
Керівник: професор кафедри ММСА, д.т.н.
Недашківська Надія Іванівна

2

Актуальність задачі

1. Увага користувача сьогодні – найцінніший ресурс.

   Інформаційне перенасичення спричиняє значне зменшення вікна уваги користувачів та велику мінливість їх рішень. Рекомендаційні системи допомагають збільшити утримання користувачів.

2. Покращення монетизації користувачів

   Достатньо точна персоналізована рекомендаційна система може сформувати звичку до сервісу і, відповідно, призвести до збільшення доходу з користувача.

3. Слабка застосовність класичних методів побудови рекомендацій до сфери e-commerce

   Велика кількість анонімних юзерів та нестабільність попиту роблять класичні підходи неефективними

3

Об’єкт, предмет та мета дослідження

Об’єкт дослідження – процес формування персоналізованих рекомендацій товарів на платформах електронної комерції на основі послідовностей взаємодій користувачів.

Предмет дослідження – моделі та методи побудови послідовних рекомендаційних систем із використанням рекурентних нейронних мереж та механізмів уваги.

Мета роботи – реалізувати та порівняти архітектури рекомендаційних систем, що спираються на аналіз послідовності кліків у сфері e-commerce.

4

Основні задачі дослідження

1. Огляд існуючих підходів до побудови рекомендаційних систем
2. Реалізація та навчання моделей LSTM4Rec, GRU4Rec та SASRec.
3. Порівняння моделей між собою та при навчанні на датасетах Retailrocket і OpenCDP.
4. Побудова фінального застосунку для формування рекомендацій (деплой моделей)

5

Формальна постановка задачі

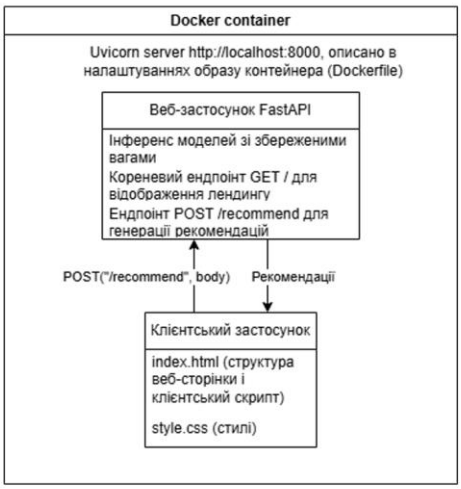
Послідовність дій одного користувача = багатовимірна впорядкована множина виду $\{i_1^{(1)}, i_2^{(2)}, \dots, i_k^{(t-1)}\}$, де $i_k^{(t)}$ - елемент, з яким користувач взаємодіяв на кроці t.

Задача прогнозування наступного елементу $i_m^{(t)}$, де 1 < m, k ≤ $|I|$:

$$i_m^{(t)} = arg \max_{i \in I} P(i | i_1^{(1)}, i_2^{(2)}, \dots, i_k^{(t-1)}).$$

6

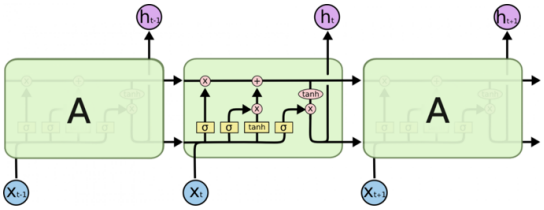
Архітектура програми



7

LSTM

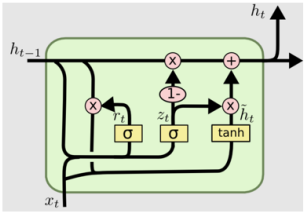
1. Гейт забування
$$\boldsymbol{f}_t = \sigma\big(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}||\boldsymbol{x}_t] + \boldsymbol{b}_f\big)$$
2. Вхідний гейт
$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}||\boldsymbol{x}_t] + \boldsymbol{b}_i)$$
3. Кандидат стану комірки
$$\widetilde{\boldsymbol{c}_t} = \tanh(\boldsymbol{W}_c[\boldsymbol{h}_{t-1}||\boldsymbol{x}_t] + \boldsymbol{b}_c)$$
4. Оновлення стану комірки
$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \widetilde{\boldsymbol{c}_t}$$
5. Вихідний гейт
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o[\boldsymbol{h}_{t-1}||\boldsymbol{x}_t] + \boldsymbol{b}_o)$$
6. Оновлення прихованого стану
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$



8

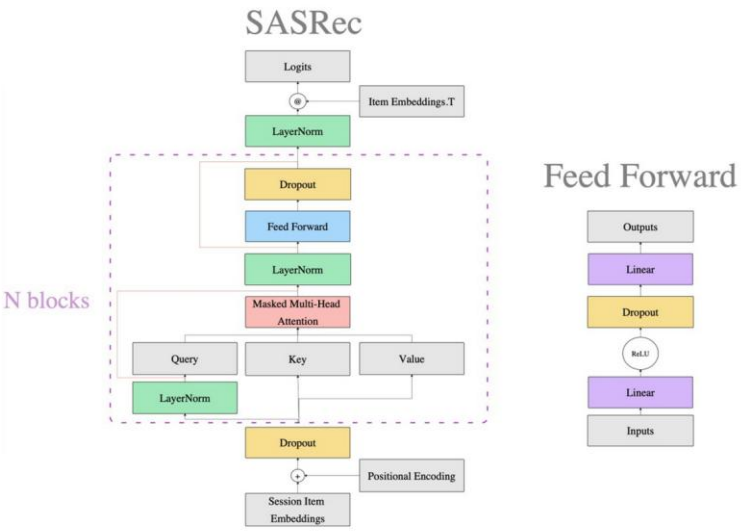
GRU

1. Гейт скидання
$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r[\boldsymbol{h}_{t-1}||\boldsymbol{x}_t] + \boldsymbol{b}_r)$$
2. Гейт оновлення
$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z[\boldsymbol{h}_{t-1}||\boldsymbol{x}_t] + \boldsymbol{b}_z)$$
3. Кандидат прихованого стану
$$\widetilde{\boldsymbol{h}_t} = \tanh(\boldsymbol{W}_h[(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1})||\boldsymbol{x}_t] + \boldsymbol{b}_h)$$
4. Оновлення прихованого стану
$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \widetilde{\boldsymbol{h}_t}$$



SASRec

9



SASRec

10

1. Позиційний ембединг

$$\boldsymbol{s}_t = \boldsymbol{W}_{proj}\boldsymbol{x}_t + \boldsymbol{P}[t]$$

2. Механізм уваги

$$Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}} + \boldsymbol{M}_{causal}\right)\boldsymbol{V}$$

3. Багатоголова увага
$MultiHead(Q, K, V) = [head_1 || \dots || head_n]\boldsymbol{W}^O$, де $head_i =$
$= Attention(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V)$

11

Набори даних

Retailrocket

Назва колонки	Тип даних	Опис колонки
timestamp	int64	Момент часу, в який сталася взаємодія в UNIX форматі (кількість мілісекунд з 01.01.1970)
visitorid	int	Унікальний ідентифікатор користувача
event	string	Один із трьох можливих типів взаємодій: view (перегляд), addtocart (додання в кошик), transaction (покупка)
itemid	int	Унікальний ідентифікатор товару
transactionid	int	Унікальний ідентифікатор транзакції (ненульове значення тільки для event = transaction)

OpenCDP

Назва колонки	Тип даних	Опис колонки
event_time	Datetime64	Дата й час взаємодії (часова зона UTC)
event_type	String	Один із чотирьох можливих типів взаємодій: view (перегляд), cart (додання в кошик), purchase (покупка), remove_from_cart (видалення з кошику)
product_id	int	Унікальний ідентифікатор товару
category_id	int64	Унікальний ідентифікатор категорії товару
category_code	str	Категорія товару
brand	str	Бренд товару
price	float	Ціна товару у доларах
user_id	int	Унікальний ідентифікатор користувача
user_session	int	Унікальний ідентифікатор користувацької сесії

12

Порівняння ефективності моделей

Retailrocket

Модель/метрика	HR@5	Precision@5	MAP@5	NDCG@5
LSTM	0.3384	0.0677	0.2977	0.308
GRU	0.3351	0.067	0.2951	0.3053
SASRec	0.3074	0.0615	0.2689	0.2787

Модель/метрика	HR@10	Precision@10	MAP@10	NDCG@10
LSTM	0.355	0.0355	0.3	0.3134
GRU	0.35	0.035	0.2972	0.3101
SASRec	0.3236	0.0324	0.2711	0.2839

Модель/метрика	HR@20	Precision@20	MAP@20	NDCG@20
LSTM	0.3691	0.0185	0.3009	0.317
GRU	0.3648	0.0182	0.2982	0.3139
SASRec	0.3382	0.0169	0.2721	0.2876

OpenCDP

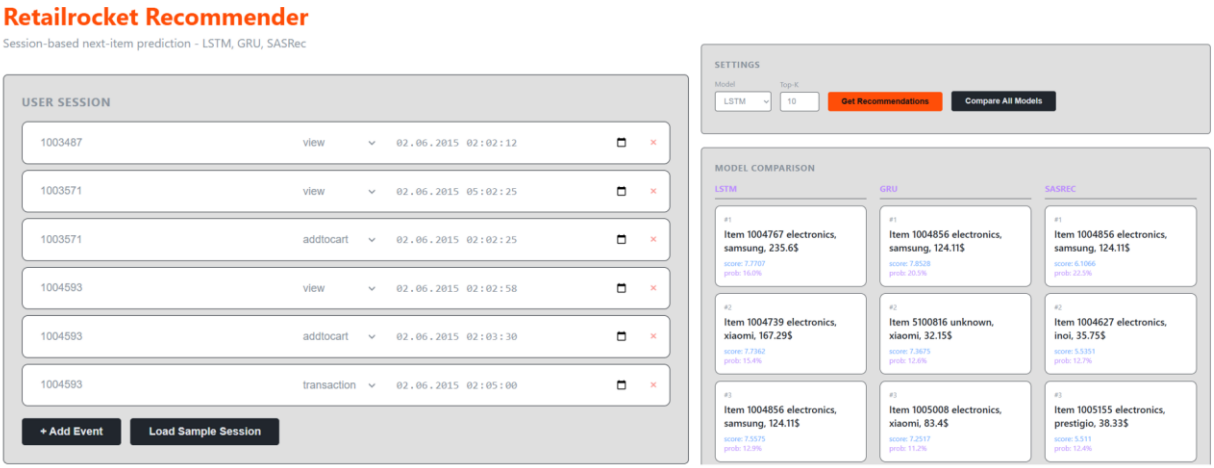
Модель/метрика	HR@5	Precision@5	MAP@5	NDCG@5
LSTM	0.4494	0.0899	0.3775	0.3955
GRU	0.4502	0.09	0.3787	0.3965
SASRec	0.4554	0.0911	0.3896	0.4060

Модель/метрика	HR@10	Precision@10	MAP@10	NDCG@10
LSTM	0.4999	0.05	0.3843	0.4118
GRU	0.5008	0.0501	0.3855	0.4129
SASRec	0.5045	0.0505	0.3961	0.4219

Модель/метрика	HR@20	Precision@20	MAP@20	NDCG@20
LSTM	0.5518	0.0276	0.3879	0.4249
GRU	0.5526	0.0276	0.3891	0.426
SASRec	0.5566	0.0278	0.3998	0.435

13

Фінальний вигляд продукту



14

Висновки

- Проаналізовано сучасний стан рекомендаційних систем в e-commerce; для дослідження обрано послідовні моделі LSTM4Rec, GRU4Rec та SASRec.
- Формалізовано математичний апарат моделей та метрик оцінювання.
- Реалізовано повну систему мовою Python (pandas, PyTorch, Polars): обробку даних Retailrocket і OpenCDP, навчання та експорт моделей, веб-сервіс для генерації рекомендацій.
- За результатами експериментів (K = 5, 10, 20) встановлено найвищу ефективність LSTM4Rec на Retailrocket та SASRec на OpenCDP.
- Система підтвердила теоретичну та практичну спроможність і придатна для інтеграції в реальні платформи e-commerce

15

Перспективи подальших досліджень

1. Виявлення змін у статистичних властивостях вхідних даних (дрейфу даних), прийняття рішення про автоматичне донавчання моделей
2. Впровадження контролю версій (Git) та побудова CI/CD-пайплайну для зручного внесення змін в побудовані моделі або клієнтський застосунок
3. Дослідження ефективності гібридних моделей на основі побудованих

16

Дякую за увагу!