Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для інтелектуального підбору книг на основі уподобань користувача
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-4
Анастасія Приходько
(власне ім'я, прізвище)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект
(повна назва освітньої програми)
Керівник ас. Максим Поліїт
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Приходько Анастасії Олександрівні
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для інтелектуального підбору книг на основі уподобань користувача

затверджена наказом університету від 19 травня 20 25 р. № 378Ст

2. Термін подання студентом роботи до екзаменаційної комісії 25 червня 20 25 р.

3. Вихідні дані до роботи Наукові статті та огляди з тематики рекомендаційних систем і великих мовних моделей; офіційна документація OpenAI Assistant API; інтерфейсні компоненти React; API-ключ для взаємодії з мовною моделлю; приклади реалізації подібних систем

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі і постановка задачі
2) Сучасні підходи до реалізації рекомендаційних систем
3) Попереднє проєктування та вибір засобів реалізації
4) Практична реалізація програмної системи

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Аналіз предметної галузі	20.05.2025 – 22.05.2025	виконано
3	Дослідження сучасних рекомендаційних систем	23.05.2025 – 24.05.2025	виконано
4	Збір даних та постановка задачі	25.05.2025 – 27.05.2025	виконано
5	Проєктування програмної системи	28.05.2025 – 03.06.2025	виконано
6	Програмна реалізація проєкту	04.06.2025 – 09.06.2025	виконано
7	Оформлення пояснювальної записки	10.06.2025 – 13.06.2025	виконано
8	Підготовка до захисту кваліфікаційної роботи	14.06.2025 – 24.06.2025	виконано
9	Захист кваліфікаційної роботи	25.06.2025	виконано

Дата видачі завдання 19 травня 2025 р.

Здобувач _____ (підпис)

Керівник роботи _____ ас. Максим Поліїт

(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 70 с., 15 рис., 2 дод., 20 джерел.

КОРИСТУВАЧ, ОБРОБКА ПРИРОДНОЇ МОВИ, ПЕРСОНАЛІЗАЦІЯ, ПРОГРАМНА СИСТЕМА, РЕКОМЕНДАЦІЙНА СИСТЕМА, СЕМАНТИЧНИЙ АНАЛІЗ, ТЕКСТОВИЙ ЗАПИТ, ШТУЧНИЙ ІНТЕЛЕКТ, GPT.

Об’єкт дослідження – інтелектуальні рекомендаційні системи, які поєднують технології обробки природної мови, ШІ та персоналізованої взаємодії з користувачем. Особлива увага приділена системам, які здатні інтерпретувати текстові запити та здійснювати підбір об’єктів на основі семантичного аналізу з використанням моделей типу GPT.

Предмет дослідження – методи, моделі та програмні засоби реалізації інтелектуального підбору книг за запитами користувача, сформульованими природною мовою. Розглянуто підходи до інтеграції LLM у застосунки для підбору персоналізованих рекомендацій.

Мета роботи – проєктування та обґрунтування архітектури програмної системи, що забезпечує пошук і підбір книг на основі природномовних запитів із застосуванням GPT-моделі.

Методи дослідження – аналіз наукових джерел, порівняння підходів до реалізації рекомендаційних систем, моделювання архітектури системи, а також використання сучасних засобів програмної розробки.

У роботі досліджено і описано основні типи рекомендаційних систем, розглянуто роль обробки природної мови та GPT-моделей. Сформульовано задачу розробки системи підбору книг, визначено функціональні вимоги, запропоновано програмне рішення. Обґрунтовано переваги такого підходу для реалізації інтелектуального пошуку.

ABSTRACT

Bachelor’s thesis contains: 70 pp., 15 fig., 2 ann., 20 references.

ARTIFICIAL INTELLIGENCE, GPT, NATURAL LANGUAGE PROCESSING, PERSONALIZATION, RECOMMENDATION SYSTEM, SEMANTIC ANALYSIS, SOFTWARE SYSTEM, TEXTUAL QUERY, USER.

The object of the study is intelligent recommender systems that combine natural language processing technologies, artificial intelligence, and personalized user interaction. Particular attention is given to systems capable of interpreting textual queries and selecting relevant items based on semantic analysis using GPT-like models.

The subject of the study is the methods, models, and software tools for implementing intelligent book recommendation based on user queries formulated in natural language. The work considers approaches to integrating LLMs into applications for generating personalized recommendations.

The aim of the research is to design and justify the architecture of a software system that enables book search and selection based on natural language queries using a GPT model.

The research methods include analysis of scientific sources, comparison of recommendation system implementation approaches, modeling of the system architecture, and the use of modern software development tools.

The work examines and describes the main types of recommender systems, discusses the role of natural language processing and GPT models, formulates the task of developing a book recommendation system, defines functional requirements, and proposes a software solution. The advantages of this approach for implementing intelligent search through natural language interaction are substantiated.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – Artificial Intelligence – штучний інтелект;

Angular – фронтенд-фреймворк для розробки вебдодатків, створений Google;

API – Application Programming Interface – інтерфейс прикладного програмування;

Assistant API – Assistant Application Programming Interface – інтерфейс прикладного програмування для взаємодії з асистентом ;

BERT – Bidirectional Encoder Representations from Transformers – двонапрямне кодувальне подання на основі трансформерів;

CSS – Cascading Style Sheets – каскадні таблиці стилів;

DOM – Document Object Model – об'єктна модель документа;

Express.js – фреймворк для Node.js;

FastText – Fast Text Representation – швидке подання тексту;

F1-score – F1-міра – гармонійне середнє точності та повноти;

GloVe – Global Vectors for Word Representation – глобальні вектори для подання слів;

GPT – Generative Pre-trained Transformer – генеративний попередньо тренований трансформер;

GPT-4-turbo – оптимізована версія мовної моделі GPT-4;

GRU – Gated Recurrent Unit – рекурентний блок з керованими воротами;

HTTP – HyperText Transfer Protocol – протокол передавання гіпертексту;

JavaScript – мова програмування для створення інтерактивних елементів на вебсторінках;

JSON – JavaScript Object Notation – нотація об’єктів JavaScript;

JSX –JavaScript XML – розширення синтаксису JavaScript;

LLaMA – Large Language Model Meta AI – велика мовна модель від Meta AI;

LLM – Large Language Models – велика мовна модель;

LSI – Latent Semantic Indexing – латентне семантичне індексування;

LSTM – Long Short-Term Memory – довготривала короткочасна пам’ять (тип рекурентної нейронної мережі);

MAE – Mean Absolute Error – середня абсолютна похибка;

NLP – Natural Language Processing – обробка природної мови;

Node.js – середовище виконання JavaScript на сервері, побудоване на рушії V8 від Google;

OpenAI – компанія, що спеціалізується на розробці штучного інтелекту;

PaLM – Pathways Language Model – мовна модель Pathways;

React – бібліотека JavaScript;

RESTful API – Representational State Transfer Application Programming Interface – інтерфейс прикладного програмування, що реалізує принципи архітектури REST для обміну даними між клієнтом і сервером;

RMSE – Root Mean Square Error – середньоквадратична похибка;

RNN – Recurrent Neural Network – рекурентна нейронна мережа;

TF-IDF – Term Frequency-Inverse Document Frequency – частота термів – обернена частота документів;

T5 – Text-to-Text Transfer Transformer – трансформер для переносу навчання «текст у текст»;

Vite – інструмент для швидкої розробки фронтенду;

Vue.js– JavaScript-фреймворк для створення інтерфейсів користувача та односторінкових додатків;

Word2Vec – Word to Vector – слово у вектор (модель подання слів у вигляді векторів)

ВСТУП

В умовах стрімкої цифрової трансформації суспільства та інформаційного перенасичення зростає потреба в системах, здатних ефективно відфільтровувати, структурувати та інтерпретувати великі обсяги даних відповідно до індивідуальних потреб користувачів. Одним із провідних інструментів, що забезпечують таку функціональність, є рекомендаційні системи – програмні рішення, призначені для формування персоналізованих пропозицій на основі аналізу поведінки, переваг, контексту чи запитів користувача.

Традиційні підходи до побудови рекомендаційних систем базуються переважно на колаборативній фільтрації, контентному аналізі або їх комбінаціях. Однак такі системи мають низку обмежень, пов’язаних з необхідністю попередньої обробки даних, чіткої структуризації запитів і складнощами у роботі з природномовними формулюваннями. Це суттєво ускладнює взаємодію з користувачем у випадках, коли той не може або не хоче точно формалізувати свої потреби у вигляді ключових слів або параметрів.

З появою великих мовних моделей, зокрема GPT, відкрились нові можливості для реалізації рекомендаційних систем, орієнтованих на природну мову. Ці моделі здатні аналізувати текстові запити користувачів, виявляти їхні інтенції, здійснювати семантичну обробку контенту й формувати логічні, контекстно релевантні відповіді. Такий підхід дозволяє перейти від фільтраційно-категоріальної взаємодії до гнучкого, діалогового формату, що наближає систему до реального спілкування людини з консультантом.

Об’єктом дослідження в межах кваліфікаційної роботи є процес формування рекомендацій за запитами користувача, вираженими у природній мові, а предметом – методи побудови програмних систем, що

поєднують технології web-розробки, обробки природної мови та інтеграцію LLM.

Метою є набуття досвіду формалізації інженерної задачі, вивчення існуючих методів її вирішення, аналіз архітектурних варіантів реалізації системи, а також розробка програмного застосунку для інтелектуального підбору книг за текстовими запитами користувачів.

У ході виконання кваліфікаційної роботи було опрацьовано теоретичну базу предметної галузі, здійснено аналіз сучасних методів рекомендації, визначено вимоги до системи, обґрунтовано вибір архітектури рішення, а також окреслено напрями подальшої реалізації функціонального прототипу. Отримані результати мають прикладне значення та можуть бути адаптовані до інших інформаційних систем, орієнтованих на персоналізовану взаємодію з користувачем у форматі природномовного діалогу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ І ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі

Рекомендаційні системи є важливою складовою сучасних інформаційних технологій, які спрямовані на персоналізований відбір інформації відповідно до потреб та уподобань конкретного користувача. Основна мета рекомендаційної системи полягає в тому, щоб автоматично надати користувачу релевантні об’єкти (товари, фільми, книги тощо), базуючись на його інтересах або поведінці.

Принцип дії будь-якої рекомендаційної системи ґрунтується на аналізі взаємодії між користувачами та об’єктами. Така взаємодія може включати перегляди, оцінки, покупки, збереження у списки бажаного, написання відгуків тощо.

Система використовує ці дані для побудови прогнозів – наприклад, які книги з високою ймовірністю можуть зацікавити користувача, навіть якщо він ще не взаємодіяв із ними.

В основі більшості систем лежить ідея зменшення інформаційного перевантаження та підвищення ефективності пошуку. У контексті цифрових бібліотек або онлайн-книгарень, де може бути доступно тисячі найменувань літератури, рекомендаційна система значно спрощує навігацію по каталогу, автоматизуючи процес відбору літератури, яка потенційно відповідатиме смакам читача.

З технічної точки зору, рекомендаційні системи реалізуються через алгоритмічні підходи, що дозволяють знаходити зв’язки між об’єктами або користувачами. На рисунку 1.1 зображена схема найпоширеніших рекомендаційних систем. Найбільш поширеними підходами в сучасних рекомендаційних системах є контентно-орієнтовані, колаборативні та гібридні методи.

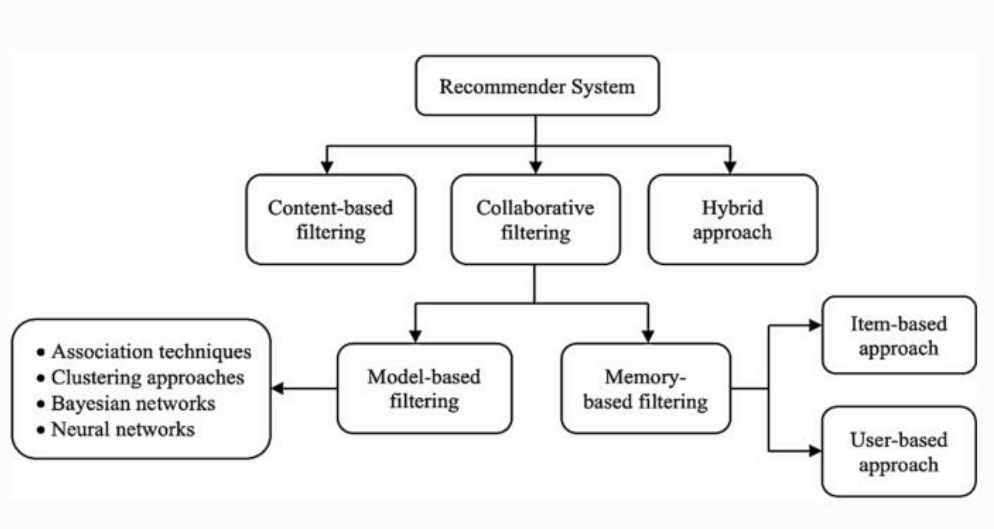

Рисунок 1.1 – Схема найпоширеніших рекомендаційних систем

Контентно-орієнтовані рекомендаційні системи ґрунтуються на аналізі властивостей самих об’єктів і створенні індивідуального профілю користувача на основі його попередніх вподобань. Основна ідея такого підходу полягає у тому, що система пропонує користувачу нові об’єкти, схожі на ті, що вже були ним позитивно оцінені. У випадку з книгами, це можуть бути твори з подібною тематикою, жанром, стилістикою або навіть мовними особливостями.

Кожен об’єкт у контентно-орієнтованій системі описується набором атрибутів (наприклад, для книги: автор, жанр, ключові слова, кількість сторінок, рік видання тощо). Ці характеристики кодуються у вигляді векторів ознак, які потім порівнюються з вектором профілю користувача за допомогою метрик схожості [1].

Формування профілю користувача здійснюється на основі історії взаємодії з об’єктами. Наприклад, якщо користувач раніше позитивно оцінював детективи з історичним сюжетом, то система з великою ймовірністю запропонує йому подібні за змістом книги. Такий підхід дозволяє системі бути індивідуалізованою і враховувати унікальні інтереси

кожного користувача. На рисунку 1.2 наведена схема контентно-орієнтованої рекомендаційні системи.

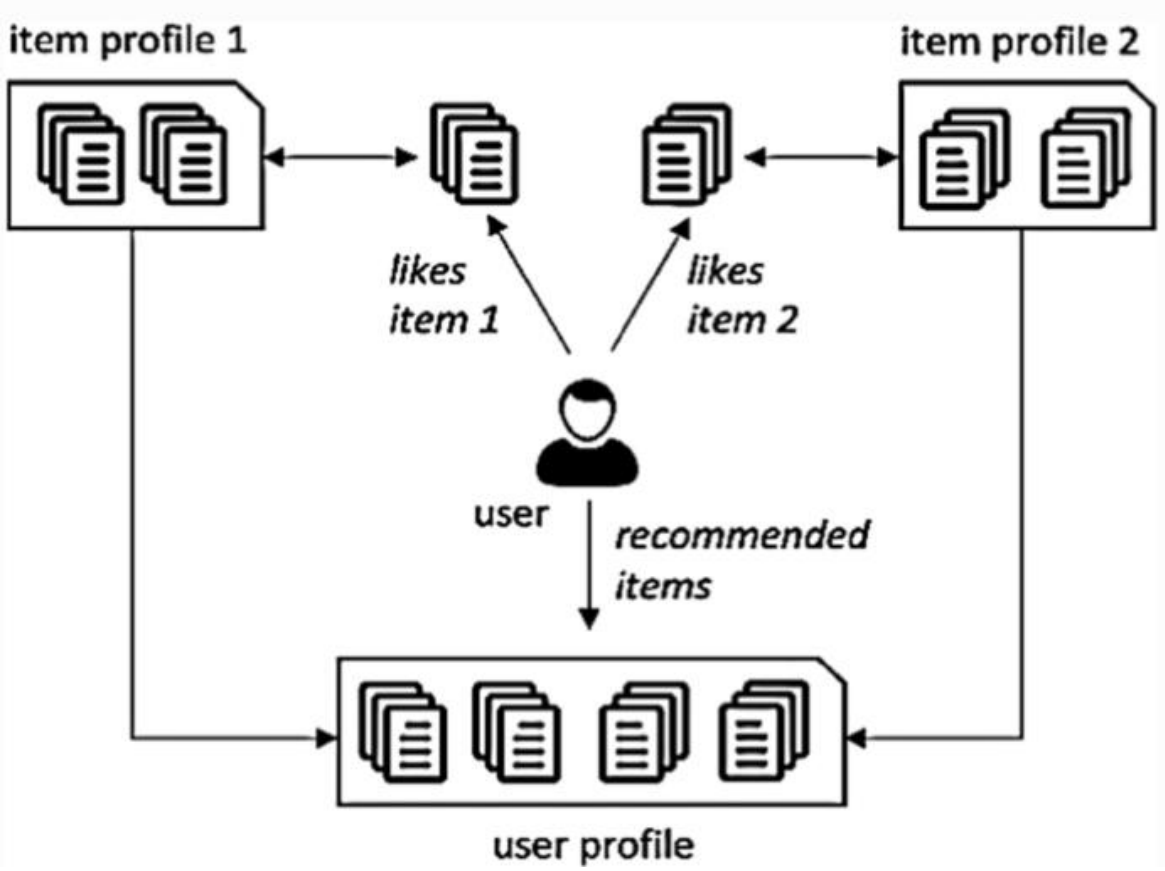

Рисунок 1.2 – Схема контентно-орієнтованої рекомендаційні системи

Колаборативна фільтрація є одним з найпоширеніших та ефективних підходів у побудові рекомендаційних систем, оскільки дозволяє враховувати поведінкові шаблони користувачів, а не лише характеристики самих об’єктів. Основна ідея полягає у припущенні, що користувачі з подібними інтересами в минулому й надалі будуть цікавитись схожими об'єктами.

У роботі [2] автори розглядають обмеження традиційної контентно-орієнтованої фільтрації та показують, що поєднання її з колаборативним підходом – зокрема, через обробку природної мови (NLP) дозволяє отримувати рекомендації більш точні, персоналізовані та інтерпретовані.

Наприклад, коли користувач залишає текстовий запит чи відгук, система за допомогою NLP розпізнає намір, а потім на основі цього створює модель переваг, яку можна співвіднести з іншими користувачами за допомогою колаборативної логіки. Інтеграція NLP у колаборативну фільтрацію дозволяє розв’язати одну з головних проблем цього підходу – нестачу даних про нових користувачів.

Проте колаборативні методи демонструють найкращу ефективність при наявності великої кількості користувачів та багатої історії взаємодій. В умовах сучасних застосунків, де доступна як поведінкова, так і текстова інформація (запити, відгуки, описи), найбільш ефективними стають гібридні системи.

Гібридні рекомендаційні системи є результатом поєднання кількох підходів – найчастіше контентно-орієнтованої фільтрації та колаборативної фільтрації. Їхньою головною метою є усунення недоліків окремих моделей і підвищення точності та стійкості системи до зміни поведінки користувачів або структури даних.

На рисунку 1.3 зображено узагальнену схему функціонування гібридної рекомендаційної системи, яка використовує дані від користувача для формування персоналізованих рекомендацій.

Рисунок 1.3 – Комбінування різних методів у рекомендаційній системі

Система отримує різноманітну вхідну інформацію: параметри користувача, його поведінку, прямі текстові запити, обмеження атрибутів. Ці дані надходять до модулів системи, які реалізують окремі підходи до формування рекомендацій.

Зокрема, вказано на аналіз рейтингів, колаборативну фільтрацію, аналіз поведінки та пряму обробку запиту. Результати роботи окремих модулів надсилаються на етап комбінування методів, де відбувається об’єднання рекомендацій, отриманих різними способами. У результаті система формує висновки щодо релевантних об’єктів і повертає користувачу персоналізовані рекомендації.

Ця схема добре ілюструє принцип гібридності, де інтегруються як колаборативні методи (через аналіз схожих користувачів), так і контентно-орієнтовані (через обробку запитів), що дозволяє підвищити точність, гнучкість і адаптивність системи до різних сценаріїв використання.

Гібридні рекомендаційні системи становлять окремий клас алгоритмів, які поєднують у собі різні підходи до формування рекомендацій з метою підвищення їхньої точності, стійкості та адаптивності. Основною метою гібридизації є подолання обмежень, притаманних окремим класам систем, зокрема контентно-орієнтованим та колаборативним. Згідно з дослідженням [3], гібридні системи демонструють вищу якість рекомендацій завдяки ефективному поєднанню переваг базових методів, що дозволяє зменшити вплив таких проблем, як «холодний старт», розрідженість даних і надмірна спеціалізація.

У науковій літературі запропоновано кілька стратегій гібридизації: послідовне об’єднання, паралельне поєднання результатів, зважене комбінування, а також методи на основі метамоделей. Кожна з них має свої особливості інтеграції вихідних алгоритмів. Наприклад, каскадне гібридне моделювання дозволяє одному алгоритму уточнювати результати іншого, тоді як зважене поєднання агрегує вихідні рекомендації із заданими ваговими коефіцієнтами.

Гібридні системи також часто включають додаткові джерела інформації – демографічні, поведінкові або контекстні дані, що дозволяє формувати більш персоналізовані рекомендації. Разом із тим автори підкреслюють, що гібридні підходи супроводжуються низкою викликів, пов’язаних з вибором оптимальної стратегії поєднання, балансуванням між точністю і різноманітністю, а також складністю реалізації та обчислювальними витратами.

Отже, гібридні рекомендаційні системи є перспективним напрямом розвитку інтелектуального підбору інформації, який поєднує точність, адаптивність та широку гнучкість у відповідь на індивідуальні потреби користувача.

1.2 Постановка задачі

У межах даної кваліфікаційної роботи ставиться задача створення застосунку, який дозволяє користувачам здійснювати пошук і отримання книжкових рекомендацій за допомогою як класичних інтерфейсів, так і інтелектуального асистента. Основна мета полягає в розробці програмної системи, що поєднує традиційний каталог книжок із можливістю фільтрації та пошуку із сучасним AI-асистентом, реалізованим на базі великої мовної моделі GPT-4-turbo через платформу OpenAI.

Завдання проєкту охоплює повний цикл проєктування та реалізації функціональних компонентів системи: від побудови структури даних локального сховища книжок до інтеграції асистента, здатного інтерпретувати запити користувачів, сформульовані природною мовою, і повертати відповідні результати виключно з наданого каталогу. Особлива увага приділяється забезпеченню гнучкості взаємодії: користувач самостійно обирає між традиційною навігацією по каталогу або діалогом із AI-асистентом. У процесі реалізації передбачено врахування вимог до доступності інтерфейсу, коректної обробки запитів, точності рекомендацій,

а також демонстрація сценаріїв взаємодії з системою – як позитивних, так і негативних.

Очікуваним результатом є універсальний застосунок, здатний емулювати поведінку консультанта в книжковому магазині, водночас демонструючи переваги використання сучасних мовних моделей у персоналізованих інформаційних системах.

1.3 Методи обробки природної мови в системах рекомендацій

Методи обробки природної мови відіграють ключову роль у сучасних рекомендаційних системах, особливо в тих випадках, коли джерелом інформації є текст – як описи товарів, рецензії користувачів, запити у вільній формі або вміст самих об’єктів (наприклад, книг). Застосування NLP дозволяє системі краще інтерпретувати контекст, розуміти наміри користувача, та генерувати релевантні персоналізовані рекомендації на основі неструктурованих текстових даних.

Один із найпоширеніших напрямів застосування NLP – це аналіз користувацьких відгуків та запитів, який дозволяє не лише оцінити загальну думку про об’єкт, а й виділити конкретні ознаки, що є значущими для рекомендацій. У статті [4] наведено приклади використання семантичного аналізу та класифікації настроїв, що дозволяють виявити позитивні або негативні оцінки певних характеристик продукту й адаптувати рекомендації відповідно до цього.

До основних технік NLP, які використовуються в рекомендаційних системах, належать:

– TF-IDF (Term Frequency-Inverse Document Frequency) – класичний статистичний метод для векторного представлення тексту;

– Word2Vec, GloVe, FastText – моделі для навчання векторних представлень слів, що враховують семантичну подібність;

– BERT (Bidirectional Encoder Representations from Transformers) – сучасна трансформерна модель, яка дозволяє враховувати контекст у всіх напрямках і демонструє високу точність у задачах класифікації, кластеризації та пошуку.

У статті [4] розглядається також роль нейронних мереж, зокрема рекурентних нейромереж та трансформерів, які використовуються для обробки послідовностей слів у запитах і рецензіях користувачів. Завдяки глибокому аналізу послідовностей, такі моделі здатні відстежувати логіку та взаємозв’язки між словами, що критично важливо для точного розуміння запиту.

Окрему увагу дослідники приділяють генерації профілю користувача на основі його лексичних патернів: аналізуючи тексти запитів, коментарів, навіть діалогів із системою, модель може сформувати семантичну карту інтересів, яка потім використовується для побудови персоналізованої стратегії рекомендацій.

Застосування NLP у рекомендаційних системах має низку переваг:

– здатність працювати з неструктурованими даними (текст замість таблиць чи чисел);

– можливість покращення релевантності рекомендацій за рахунок глибшого розуміння змісту;

– адаптація до нових запитів користувача без попередніх оцінок (розв’язання проблеми cold-start);

– пояснюваність – система може формулювати причини рекомендацій у природній мові.

Крім класичних підходів, важливу роль відіграють інтерактивні NLP-механізми, які дозволяють користувачу формувати запити в природній мові, а системі – динамічно інтерпретувати їх і адаптувати рекомендації в реальному часі. Такі функції реалізуються за допомогою моделей діалогової взаємодії, які включають розпізнавання наміру, визначення сутностей та генерацію відповідей. У результаті користувач може отримувати

рекомендації не за допомогою натискання кнопок або виставлення оцінок, а через спілкування з системою у вільній формі.

Сучасні дослідження також акцентують на перспективності пояснювані моделі обробки природної мови – такі, що не лише видають рекомендацію, але й пояснюють її у зрозумілому для людини форматі. Наприклад, система може зазначити: «Ми рекомендуємо вам цю книгу, тому що вона має схожий стиль і тему до тих, які вам уже подобались». Це не лише підвищує довіру користувачів до системи, але й відкриває нові можливості для комбінування рекомендаційних алгоритмів із інтерфейсами штучного інтелекту, орієнтованими на діалог.

Таким чином, методи обробки природної мови розширюють функціональність рекомендаційних систем, дозволяючи їм не лише реагувати на поведінкові патерни, а й активно «розмовляти» з користувачем, створюючи більш гнучкий та інтелектуальний інтерфейс. Це закладає основу для інтеграції таких систем з великими мовними моделями (GPT, BERT, T5), які виконують функції генерації, узагальнення та діалогового аналізу в реальному часі.

1.4 Проблеми при побудові рекомендаційних систем

Попри значний прогрес у розвитку рекомендаційних систем, сучасні рішення стикаються з низкою викликів, які обмежують їхню ефективність та масштабованість. У статті [1] зазначено, що навіть найсучасніші системи мають обмеження, пов'язані з масштабуванням, проблемою «холодного старту», розрідженістю даних та іншими аспектами.

Проблема «холодного старту» є однією з найпоширеніших у рекомендаційних системах і виникає у випадках, коли система не має достатньо інформації про нових користувачів або нові об'єкти. Це особливо критично для колаборативних методів, що ґрунтуються на аналізі історії взаємодій. На рисунку 1.4 зображено поділ проблеми «холодного старту» на

два основні підтипи – користувацький та об’єктний, а також наведено типові підходи до їх подолання, зокрема використання контекстних даних, соціальної інформації та гібридної фільтрації.

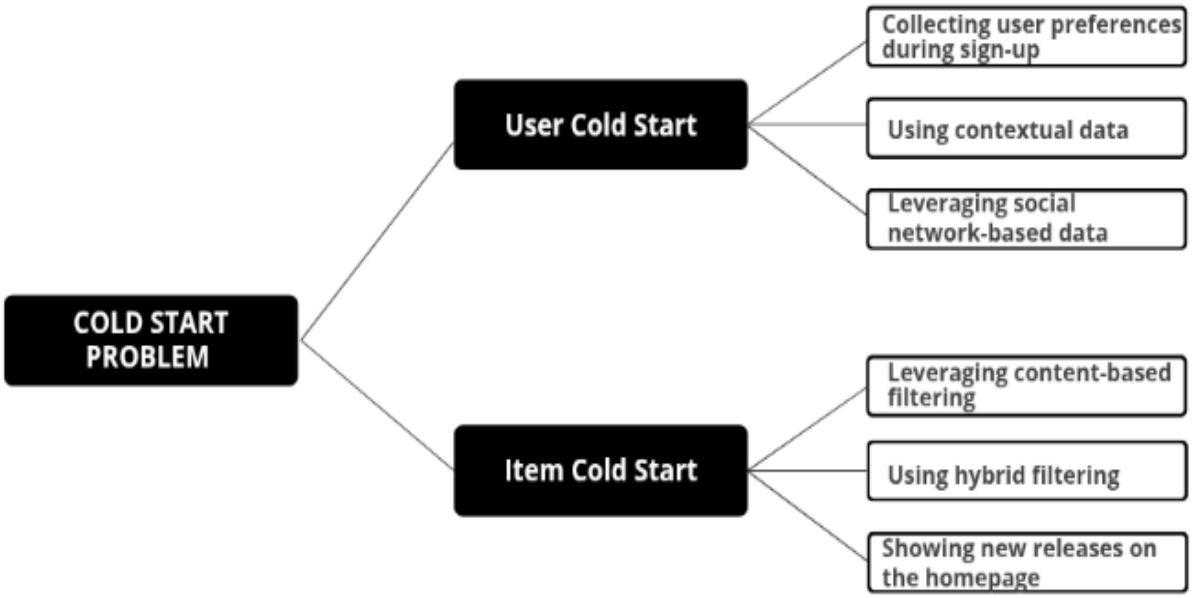

Рисунок 1.4 – Комбінування різних методів у рекомендаційній системі

Розрідженість даних є ще однією суттєвою проблемою. У багатьох системах користувачі взаємодіють лише з невеликою частиною доступного контенту, що призводить до великої кількості відсутніх значень у матриці користувач-об'єкт. Це ускладнює побудову точних моделей рекомендацій.

Масштабованість стає критичною при обробці великих обсягів даних. Зі збільшенням кількості користувачів та об'єктів традиційні алгоритми можуть не справлятися з обчислювальним навантаженням, що впливає на швидкість та якість рекомендацій.

Відсутність стандартизованих метрик оцінювання ускладнює порівняння різних систем. У статті зазначено, що різні дослідження використовують різні метрики, такі як точність, повнота, F1-міра, MAE, RMSE, що ускладнює об'єктивне оцінювання ефективності систем.

Проблеми з конфіденційністю та етичністю набувають все більшого значення. Збір та обробка персональних даних користувачів викликає питання щодо захисту приватності та етичного використання інформації.

Якщо система постійно пропонує користувачу схожий контент, то виникає ефект «інформаційної бульбашки», обмежуючи доступ користувача до різноманітної інформації. Це може знижувати якість користувацького досвіду та обмежувати відкриття нового контенту.

Пояснюваність рекомендацій є важливою для довіри користувачів до системи. Багато сучасних моделей, особливо ті, що базуються на глибокому навчанні, є «чорними ящиками», що ускладнює розуміння причин певних рекомендацій.

Адаптація до змін уподобань користувачів є ще одним викликом. Смаки та інтереси користувачів можуть змінюватися з часом, і система повинна бути здатна швидко реагувати на ці зміни, оновлюючи свої моделі та рекомендації відповідно.

Коли відгуки окремого користувача не збігаються з вподобаннями жодної з груп користувачів, виникає проблема «сірої вівці». У таких випадках система не здатна сформувати релевантні рекомендації на основі колаборативної логіки. Один із варіантів вирішення – використання контентно-орієнтованих моделей, які ґрунтуються не на поведінці інших користувачів, а на індивідуальному профілі та характеристиках об’єктів.

Усі ці виклики вимагають подальших досліджень та розробки нових методів, які дозволять створювати більш ефективні, гнучкі та етичні рекомендаційні системи.

2 СУЧАСНІ ПІДХОДИ ДО РЕАЛІЗАЦІЇ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

2.1 Великі мовні моделі в рекомендаційних системах

В останнє десятиріччя відбулась революція в галузі обробки природної мови завдяки появі великих мовних моделей, таких як GPT, BERT, T5 та їхніх похідних. Ці моделі стали основою великої кількості застосунків, які потребують високоякісного аналізу та генерації тексту.

У міру зростання складності інформаційних потреб користувачів, класичні алгоритми рекомендацій (колаборативні, контентно-орієнтовані) стикаються з низкою обмежень, зокрема в інтерпретації запитів, адаптації до нових сценаріїв та обробки неструктурованих даних. У цьому контексті великі мовні моделі відкривають нові можливості для побудови рекомендаційних систем, здатних ефективно працювати з вхідною інформацією у природній мові, підтримувати діалогову взаємодію, забезпечувати високу якість персоналізації та зменшувати залежність від історичних даних користувача.

Поява великих мовних моделей стала результатом багаторічної еволюції в галузі обробки природної мови та машинного навчання. Початкові спроби застосування текстових моделей у рекомендаційних системах базувалися на простих техніках – таких як мішок слів (bag-of-words), TF-IDF та латентне семантичне індексування (LSI). Вони дозволяли виконувати базову контентну фільтрацію, але не враховували контекст або структуру мови.

Зі зростанням обсягів даних і появою глибокого навчання виникли передумови для створення потужніших моделей, зокрема рекурентних нейромереж (RNN) та їхніх удосконалень, таких як LSTM і GRU, які використовувалися для побудови рекомендацій на основі послідовностей

тексту. Утім, ці моделі мали обмежену здатність до масштабування і довготривалого контекстного аналізу.

Прорив стався у 2018 році з публікацією моделі BERT, яка поклала початок епохи трансформерів – архітектури, здатної ефективно обробляти довгі контексти завдяки механізму self-attention. Надалі були створені більш масштабні та продуктивні моделі: GPT-2 (2019), GPT-3 (2020), T5, PaLM, LLaMA, які навчаються на трильйонах токенів і демонструють здатність до узагальнення, генерації тексту, виведення знань та ведення діалогу.

Починаючи з 2021 року, дослідники почали інтегрувати LLM у рекомендаційні системи. Згідно з оглядом статті [5], спочатку моделі використовувались як інтерфейсні агенти, які розпізнають запити користувача у вільній формі. Надалі почали розробляти підходи, де LLM стає ядром усієї рекомендаційної системи – від обробки запиту до ранжування об’єктів та генерації пояснень. У новітніх дослідженнях акцент робиться на zero-shot та few-shot можливостях моделей, що дозволяє їм функціонувати без великих наборів навчальних даних, адаптуватися до нових предметних галузей та підтримувати персоналізацію без прямої історії взаємодій.

2.1.1 Обмеження та проблеми LLM

Незважаючи на стрімкий розвиток і багатообіцяючі результати, впровадження великих мовних моделей у рекомендаційні системи супроводжується низкою обмежень і відкритих проблем. Однією з головних технічних перешкод є високі обчислювальні витрати: моделі з мільярдами параметрів, зокрема GPT-3, PaLM або LLaMA, потребують значних ресурсів як для навчання, так і для інференсу (процесу генерації відповіді). У випадку рекомендаційних систем, де швидкість реакції є критичною, це може ускладнювати інтеграцію моделей у реальні сервіси. Додатково постає проблема недостатньої персоналізації – за замовчуванням LLM не мають

доступу до індивідуального контексту користувача. Хоча сучасні підходи, такі як налаштування запитів або поєднання з векторними базами знань, частково зменшують цю проблему, повноцінна персоналізація залишається складним завданням.

Ще однією проблемою є феномен катастрофічного забування, коли донавчання моделі на нових даних призводить до втрати раніше набутого знання. Це особливо актуально в динамічних середовищах, де переваги користувачів змінюються з часом і система має бути здатною адаптуватися без шкоди основній функціональності. Важливою проблемою є також брак прозорості – LLM функціонують як «чорні скриньки», що ускладнює інтерпретацію результатів та знижує довіру користувачів до рекомендацій. Хоча генеративні моделі можуть надавати пояснення у формі природної мови, такі пояснення не завжди є достовірними чи послідовними.

Окрему увагу дослідники звертають на схильність LLM до галюцинацій – тобто генерації відповідей, які звучать переконливо, але не мають фактичного підґрунтя. У контексті рекомендацій це може спричинити появу нерелевантних або навіть вигаданих об'єктів. Крім того, LLM демонструють схильність до відтворення упереджень, що містяться в тренувальних даних, що може мати наслідки для етичності й об’єктивності рекомендацій. Також не слід забувати про складнощі технічної інтеграції: застосування LLM потребує нових архітектурних рішень і часто не узгоджується з традиційними підходами до реалізації рекомендаційних платформ.

2.1.2 Перспективи розвитку LLM

Перспективи застосування великих мовних моделей у сфері рекомендаційних систем залишаються обнадійливими. Розвиток ефективніших, менш ресурсоємних моделей, зокрема через дистиляцію знань та архітектурну оптимізацію (наприклад, LLaMA, DistilGPT), сприяє

зменшенню бар’єру для їх впровадження у реальні системи. Окрім цього, поява інструментів, що підтримують локальне виконання моделей, розширює можливості персоналізації без порушення конфіденційності користувачів.

Іншим перспективним напрямком є інтеграція LLM з багатомодальними джерелами даних. Рекомендаційні системи нового покоління потенційно здатні об’єднувати текстову, візуальну, звукову та поведінкову інформацію для формування комплексного розуміння користувацьких інтересів. Це дозволить виходити за межі лінійної фільтрації та забезпечити глибшу адаптацію до контексту та інтенцій користувача. Перспективним залишається також напрям explainable AI. Це розвиток моделей, здатних не лише генерувати рекомендації, а й надавати зрозумілі пояснення до них у природній мові, що особливо важливо в освітніх, медичних та правових системах.

Важливо також зазначити роль відкритих дослідницьких ініціатив та спільнот, які активно досліджують поєднання LLM з методами колаборативної фільтрації, векторного пошуку та підходами zero-shot або few-shot навчання. Такий міждисциплінарний підхід сприяє створенню більш гнучких і стійких архітектур, що можуть адаптуватися до нових умов та запитів користувачів.

Отже, великі мовні моделі репрезентують новий етап в еволюції рекомендаційних систем – від механістичних алгоритмів до контекстно-орієнтованих, діалогових і семантично чутливих агентів. Їхнє подальше дослідження та впровадження мають всі підстави істотно трансформувати підходи до персоналізації в цифровому середовищі.

2.2 Напрями розвитку рекомендаційних систем

Сучасні рекомендаційні системи продовжують еволюціонувати у напрямі гнучкішої взаємодії з користувачем, що зумовлює потребу в

переосмисленні їхньої архітектури. На зміну класичним підходам, побудованим на колаборативній фільтрації та аналізі контенту, приходять мовно-орієнтовані, контекстно-чутливі та діалогові системи, засновані на великих мовних моделях, таких як GPT.

В умовах цієї трансформації важливо усвідомлювати, які саме компоненти змінюються, які підходи залишаються актуальними, а які поступово витісняються. У статті [6] запропоновано концептуальну модель, що класифікує основні напрями розвитку рекомендаційних систем та структурує їх за рівнем складності, інтелектуальності та здатності до персоналізованої взаємодії.

На рисунку 2.1 представлено модель розвитку та класифікації рекомендаційних систем за ключовими напрямами, що охоплюють як традиційні, так і сучасні підходи на основі великих мовних моделей. Модель структурована за чотирма головними осями: традиційні системи (Traditional RS), новітні тенденції (Emerging Trends), системи на основі GPT-чат-ботів (GPT-Based Chatbots) та горизонтальні тематичні напрями, що включають прикладне застосування, налаштування, переваги або недоліки та майбутні виклики.

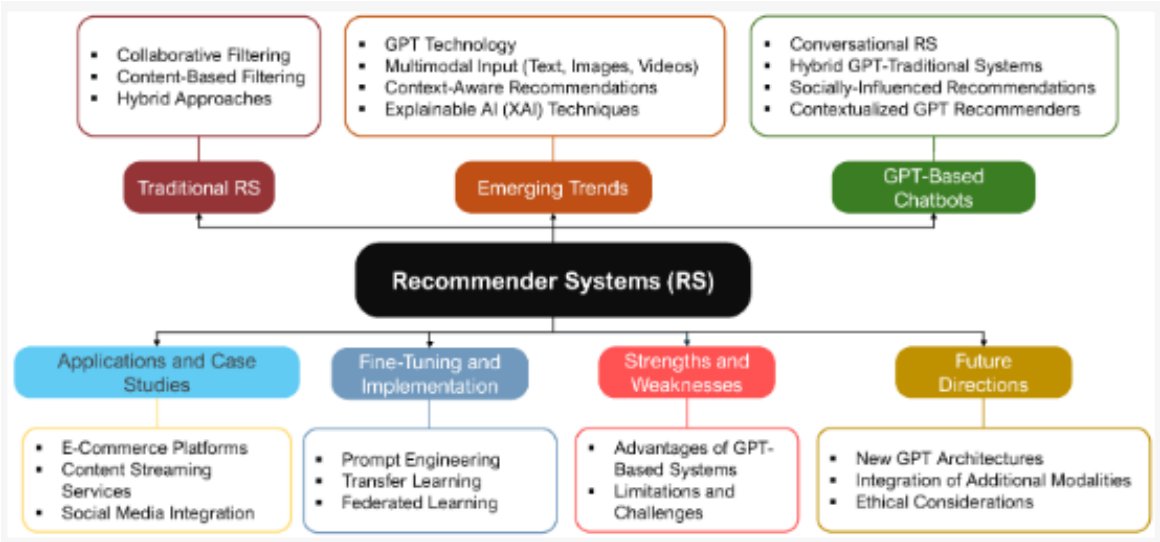

Рисунок 2.1 – Основні напрями розвитку рекомендаційних систем

У сегменті Traditional RS виділяються три класичні підходи: колаборативна фільтрація, контентно-орієнтовані системи та гібридні моделі. Ці підходи домінували в рекомендаційних системах протягом останніх десятиліть і добре масштабуються, однак мають обмеження в контекстній чутливості та пояснюваності.

Категорія Emerging Trends включає нові технології, що доповнюють або розширюють традиційні методи: GPT-моделі, мультимодальні входи, контекстно-чутливі рекомендації та техніки Explainable AI. Ці підходи націлені на глибше розуміння намірів користувача та забезпечення прозорості результатів.

GPT-Based Chatbots відображають новий етап еволюції систем, де рекомендації надаються у формі семантичної взаємодії через діалог, а самі системи можуть бути контекстуальними, соціально обґрунтованими та поєднувати традиційні та сучасні моделі. Це дозволяє системі пояснювати логіку вибору та адаптуватися до стилю спілкування користувача.

Горизонтальні блоки під діаграмою – Applications and Case Studies, Fine-Tuning, Strengths and Weaknesses, Future Directions – деталізують практичні аспекти: від платформ застосування до етичних викликів, необхідності трансферного навчання та проблем галюцинацій.

Загалом, рисунок ілюструє перехід від алгоритмічної логіки до діалогової та пояснюваної персоналізації, що є ключовим напрямом розвитку рекомендаційних систем нового покоління.

2.2.1 GPT-моделі в системах рекомендацій

Останні роки ознаменувалися стрімким розвитком великих мовних моделей, зокрема таких, як GPT, що мають потужний потенціал для застосування в системах інтелектуальних рекомендацій. Їхні можливості виходять далеко за межі класичних NLP-інструментів, дозволяючи не лише обробляти текстові дані, а й розуміти запити користувачів у природній мові,

генерувати персоналізовані відповіді та адаптувати рекомендації до контексту діалогу.

На відміну від традиційних моделей, які потребують чіткого подання запиту або структури даних, LLM здатні аналізувати неструктуровану інформацію – наприклад, опис книжки, вільну форму побажань користувача або навіть його емоційний тон. Завдяки попередньому навчанню на масштабних корпусах текстів, GPT може доповнювати запити, розуміти приховані наміри, пропонувати варіанти навіть при мінімальній інформації.

Сучасні дослідження вказують на те, що інтеграція LLM у рекомендаційні системи може реалізовуватися у двох основних сценаріях:

– LLM як інтерфейс для взаємодії з користувачем – користувач формує запит у природній мові, LLM інтерпретує його, здійснює фільтрацію або генерацію списку рекомендованих об’єктів;

– LLM як ядро рекомендаційного модуля – модель аналізує базу знань (наприклад, каталог книг у JSON-форматі), обчислює релевантність об’єктів до запиту і формує відповідь із поясненням.

Одна з переваг GPT-подібних моделей – це адаптивність без явного донавчання: система може працювати в zero-shot або few-shot режимі, тобто виконувати завдання без великої кількості прикладів, лише на основі формулювання завдання у вигляді текстового підказу. Це відкриває нові можливості для застосування в реальних системах рекомендацій, зокрема для книжкових платформ, де користувачі не завжди готові проходити реєстрацію чи надавати багато даних.

Крім того, великі мовні моделі створюють можливість реалізації пояснюваних рекомендацій. LLM можуть формулювати пояснення до кожної поради у зрозумілій для користувача формі. Наприклад: «Ця книга рекомендована вам, оскільки в описі вказано елементи наукової фантастики та подорожей у часі – схожі риси були у книгах, які вас цікавили раніше».

Ілюстрацією практичного використання великої мовної моделі GPT у системі рекомендацій є приклад, поданий у статті [5], де розглядається

взаємодія між користувачем і LLM у межах задач персоналізованої рекомендації. На рисунку 2.2 представлено узагальнену схему використання великих мовних моделей (LLM), зокрема GPT-подібних архітектур, у контексті реалізації систем рекомендацій. Ілюстрація демонструє, як LLM здатні виконувати декілька ключових завдань у рамках одного рекомендаційного процесу – від аналізу історії переглядів до генерації пояснень у природній мові.

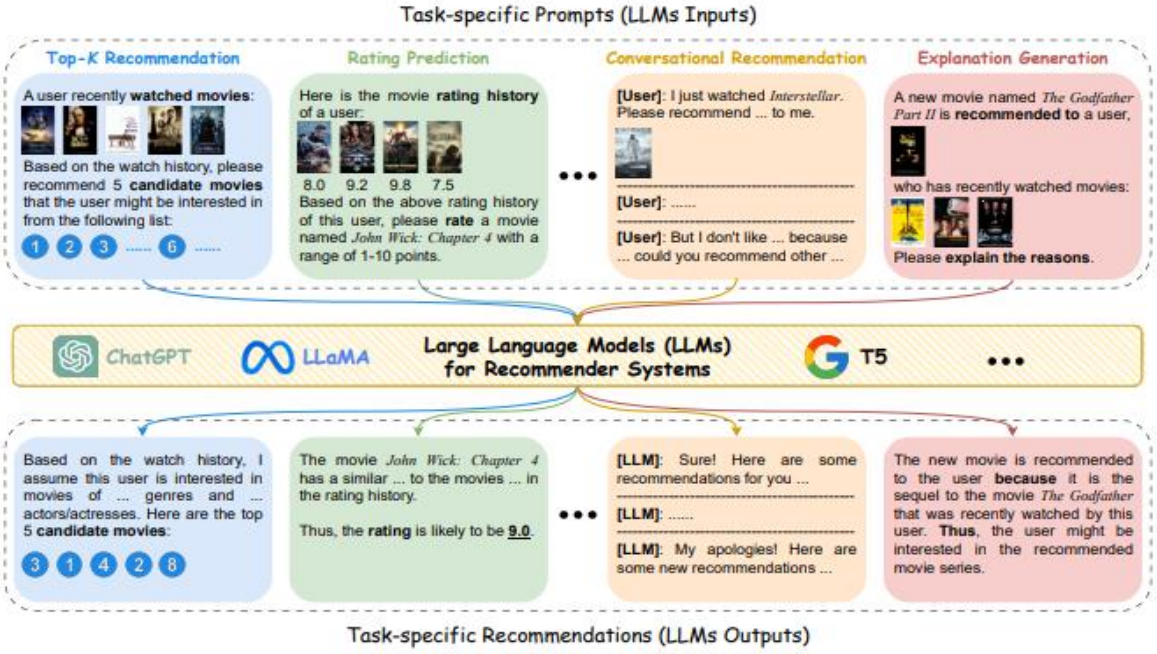

Рисунок 2.2 – Застосування LLMs для виконання типових задач рекомендації

У верхній частині рисунка зображено вхідні дані (Task-specific Prompts): чотири типи вхідних підказок, сформульованих у природній мові. Кожна з них відповідає певному сценарію використання системи:

– Top-K Recommendation: користувач надає інформацію про нещодавно переглянуті фільми. Підказка просить модель запропонувати кілька релевантних об’єктів;

– Rating Prediction: модель отримує історію оцінок, яку користувач виставив іншим фільмам. Підказка формулює прохання передбачити оцінку для нового фільму, базуючись на попередніх рейтингах;

– Conversational Recommendation: діалогова взаємодія, коли користувач ініціює бесіду з системою і отримує рекомендації у відповідь;

– Explanation Generation: система повинна пояснити, чому вона рекомендує певний об’єкт.

У центральній частині схеми розміщено умовне зображення мовної моделі, що обробляє запити.

У прикладі згадуються GPT та інші моделі, як представники великих мовних моделей, здатних до генерації тексту, аналізу контексту, а також адаптивної взаємодії з користувачем.

Нижній блок демонструє результати генерації для кожного з вхідних сценаріїв:

– Top-K Recommendation Output: модель формує список з п’яти релевантних фільмів, враховуючи жанри, акторів та попередні вподобання користувача;

– Rating Prediction Output: виводиться оцінка й коротке пояснення, що певний фільм подібний до вже оцінених;

– Conversational Response: модель відповідає у форматі діалогу з урахуванням уточнень користувача;

– Explanation Generation Output: генерується пояснення.

Даний рисунок ілюструє підхід, за якого LLM повністю заміщує традиційні компоненти рекомендаційної системи. Це підтверджує гнучкість і потенціал GPT-подібних моделей як універсальних мультимодальних агентів у персоналізованих системах.

Незважаючи на очевидні переваги, застосування LLM у рекомендаційних системах має і свої проблеми. До них належать високі обчислювальні витрати, ризики галюцинацій – генерація неправдивих або вигаданих рекомендацій. Також недоліками є непрозорість прийняття

рішень при використанні моделей «чорної скриньки» та потреба в адаптації під специфіку предметної галузі.

Однак, враховуючи динаміку розвитку LLM, усе більше дослідників та компаній схиляються до думки, що майбутнє рекомендаційних систем за поєднанням мовних моделей із класичними алгоритмами фільтрації та структурованими базами даних.

3 ПОПЕРЕДНЄ ПРОЄКТУВАННЯ ТА ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

3.1 Опис функціональності застосунку

Розроблена система рекомендаційного типу призначена для надання персоналізованих порад щодо вибору книжкової продукції з наперед визначеного асортименту. Взаємодія з користувачем реалізується через інтерфейс чат-бота, побудованого на базі великої мовної моделі GPT-4-turbo із залученням інструментів Assistant API від OpenAI.

Основною функцією програмного засобу є обробка запитів користувачів природною мовою та формування релевантних рекомендацій на основі вбудованого каталогу книжок, представленого у форматі JSON-файлу. У процесі взаємодії користувач має змогу сформулювати побажання щодо жанру, тематики, настрою, цільової вікової аудиторії, кількості сторінок тощо. Система інтерпретує запит, аналізує вміст локального сховища даних та генерує відповідь у вигляді короткої поради або добірки відповідних позицій з каталогу.

Система володіє низкою ключових функціональних можливостей, спрямованих на забезпечення ефективної взаємодії з користувачем у межах рекомендаційного сервісу. Насамперед, вона здатна аналізувати текстові запити, розпізнаючи в них основні ознаки, які вказують на побажання щодо майбутньої книжкової рекомендації. На основі цих ознак система здійснює фільтрацію доступного книжкового масиву, орієнтуючись на жанрові, тематичні, вікові та інші релевантні характеристики.

Результатом такої обробки є формування відповіді, що містить лаконічний, але змістовний опис однієї або кількох книг, які найбільше відповідають запиту. У разі, якщо жодна книга не задовольняє критерії пошуку, система повідомляє про це користувача та, за можливості, пропонує альтернативні варіанти з каталогу. Важливою особливістю є

також забезпечення доброзичливої, чіткої й структурованої комунікації, яка здійснюється в межах заздалегідь визначеного сценарію взаємодії.

Окрім діалогового інтерфейсу з вбудованим AI-асистентом, програмна система також передбачає традиційний механізм взаємодії з користувачем у вигляді класичного каталогу книжок. Цей підхід орієнтований на користувачів, які вже мають чітке уявлення про бажану книгу або її параметри. У межах цієї опції реалізовано можливість швидкої навігації за допомогою базових фільтрів, зокрема за жанром, автором, алфавітом чи іншими метаданими. Також користувачеві доступна функція пошуку, яка дозволяє оперативно знаходити потрібну позицію за ключовими словами.

Таке розмежування функціональних режимів на асистента з генеративною логікою та звичайний каталог задовольняє різні сценарії поведінки користувачів і забезпечує більшу гнучкість у користуванні системою. В результаті покращується загальна зручність користування, адже кожен відвідувач має змогу обрати оптимальний для себе спосіб отримання рекомендацій або інформації про наявний книжковий асортимент.

Система виконує функцію інтелектуального консультанта книжкового магазину, обмеженого локальним переліком книжок, без використання зовнішніх джерел чи довільного генеративного контенту. Уся логіка побудована навколо інтеграції мовної моделі з локальним сховищем даних у форматі JSON.

3.2 Архітектура системи

Опис архітектури відображає загальну структуру програмного засобу та принципи реалізації його функціональних можливостей. Архітектурна схема дозволяє зрозуміти, яким чином організовано компоненти системи, як саме відбувається взаємодія між ними, а також якою є логіка обробки

запитів користувача. Такий опис є необхідною передумовою для ефективного проєктування, масштабування, супроводу та подальшої модифікації програмного продукту.

Система побудована на основі клієнт-серверної архітектури, де окремі компоненти відповідають за різні аспекти її функціонування. Користувацький інтерфейс виконує роль посередника між користувачем і внутрішньою логікою застосунку, забезпечуючи як класичну навігацію по каталогу книжок із можливістю пошуку та фільтрації, так і інтерактивну взаємодію з AI-асистентом, який пропонує персоналізовані рекомендації.

Серверна частина виконує обробку запитів, отриманих із клієнтської сторони: у першому випадку вона відповідає за фільтрацію локального каталогу, а у другому – за формування інструкцій для мовної моделі.

Джерелом інформації в обох сценаріях слугує локальне сховище у форматі JSON, що містить структуровані дані про доступні книги, включаючи метадані, необхідні для фільтрації й аналізу. Генерація відповідей здійснюється через Assistant API на базі GPT-4-turbo, який функціонує в межах заздалегідь визначеного сценарію й використовує лише надані дані з локального каталогу.

На рисунку 3.1 наведено узагальнену взаємодію між компонентами системи. Схема ілюструє два сценарії взаємодії користувача з системою: через класичний каталог і за допомогою AI-асистента. Для кожного з варіантів показано відповідний потік даних – від моменту ініціації користувацької дії до формування відповіді, що базується на локальному сховищі книжок. Така візуалізація дає змогу чітко простежити розмежування відповідальностей між клієнтською, серверною та AI-компонентами.

Як показано на рисунку 3.1, взаємодія користувача із системою починається з графічного інтерфейсу, що є складовою клієнтської частини. Після завантаження сторінки користувачеві пропонується вибір режиму взаємодії – з використанням AI-асистента або без нього.

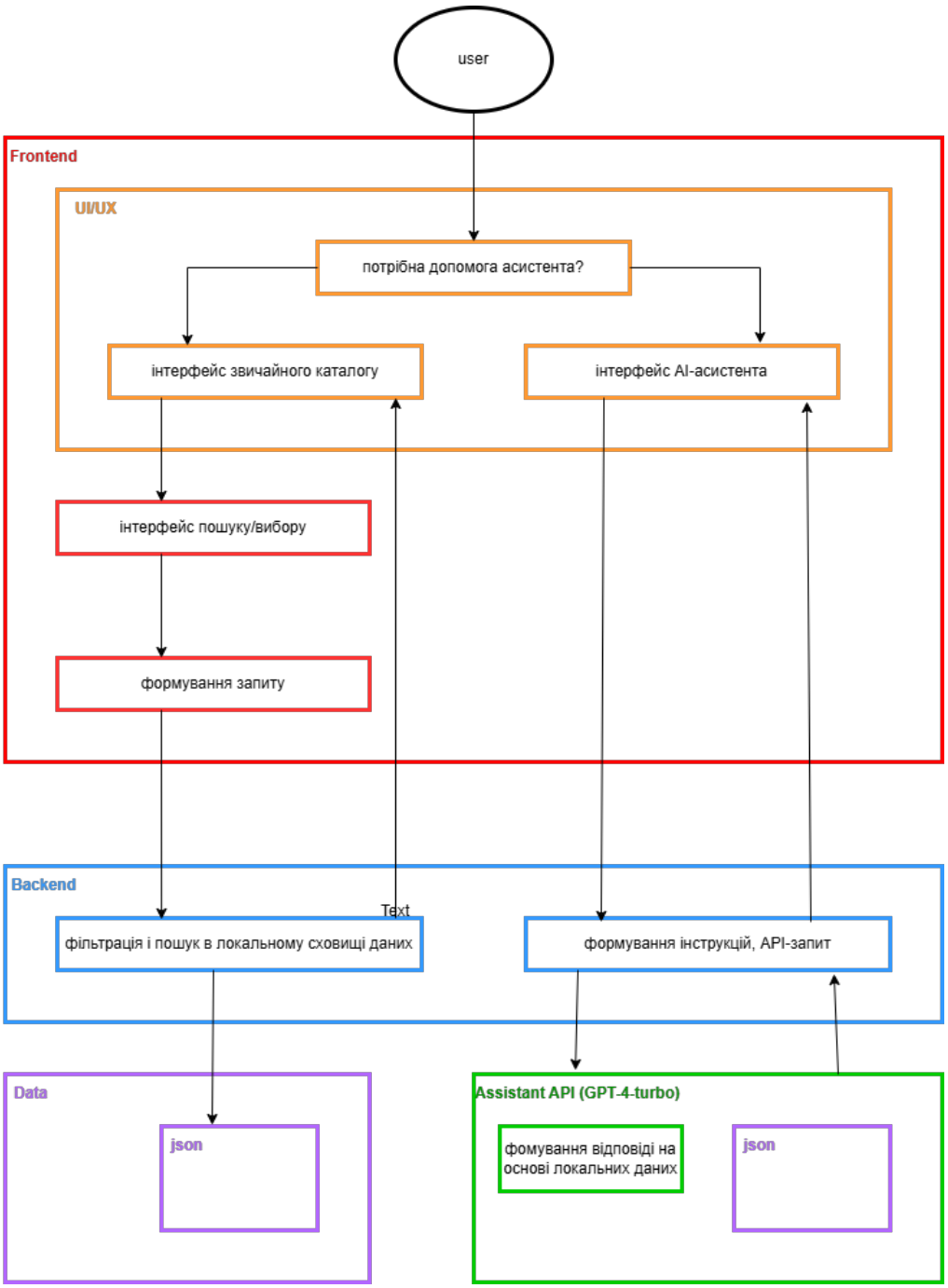

Рисунок 3.1 – Архітектурна схему програмного засобу

Система ініціює діалог, у якому уточнюється, чи потребує користувач допомоги у виборі книжки. У разі відмови за умови вибору класичного сценарію користувач потрапляє до інтерфейсу звичайного каталогу, де йому доступний візуальний перелік книжок та набір фільтрів для зручної навігації. Інтерфейс підтримує сортування й пошук за жанром, автором,

алфавітом, кількістю сторінок, віковою категорією та іншими параметрами, що відображаються у формі випадаючих списків або текстових полів для введення запиту.

Після вибору критеріїв запит структурується у фронтенді та передається до серверної частини, де потрапляє до модуля, відповідального за фільтрацію та пошук у локальному сховищі даних. Це сховище представлено у форматі JSON-файлу, що містить каталог книжок із відповідними метаданими: назвою, автором, коротким описом, жанром, кількістю сторінок, цільовою аудиторією тощо. Система виконує пошук відповідно до заданих параметрів, формує відповідь у вигляді масиву релевантних книжок і повертає її у клієнтську частину, де результати відображаються у зрозумілому для користувача вигляді – наприклад, у вигляді карток або списку.

Якщо ж користувач обирає взаємодію з AI-асистентом, запит надходить до відповідного інтерфейсу, що імітує формат діалогу у вигляді чат-повідомлень. У цьому випадку система передає запит до бекенду, де відбувається формування спеціальної інструкції для мовної моделі. Ця інструкція включає не лише сам запит користувача, сформульований у природній мові, але й контекстну інформацію на основі локального JSON-каталогу, яка допомагає моделі краще зрозуміти межі дозволених відповідей.

Сформований промпт надсилається до Assistant API, який, маючи доступ до структури каталогу, генерує відповідь виключно на основі наданих локальних даних. Відповідь повертається через бекенд у фронтенд, де вона відображається в інтерфейсі AI-асистента у формі повідомлення.

Таким чином, архітектурна схема демонструє чіткий розподіл логіки між компонентами системи. Фронтенд відповідає за користувацьку взаємодію та формування запитів, бекенд – за обробку, фільтрацію та маршрутизацію, а Assistant API виконує інтелектуальну генерацію відповідей у межах дозволеного контексту. Такий підхід забезпечує

гнучкість, масштабованість і простоту підтримки системи, водночас підвищуючи якість персоналізованих рекомендацій.

3.3 Вибір технологій

Під час розробки програмного забезпечення було обрано класичний клієнт-серверний стек, який поєднує сучасні web-технології з можливістю інтеграції мовної моделі через зовнішнє API. Такий підхід забезпечує ефективне розділення відповідальностей між інтерфейсною та серверною частинами, дозволяє легко масштабувати систему, а також забезпечує адаптивність до різних сценаріїв користування.

У якості мови програмування як на клієнтському, так і на серверному боці було обрано JavaScript – мову з широкою підтримкою, активною спільнотою та великою кількістю бібліотек і фреймворків. Це дозволяє досягти високої швидкості розробки та легко інтегрувати зовнішні сервіси, зокрема мовну модель від OpenAI.

3.3.1 Фронтендна частина системи

Клієнтська частина програмного забезпечення реалізована з використанням React – сучасної JavaScript-бібліотеки для побудови інтерактивних інтерфейсів користувача. Вибір React зумовлений його компонентною структурою, високою продуктивністю завдяки віртуальному DOM, активною підтримкою спільноти та широким екосистемним оточенням. React стандартом у галузі фронтенд-розробки, що забезпечує надійність і масштабованість рішень.

Для запуску проєкту та організації зручного середовища розробки застосовано Vite – інструмент нового покоління для збірки проєктів. На відміну від традиційних рішень на зразок Webpack, Vite забезпечує миттєвий запуск, гаряче оновлення модулів та значно швидшу компіляцію.

Це дозволяє суттєво скоротити час розробки, що є критично важливим у навчальних проєктах.

Візуальний інтерфейс оформлено з використанням Tailwind CSS – утилітарного CSS-фреймворку, який забезпечує швидке і контрольоване створення стилів без необхідності писати традиційний CSS. Tailwind дозволяє досягти високого рівня адаптивності, доступності та візуальної узгодженості інтерфейсу при мінімальному обсязі коду.

Для реалізації чатоподібного інтерфейсу з AI-асистентом було використано бібліотеку «react-simple-chatbot», яка забезпечує побудову діалогового інтерфейсу у вигляді чату. Це рішення дозволило легко реалізувати сценарій покрокової взаємодії з користувачем, керованої логікою програми, та інтегрувати відповіді GPT-моделі у відповідну форму подачі.

Порівняно з альтернативними фреймворками, такими як Angular чи Vue.js, React виявився більш придатним для цього проєкту завдяки простоті входження, активному використанню у сучасних розробницьких екосистемах і кращій підтримці бібліотек для роботи з формами, запитами, маршрутизацією та анімаціями. Angular, хоч і є потужним фреймворком, передбачає більш складну структуру та криву навчання, а Vue хоча й простий, має дещо меншу екосистему та спільноту в порівнянні з React.

Таким чином, вибір React у поєднанні з Vite та Tailwind забезпечив оптимальний баланс між швидкістю розробки, гнучкістю, масштабованістю інтерфейсу та зручністю користування системою.

3.3.2 Серверна логіка обробки запитів

Серверна логіка програмного забезпечення реалізована за допомогою зв’язки Node.js та Express.js, що є одним із найпопулярніших та найефективніших підходів для розробки RESTful API в екосистемі

JavaScript. Вибір цієї технології зумовлений наявністб великою кількості переваг.

Node.js – це середовище виконання JavaScript на стороні сервера, яке побудоване на базі V8-двигуна від Google. Завдяки неблокуючій архітектурі Node.js забезпечує високу продуктивність при обробці великої кількості одночасних запитів, що є важливим для системи, яка обробляє запити як від AI-асистента, так і від традиційного каталогу.

Використання Express.js дозволяє легко створювати маршрути та обробники запитів. Express є надзвичайно гнучким і не накладає жорстких структурних обмежень, що дозволяє адаптувати архітектуру під специфіку конкретного проєкту. Простота налаштування і розширення робить його зручним інструментом для швидкої розробки.

Серверна частина також реалізує два основні завдання: обробку запитів на фільтрацію даних із JSON-файлу для класичного пошуку та формування структурованих інструкцій для GPT-моделі через OpenAI Assistant API. Для цього використовуються базові можливості Express разом із модулями для роботи з файловою системою, обробки HTTP-запитів, а також обробки JSON-структур.

Загалом, використання Node.js і Express.js як основи для серверної частини є виправданим з точки зору продуктивності, простоти розгортання, гнучкості та сумісності з іншими частинами системи, що реалізовані на JavaScript та React.

3.3.3 Вибір Assistant API як засобу доступу до GPT-моделі

Ключовим компонентом інтелектуальної підсистеми програмного продукту є вбудований AI-асистент, реалізований шляхом інтеграції з Assistant API, що надається компанією OpenAI. Цей інтерфейс забезпечує доступ до мовної моделі GPT-4-turbo, яка використовується для генерації

змістовних, контекстуально релевантних відповідей на основі запитів користувачів природною мовою.

На відміну від стандартного способу інтеграції моделей GPT, де розробник самостійно повинен формувати повну історію діалогу для кожного запиту, Assistant API забезпечує можливість зберігання контексту у вигляді окремих сесій. Це значно спрощує реалізацію послідовного спілкування з користувачем, дозволяючи асистенту враховувати попередні звернення без додаткових зусиль з боку розробника.

У той час як традиційні інтеграції потребують вручну передавати весь попередній контекст, Assistant API автоматично підтримує діалогову пам’ять, що підвищує стабільність, зменшує ризик помилок та робить систему більш масштабованою і зручною для підтримки.

Однією з ключових переваг використання OpenAI Assistant API є можливість підключення зовнішніх ресурсів у вигляді файлів та додаткових інструментів, які асистент може використовувати під час діалогу. Це забезпечує значно більшу функціональну гнучкість у порівнянні зі стандартними API-запитами до моделей GPT.

У межах розроблюваного застосунку така можливість була використана для підключення локального каталогу книжок у форматі JSON. Завдяки цьому асистент отримав доступ до структурованої інформації про всі наявні позиції в магазині – без потреби дублювати ці дані у промптах або програмно передавати їх при кожному зверненні. Такий підхід підвищує ефективність відповіді, зменшує розмір запитів і забезпечує стабільну логіку взаємодії.

Крім того, Assistant API дозволяє підключати спеціалізовані функції, які можуть бути викликані мовною моделлю для виконання конкретних дій. У перспективі це відкриває можливості для масштабування системи: наприклад, додавання пошуку по базі даних, підключення зовнішніх API, формування замовлення тощо. Це відрізняє Assistant API від стандартного

підходу, де вся логіка взаємодії має бути оброблена виключно на стороні сервера або через інтерпретацію відповіді моделі.

Використання файлів і інструментів у рамках Assistant API дозволяє створювати більш адаптивні, структуровані й масштабовані рішення для інтерактивної взаємодії з користувачами.

Ще одним важливим аспектом інтеграції з OpenAI Assistant API є можливість чітко задати правила поведінки моделі через системні інструкції. Це дозволяє сформувати межі її функціональності відповідно до поставленої задачі, зокрема – уникати вигаданої інформації або відповідей, що не стосуються предметної галузі.

У межах розробленої системи було створено детальний промпт, що моделює поведінку віртуального консультанта книжкового магазину. У ньому зазначено, що асистент повинен працювати виключно з інформацією, що міститься у завантаженому JSON-файлі, не вигадувати назви книг або їхніх описів, не відповідати на запитання загального характеру і завжди підкреслювати, що його поради базуються на наявному каталозі магазину. Також промпт вказує на необхідність підтримувати дружній, але лаконічний стиль спілкування та формулювати відповіді у зрозумілому вигляді.

Такий рівень деталізації дозволяє досягти високого ступеня керованості відповіді моделі, зменшуючи ризик некоректної або надмірної генерації. Це також є важливою перевагою над іншими підходами інтеграції моделей GPT, де подібна поведінка потребує складної логіки фільтрації результатів на рівні бекенду. Добре структуровані системні інструкції виступають критичним елементом забезпечення надійної, послідовної та контрольованої взаємодії між користувачем і мовною моделлю.

Однією з ключових функціональних можливостей Assistant API є підтримка механізму збереження контексту в межах так званих сесій (threads). Кожна сесія дозволяє зберігати історію запитів і відповідей між користувачем та мовною моделлю, що значно покращує якість діалогу та забезпечує ефект «продовження розмови». Завдяки цьому асистент має

змогу не лише відповідати на окремі повідомлення, а й враховувати попередні звернення користувача, формуючи більш узгоджені й логічно послідовні рекомендації.

На відміну від класичного підходу, де кожне звернення обробляється незалежно, використання threads дає змогу моделювати діалог із постійним станом. У контексті розробленої системи ця можливість використовується для збереження індивідуального запиту користувача, сформованого на початку взаємодії, та забезпечення його присутності в подальших повідомленнях, навіть якщо користувач змінює або доповнює критерії пошуку.

Технічно, кожен новий користувач або новий сеанс ініціює створення окремого об’єкта thread, у якому накопичується інформація про запити та відповіді. У подальшому цей об’єкт передається до API разом із новим зверненням, що дозволяє зберігати безперервність розмови.

Використання такого підходу підвищує гнучкість системи та наближає поведінку асистента до людської моделі ведення діалогу, що, в свою чергу, покращує користувацький досвід та точність рекомендацій.

При інтеграції великих мовних моделей у рекомендаційні системи одним із важливих викликів є запобігання генерації недостовірних або вигаданих даних. Моделі на зразок GPT можуть створювати переконливі тексти навіть у випадках, коли запитана інформація відсутня, що є неприйнятним у системах із жорстко обмеженим джерелом даних – таких, як рекомендаційний сервіс з фіксованим каталогом книг.

У розробленій системі ця проблема вирішується шляхом поєднання трьох підходів. По-перше, застосовується жорстко сформульований промпт, який чітко інструктує модель обмежувати відповіді лише тими книгами, що наявні в каталозі. У тексті інструкції прямо забороняється вигадувати книжки або відповідати на запити, які виходять за межі локальної бази даних. Це дозволяє суттєво зменшити ймовірність галюцинацій. По-друге, модель працює в умовах ізольованого середовища, в якому доступні лише

попередньо завантажені файли і не використовуються зовнішні джерела знань. Це означає, що навіть за відсутності чітких обмежень у запиті, GPT не має технічної можливості звертатися до інформації, якої немає в каталозі. По-третє, в межах інтеграції Assistant API передбачено фіксацію поведінки моделі за допомогою thread та instructions, де інструкції є статичними і передаються автоматично при кожному зверненні до API. Таким чином, забезпечується сталість поведінки асистента та унеможливлюється спонтанна зміна тону, стилю або обсягу відповідей.

Завдяки поєднанню цих технічних і лінгвістичних засобів вдається досягнути максимальної відповідності відповідей фактичному вмісту книжкового каталогу, що є критично важливим для забезпечення надійності та довіри до системи з боку кінцевого користувача.

3.3.4 Формат і структура даних у системі

Функціонування системи рекомендацій базується на структурованому локальному сховищі книжок, реалізованому у форматі JSON-файлу. У даному каталозі кожен об’єкт описує окрему книжкову позицію з набором ключових атрибутів: назва, автор, жанр, теми, настрій, цільова вікова аудиторія, кількість сторінок, мова, рік видання, основні персонажі та розширений опис.

Саме ця інформація є єдиним джерелом знань для асистента, що працює в межах Assistant API. Завдяки JSON-структурі дані легко обробляються як класичними методами фільтрації в межах інтерфейсу каталогу, так і шляхом інтерпретації запиту мовною моделлю у сценарії з AI-асистентом. Модель отримує доступ до цього файла у вигляді прикріпленого ресурсу, що робить можливою його контекстну обробку при генерації відповіді.

Окрему перспективу становить застосування векторного представлення книжок для реалізації семантичного пошуку. Векторизація

дозволяє перетворити текстові описи книжок на вектори, що дає змогу зіставляти запити користувачів з релевантними позиціями на основі близькості в семантичному просторі. Попри те, що в поточній реалізації система не використовує векторне сховище, архітектура дозволяє в майбутньому інтегрувати Vector Store (наприклад, на базі OpenAI embeddings або інших аналогів), що значно розширить можливості рекомендаційного модуля.

Таким чином, JSON-каталог виконує не лише роль джерела даних, але й слугує основою для масштабування системи в напрямі інтелектуального пошуку та персоналізованих рекомендацій.

4 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ

Початковим етапом взаємодії користувача із системою є головний екран, зображений на рисунку 4.1. Його головна мета – ініціювати діалог та запропонувати два альтернативні шляхи взаємодії: класичний пошук по каталогу або персоналізовану допомогу від AI-асистента. Користувачеві ставиться просте запитання з можливістю обрати один з двох варіантів відповіді.

Це не лише візуальний елемент, а частина логіки, реалізованої у фронтенді. Залежно від обраного варіанту, система перенаправляє користувача або до інтерфейсу класичного каталогу, де доступні фільтрація і пошук, або до інтерфейсу AI-асистента, що дозволяє взаємодіяти з мовною моделлю GPT на основі локального каталогу.

Рисунок 4.1 – Головний екран з логічним розгалуженням сценаріїв взаємодії

На відміну від більшості цифрових сервісів, ця система не вимагає авторизації, збору персональних даних або налаштування профілю. Такий

підхід зумовлений самою концепцією проєкту – створити емуляцію книжкового консультанта, подібного до того, який може працювати у фізичній книгарні. Консультант-людина не знає наперед, хто перед ним, але здатен поставити кілька уточнювальних запитань і надати доречну рекомендацію. Саме цю модель реалізовано у цифровому вигляді.

Окрему увагу приділено дизайну стартового екрана. Інтерфейс побудовано інтуїтивно зрозумілим та візуально приємним, аби не викликати у користувача відчуття технічної складності.

Навіть для тих, хто не має досвіду користування подібними інструментами, взаємодія із системою буде простою, бо потрібно лише натиснути одну з двох кнопок – залежно від того, чи потрібна допомога з вибором.

Аватар AI-консультанта, м’які кольори та доброзичливий тон запрошення створюють атмосферу уваги, спокою та поваги до користувача. Система не тисне, не нав’язує маркетингових повідомлень і не вимагає негайних дій. Це дозволяє користувачу самостійно обрати комфортний шлях взаємодії.

Головний екран виконує не лише функцію навігації, а й формує перше враження про сервіс – як про відкриту, безпечну і зручну платформу для пошуку книг.

4.1 Реалізація класичного каталогу книжок

Після обрання користувачем відповіді «ні, я знаю що хочу» на головному екрані, система перенаправляє його до традиційного каталогу книжок, приклад якого наведено на рисунку 4.2. Цей каталог реалізує просту й зрозумілу для більшості користувачів логіку: перегляд усіх доступних позицій у вигляді карток з назвою, автором, описом та кнопкою перегляду детальної інформації.

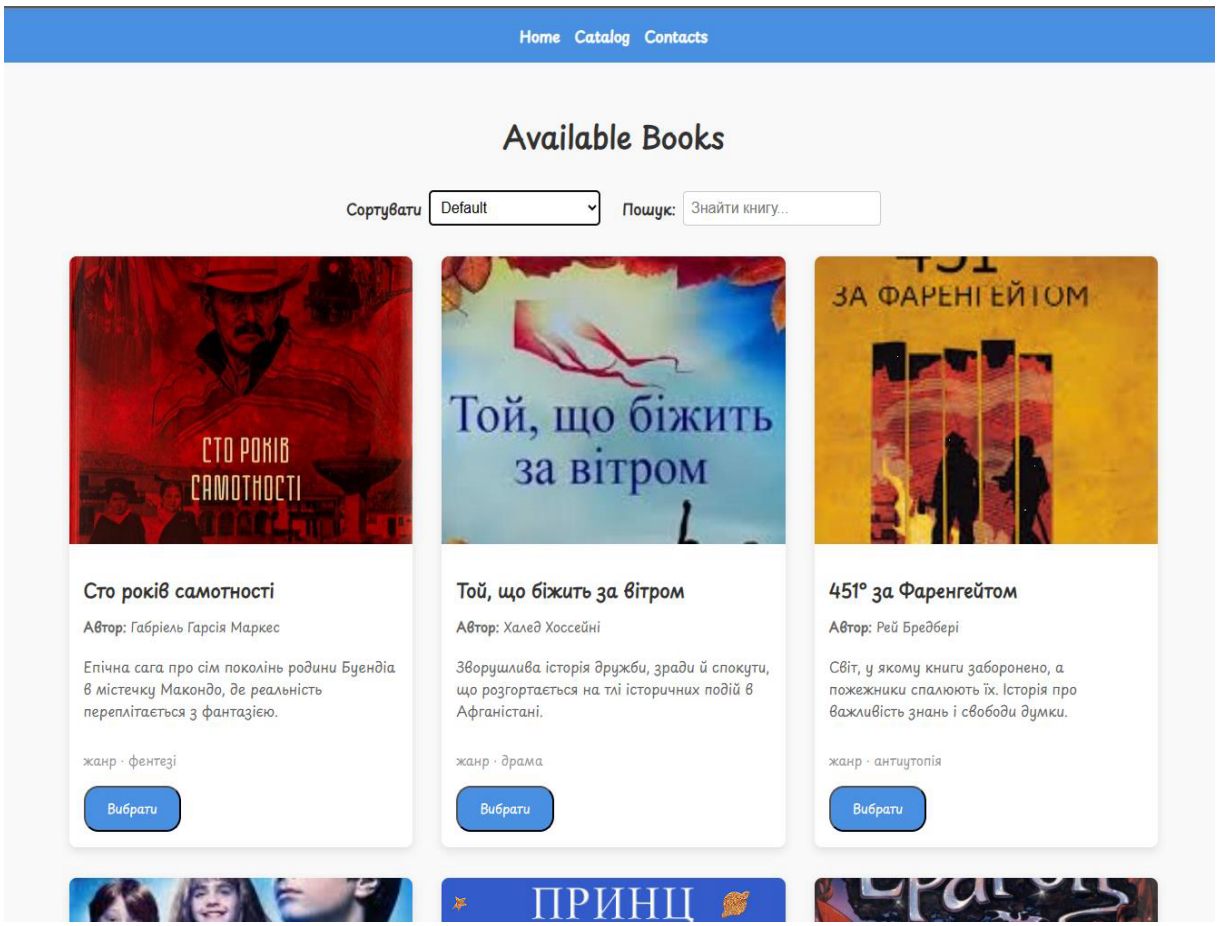

Рисунок 4.2 – Інтерфейс традиційного каталогу книжок

Незважаючи на інтеграцію сучасного AI-асистента, наявність класичного каталогу залишається критично важливою частиною системи, оскільки орієнтована на користувачів, які вже мають чітке уявлення про те, що саме шукають. У таких випадках прямий доступ до структури даних дозволяє швидко знаходити потрібні книги без проміжного діалогу з асистентом.

Класичний каталог також виконує роль резервного або паралельного інтерфейсу, забезпечуючи функціональну незалежність від мовної моделі. Це важливо як з технічної точки зору, якщо AI може бути недоступний або потребує додаткових обчислювальних ресурсів, так і з погляду користувацького вибору, оскільки деякі користувачі принципово віддають перевагу звичним, структурованим способам навігації по даних.

Крім того, класичний інтерфейс дозволяє реалізувати точну фільтрацію за метаданими. У випадках, коли параметри відомі заздалегідь, така фільтрація є більш ефективною за генеративні підходи.

Основна функціональність інтерфейсу каталогу полягає у можливості виконувати пошук за назвою, автором, жанром, алфавітне сортування тощо. Користувач може комбінувати ці параметри, що дозволяє максимально звузити коло результатів відповідно до власних вподобань.

У рамках клієнтської частини для реалізації класичного каталогу книг було використано функціонал пошуку та сортування. Компонент забезпечує швидке оновлення результатів при зміні параметрів, що дозволяє покращити взаємодію з користувачем у режимі реального часу (лістинг 4.1).

Лістинг 4.1 – Реалізація фільтрації та сортування

```
const filteredBooks = booksData
  .filter(book =>

book.title.toLowerCase().includes(searchQuery.toLowerCase())
  )
  .sort((a, b) => {
    if (sortOption === 'title') return
a.title.localeCompare(b.title);
    if (sortOption === 'author') return
a.author.localeCompare(b.author);
    return 0;
  });
```

Цей фрагмент коду забезпечує попередню обробку масиву книжок до виведення на екран, враховуючи введене ключове слово та обране значення сортування.

Для виведення результатів на сторінку використовується розмітка з двома основними елементами керування: поле пошуку та селектор сортування. Відповідний JSX-розмітка наведена нижче у лістингу 4.2.

Лістинг 4.2 – Компонент відображення відфільтрованого каталогу

```
    return (
      <div className="catalog-container">
        <h2 className="catalog-title">Available Books</h2>
        <div className="catalog-controls">
          <label>
            <strong>Сортувати </strong>
            <select   value={sortOption}   onChange={(e)   =>
setSortOption(e.target.value)}>
              <option value="Default">Default</option>
              <option value="Title">Назва</option>
              <option value="Author">Автор</option>
            </select>
          </label>
          <label>
            <strong>Пошук: </strong>
            <input
              type="text"
              placeholder="Знайти книгу..."
              value={searchQuery}
              onChange={(e) => setSearchQuery(e.target.value)}
            />
          </label>
        </div>
        <div className="book-grid">
          {filteredBooks.map((book) => (
            <div key={book.id} className="book-card">
              <img      src={book.cover}      alt={book.title}
className="book-cover" />
              <h3>{book.title}</h3>
              <p><strong>Автор:</strong> {book.author}</p>
              <p>{book.description}</p>
              <p><em>жанр · {book.genre}</em></p>
              <button>Вибрати</button>
            </div> )
```

Цей підхід дозволяє досягти інтуїтивного користувацького досвіду, при цьому не вимагаючи складної архітектури або використання зовнішніх бібліотек для реалізації фільтрації. Компонент повністю автономний і працює з локальним джерелом даних.

Повний код компонента, що реалізує функціональність класичного каталогу з можливістю фільтрації та сортування книжок, наведено в додатку А. Він включає логіку обробки запитів, елементи інтерфейсу, що забезпечують коректне відображення інформації на сторінці.

4.2 Інтеграція AI-асистента для рекомендацій книжок

Інтерфейс взаємодії з AI-асистентом побудовано у вигляді просто та інтуїтивно зрозуміло. Такий підхід дає змогу наблизити досвід користувача до живої розмови з продавцем-консультантом у книжковій крамниці.

Особливість реалізації – асистент ініціює діалог сам, запрошуючи користувача поставити запитання або звернутися за допомогою (рисунок 4.3).

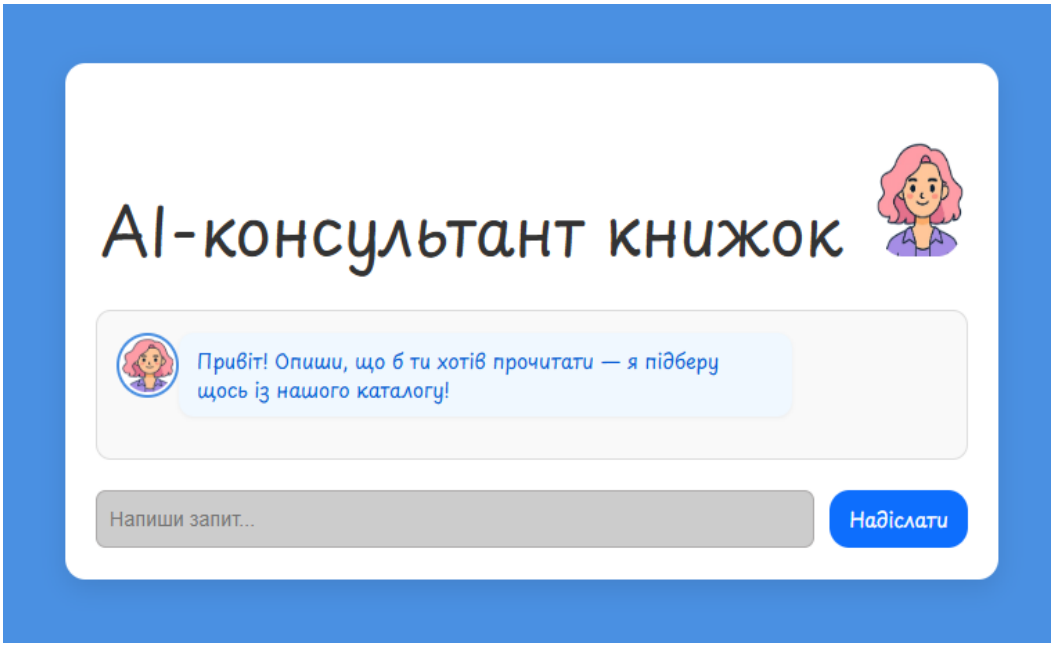

Рисунок 4.3 – Інтерфейс AI-консультанта

Для створення діалогового інтерфейсу було використано бібліотеку React Chat UI, яка забезпечує простий і гнучкий спосіб реалізації чатового вікна в React-застосунках.

Ця бібліотека дозволяє легко оформити повідомлення користувача та асистента, підтримує аватари, індикатори часу, стилізацію повідомлень і логіку оновлення стану діалогу. Завдяки цьому вдалося швидко реалізувати інтерфейс, який візуально нагадує звичні месенджери, що підвищує зручність і зрозумілість для кінцевого користувача.

Компонент створений у React із використанням хуків useState для керування станом діалогу. Після ініціації вікна, асистент автоматично надсилає стартове повідомлення. Користувач має змогу ввести запит у поле вводу й отримати відповідь від AI.

Компонент чатового вікна, який відображає повідомлення асистента та дозволяє користувачеві вводити запити наведений в лістингу 4.3.

Повідомлення зберігаються у стані messages, а після натискання кнопки «Надіслати» відбувається виклик функції handleSend, яка додає нове повідомлення до списку й надсилає його до асистента через API. Компонент використовує типову структуру для створення сучасного інтерфейсу діалогу.

Лістинг 4.3 – Початкове повідомлення та стани

```
const [messages, setMessages] = useState([
  {
    role: 'assistant',
    content: 'Привіт! Опиши, що б ти хотів прочитати — я підберу щось із нашого каталогу!'
  }
]);
const [userInput, setUserInput] = useState('');
```

У фрагменті коду, наведеному в лістингу 4.4 реалізовано функцію handleSend, яка відповідає за обробку введеного користувачем запиту та отримання відповіді від AI-асистента. Спочатку функція перевіряє, чи не є вхідне повідомлення порожнім. Далі формується новий масив newMessages, який містить попередні повідомлення чату та поточний запит користувача, після чого оновлюється стан messages.

Для надсилання даних на сервер використовується fetch-запит. У тілі запиту передається масив messages, який зберігає весь контекст діалогу. Серверна частина приймає ці дані, формує відповідний запит до Assistant API, отримує згенеровану відповідь і повертає її у форматі JSON.

Якщо у відповіді присутнє поле reply, то воно додається до масиву повідомлень як нове повідомлення з роллю assistant. Таким чином реалізується повноцінна комунікація з AI-асистентом у форматі чат-інтерфейсу.

Лістинг 4.4 – Обробка введення користувача та запит до AI

```
const handleSend = async () => {
  if (!userInput.trim()) return;
  const newMessages = [...messages, { role: 'user', content: userInput }];
  setMessages(newMessages);
  setUserInput('');
  const response = await fetch('/api/assistant', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ messages: newMessages }),
  });
  const data = await response.json();
  if (data?.reply) {
    setMessages([...newMessages, { role: 'assistant', content: data.reply }]);
```

У лістингу 4.5 реалізовано візуальну частину чат-інтерфейсу, яка надає користувачеві змогу вести діалог із рекомендаційним AI-асистентом. Компонент складається з таких трьох основних структурних частин, як заголовок, вікно чату, інтерактивне поле введення.

Лістинг 4.5 – Інтерфейс компонента (JSX)

```
return (
  <div className="assistant-container">
    <h2>AI-консультант книжок</h2>
    <div className="chat-box">
      {messages.map((msg, index) => (
        <div        key={index}        className={`message
${msg.role}`}>
          {msg.content}
        </div>
      ))}
    </div>
    <div className="input-box">
      <input
        type="text"
        placeholder="Що тебе цікавить?"
        value={userInput}
        onChange={(e) => setUserInput(e.target.value)}
      />
      <button onClick={handleSend}>Порадь</button>
    </div>
  </div>
);
```

4.3 Технічна реалізація асистента

Для забезпечення взаємодії користувача з мовною моделлю GPT-4-turbo було використано функціонал Assistant API, доступний через

платформу OpenAI. Цей інструмент дозволяє створити індивідуального асистента з фіксованими інструкціями, доступом до файлів, інструментами, та інтегрувати його як програмно, так і через інтерфейс OpenAI.

На рисунку 4.4 наведено конфігурацію асистента у вкладці Assistants. Видно, що вибрано модель gpt-4-turbo, вказано унікальний Assistant ID, який використовується для подальшої інтеграції через API, а також додано інструкції, які регламентують поведінку асистента.

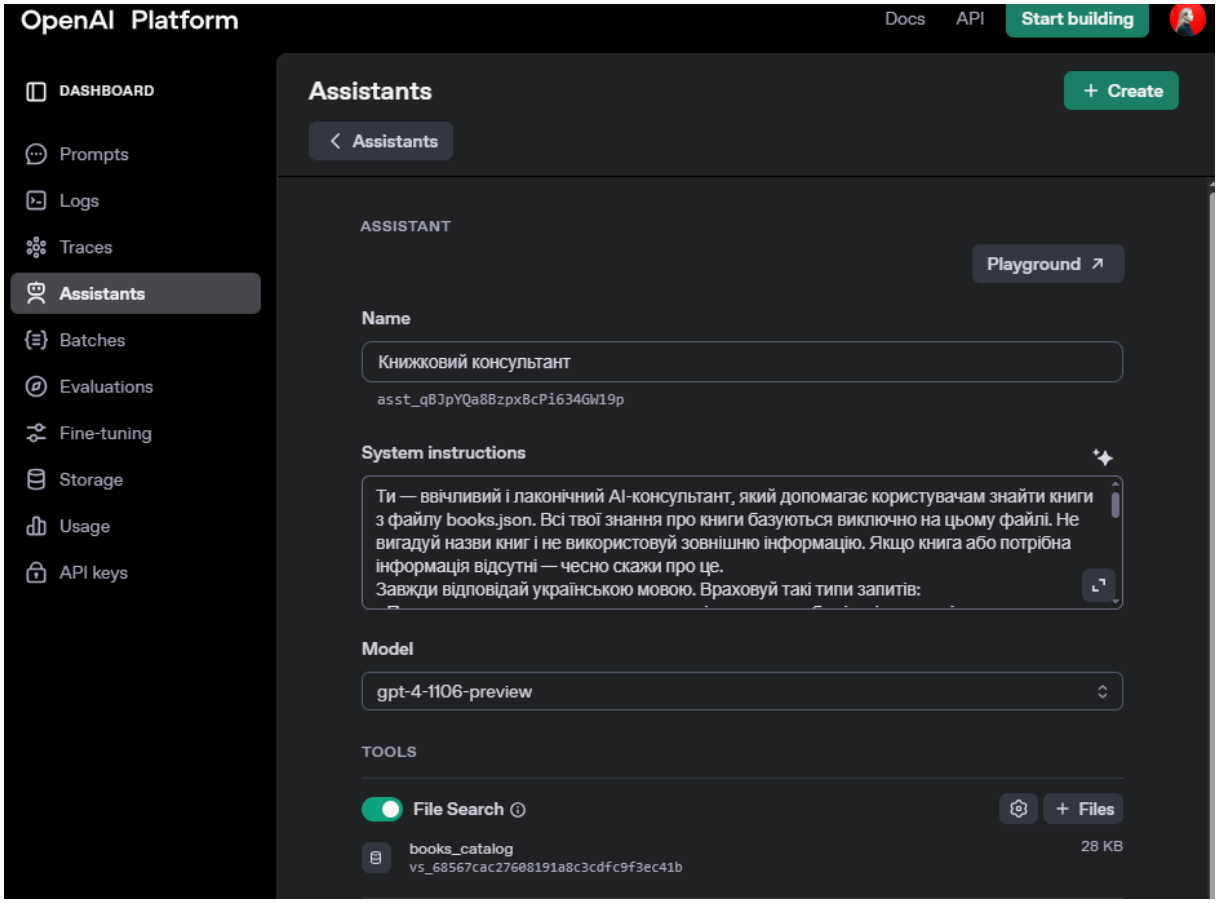

Рисунок 4.4 – Конфігурація AI-асистента у OpenAI Platform

Інструкції сформульовано таким чином, щоб асистент діяв виключно в межах наданого каталогу книжок, не вигадував нові назви, яких немає в JSON-файлі, чітко давав поради, орієнтуючись на жанрові та вікові побажання, в разі неможливості підібрати релевантну позицію – повідомляв про це користувача.

Обмеження на вигадування реалізовано за рахунок чітких інструкцій та прикріплення JSON-файлу як єдиного джерела даних. Асистент не має доступу до інтернету та не виконує довільну генерацію поза межами наданої інформації.

Файл каталогу на рисунку 4.5 прикріплений до конфігурації асистента у вигляді локального ресурсу. Його структура дозволяє зручно опрацьовувати дані: кожна книжка описана через назву, автора, жанр, вік, опис тощо. Пошук рекомендацій відбувається саме по цьому файлу.

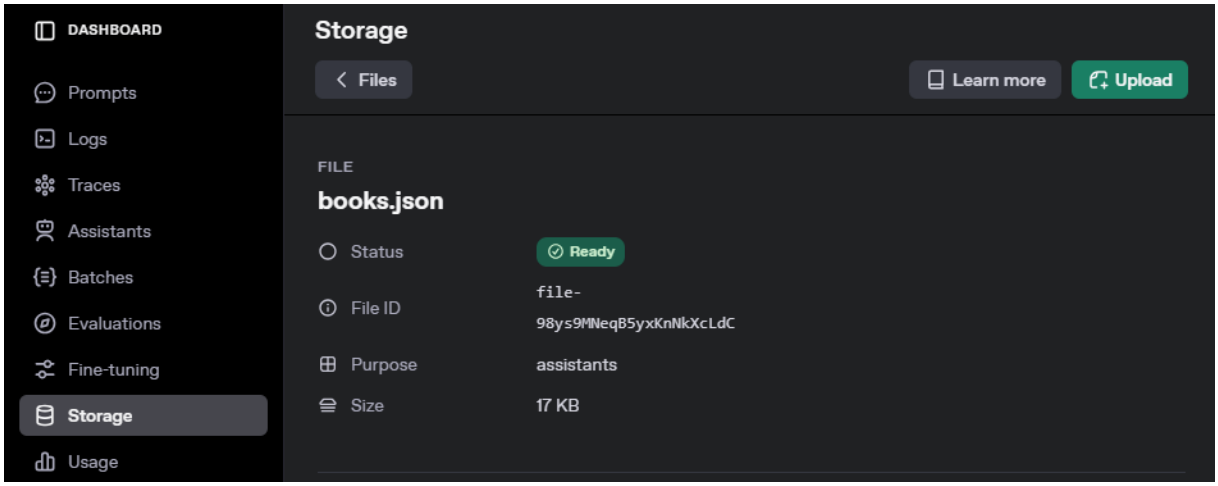

Рисунок 4.5 – Підключення файлу каталогу до AI-консультанта

У процесі конфігурації асистента важливу роль відіграє не лише сам факт прикріплення JSON-файлу, а й спосіб, у який модель отримує до нього доступ. Для цього використовується векторне сховище (vector store), яке приведено на рисунку 4.6. У верхній частині інтерфейсу вказано ID векторного сховища,яке є унікальним ідентифікатором, який дозволяє програмно звертатися до сховища.

У розділі Files attached вказано, що до сховища прикріплено файл books.json, який і містить каталог книжок.

У нижньому блоці Used by зазначено, що дане сховище використовується асистентом під назвою «Книжковий консультант» з

певним ID. Це підтверджує, що асистент працює з локальним векторизованим сховищем, що пришвидшує доступ до знань та дозволяє формувати точні рекомендації на основі запитів користувачів.

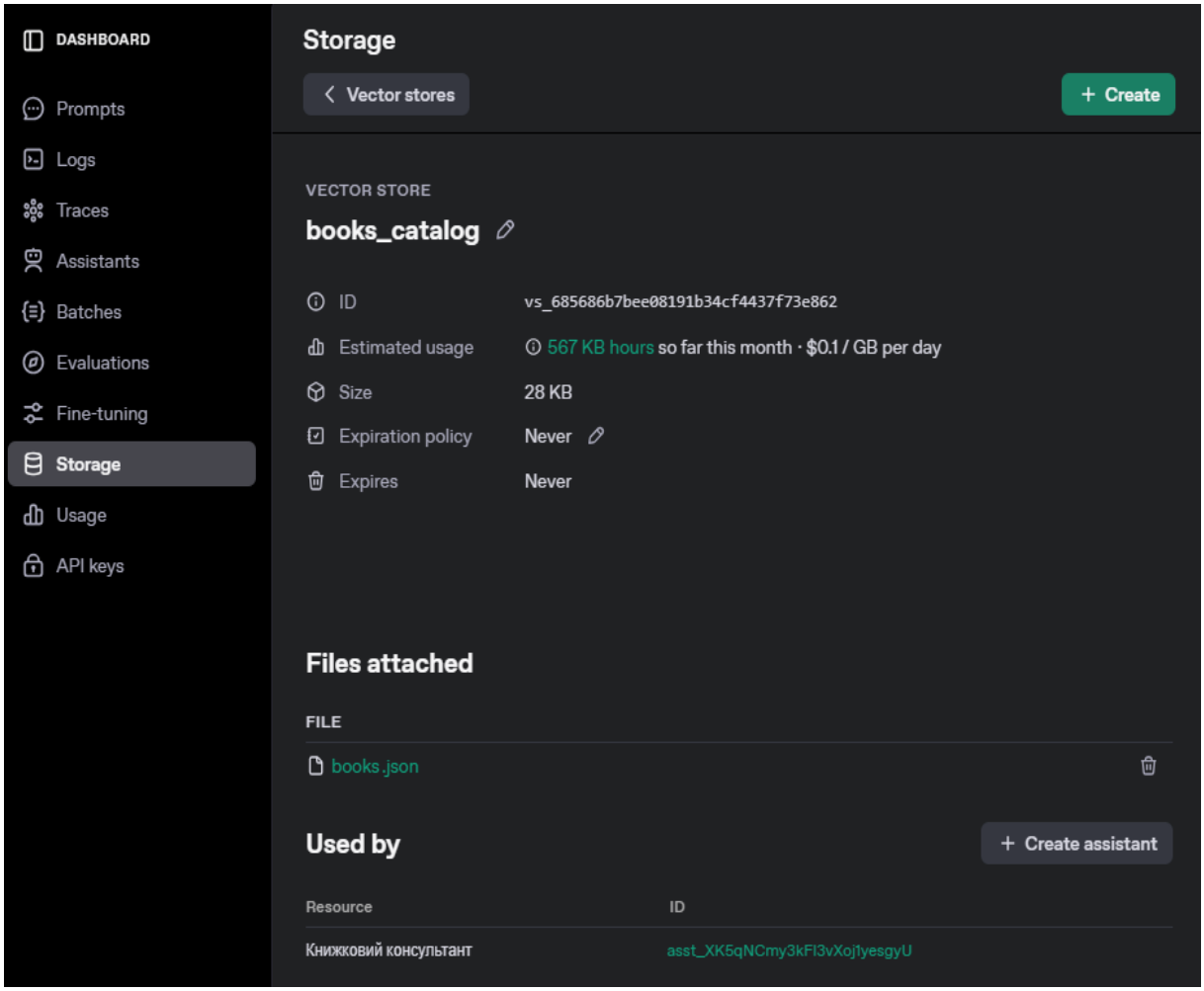

Рисунок 4.6 – Векторне сховище для прикріпленого JSON-файлу на платформі OpenAI

Сховище автоматично створюється при додаванні файлу до асистента. Його завдання полягає в тому, щоб розбити вміст документа на сегменти та трансформувати їх у векторні подання, які GPT-4-turbo зможе ефективно використовувати при формуванні відповідей.

Завдяки такому підходу, мовна модель не просто «бачить» JSON-файл як суцільний текст, а має змогу виконувати семантичний пошук по вмісту,

фокусуючись на релевантних частинах документа. Це критично важливо у сценаріях, коли файл містить десятки або сотні книжкових описів: GPT-4-turbo зможе вибирати ті, що найбільше відповідають запиту користувача, без необхідності повністю завантажувати увесь документ у контекст кожного запиту.

Такий механізм також підвищує ефективність використання токенів, дозволяє дотримуватись обмежень на вигадування та забезпечує масштабованість системи – у майбутньому до векторного сховища можна буде додавати нові файли без потреби повного перенавчання асистента.

У межах даного проєкту асистент був створений і налаштований програмно за допомогою OpenAI SDK для Node.js. В лістингу 4.6 наведено фрагмент коду з підключенням і налаштуванням асистента. Це дало змогу максимально гнучко визначити його поведінку: було обрано модель GPT-4 Turbo, сформульовано чіткі інструкції щодо стилю відповіді та обмежень на вигадування, а також прикріплено файл з каталогом книжок у форматі JSON для забезпечення точних рекомендацій. Завдяки використанню SDK, асистент став доступним як для інтеграції в інтерфейс вебзастосунку, так і для керування через OpenAI Platform.

Лістинг 4.6 – Програмне створення асистента через OpenAI SDK

```
import OpenAI from 'openai';
const openai = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY,
});
const assistant = await openai.beta.assistants.create({
name: 'Книжковий консультант',
  instructions: 'Допомагай користувачам знаходити книжки з каталогу…',
  model: 'gpt-4-turbo',
  tools: [{ type: 'file_search' }],
  file_ids: ['file-abc123xyz456'], // ID JSON-файлу з каталогом});
```

4.4 Демонстрація сценаріїв взаємодії з AI-консультантом

У процесі проєктування та тестування AI-асистента важливо враховувати не лише успішні сценарії взаємодії, але й ті, що не дають очікуваного результату. Такі ситуації є неминучими в реальних умовах використання, оскільки користувачі можуть формулювати запити неочікуваним або надто загальним чином. Виявлення та аналіз як позитивних, так і негативних сценаріїв дозволяє вдосконалити логіку обробки запитів, структуру даних та інструкції асистента, підвищуючи загальну якість взаємодії з системою.

З метою перевірки працездатності AI-консультанта було проведено моделювання типових сценаріїв звернення користувача. Усі запити оброблялися у межах наявного JSON-каталогу, прикріпленого до асистента через OpenAI Platform. На рисунку 4.7 представлено чотири успішні приклади використання, які підтверджують практичну ефективність запропонованого рішення.

На рисунку представлено чотири приклади діалогів, які демонструють ефективність рекомендаційного механізму.

У верхньому лівому куті користувач звертається із розпливчастим запитом: «порадь книгу великого обсягу, щось фентезі». У цьому випадку пошук здійснюється не за конкретною назвою твору, а за метаданими – жанром та обсягом. Асистент правильно інтерпретує побажання й пропонує відповідний варіант – роман «Дюна», що справді належить до жанру фентезі та має значний обсяг.

У верхньому правому куті користувач формулює запит у вигляді опису сюжету, не вказуючи ні назви книги, ні автора: «головним героєм є хлопчик, який живе по планетах і дружить з лисом». У цьому випадку асистент успішно розпізнає твір «Маленький принц» Антуана де Сент-Екзюпері. Важливо, що в запиті відсутні прямі метадані (назва, автор, жанр тощо) – орієнтація відбувається виключно на описову інформацію. Це

свідчить про здатність моделі працювати з неструктурованими, контекстно насиченими запитами, що виходять за межі формального фільтрування.

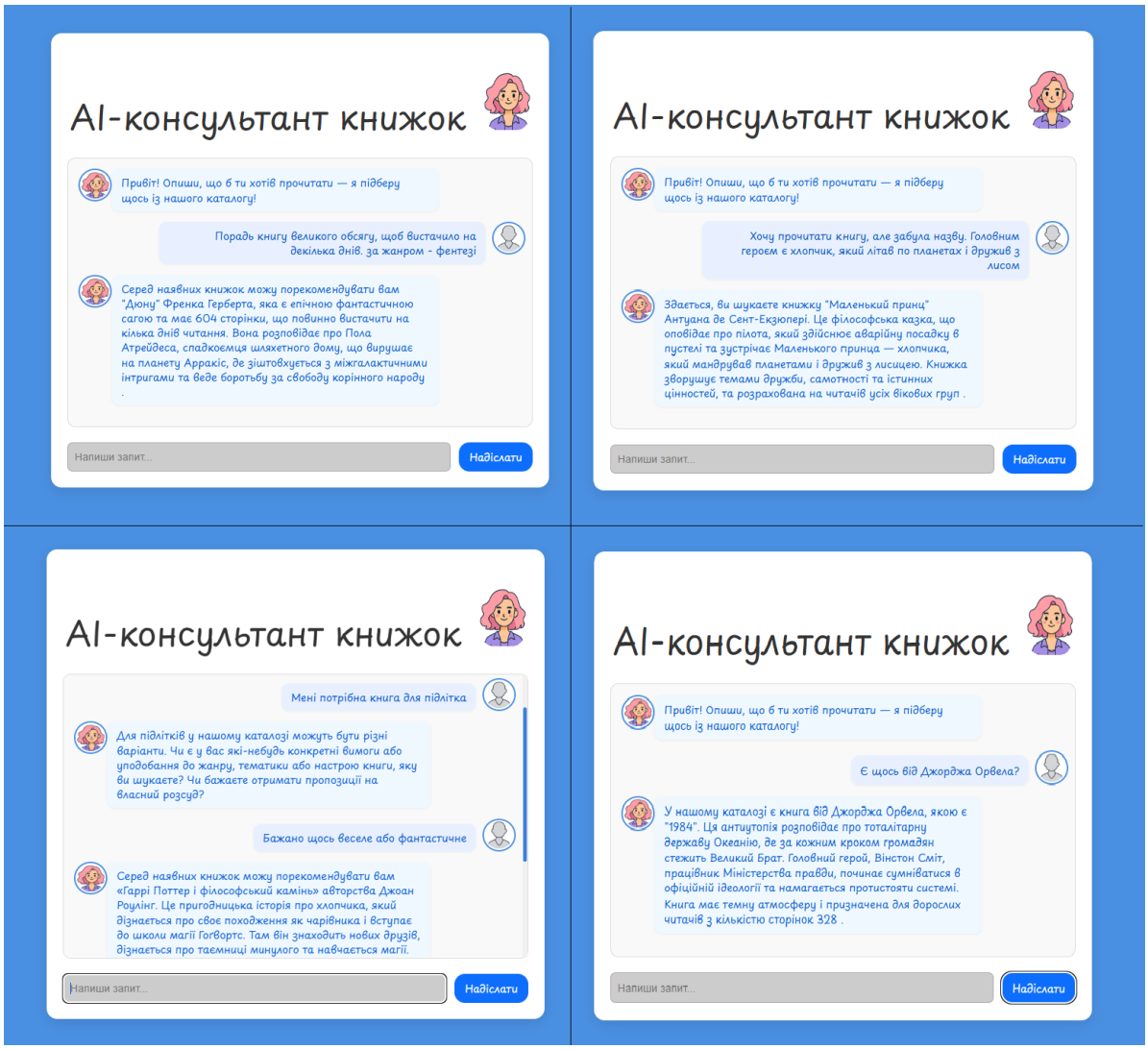

Рисунок 4.7 – Приклади позитивних сценаріїв

У нижньому лівому куті користувач звертається із загальним запитом «книга для підлітка», який не містить конкретних жанрових чи змістових орієнтирів. Асистент не надає негайної відповіді, а ініціює уточнення, запитуючи про бажану тематику чи настрій. Після уточнення з боку користувача система пропонує відповідну книжку – «Гаррі Поттер». Такий сценарій демонструє здатність асистента підтримувати контекстно-

залежний діалог, коректно інтерпретувати додаткові дані та адаптувати рекомендацію відповідно до розширених побажань.

У нижньому правому куті користувач ставить чітко сформульований запит, тобто звертається до системи із зазначенням конкретного автора. Асистент правильно інтерпретує запит, виконує фільтрацію каталогу за авторською ознакою та пропонує книгу «1984». При цьому надається не лише назва, але й стислий зміст, жанрова належність і згадка про великий обсяг твору. Така відповідь свідчить про коректну роботу механізму обробки запитів із чітко заданими параметрами, демонструє вміння асистента ефективно знаходити збіги за метаданими та надавати вичерпну інформацію про знайдену позицію (рисунок 4.8).

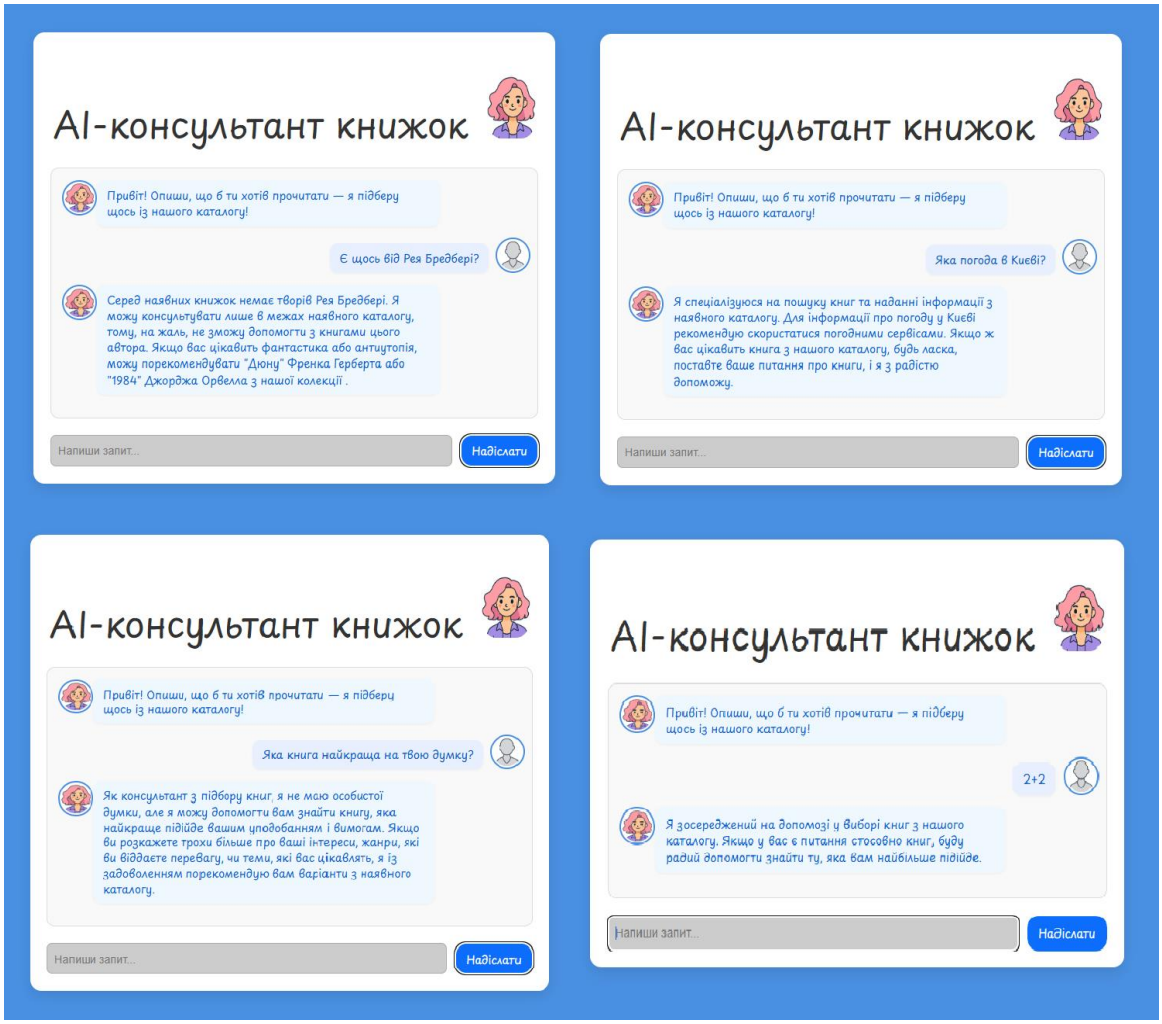

Рисунок 4.8 – Приклади негативних сценаріїв

У верхньому лівому куті наведено приклад сценарію, який ілюструє ситуацію, коли користувач формує запит на основі конкретного автора, проте відповідна інформація відсутня у локальному каталозі. AI-асистент коректно розпізнає запит і не здійснює вигадування або моделювання неіснуючої інформації. Замість цього він повідомляє про відсутність творів автора у наявній базі даних та пропонує альтернативні варіанти, тематично наближені до очікувань користувача.

У верхньому правому куті розглянуто сценарій, де прикладі користувач звертається до асистента із запитом, що виходить за межі його функціонального призначення. Оскільки AI-консультант розроблений виключно для роботи з локальним каталогом книжок, він не має доступу до зовнішніх джерел даних у реальному часі та не може надати інформацію про погоду. Система коректно ідентифікує невідповідність запиту та ввічливо пояснює сферу своєї компетентності, одночасно пропонуючи продовжити взаємодію у межах тематики книжкових рекомендацій.

У нижньому лівому куті користувач формулює запит, що передбачає наявність особистої думки або оцінки з боку асистента, яка не відповідає природі його функціонування. Оскільки AI-консультант не володіє суб’єктивними переконаннями, він не може робити емоційно чи особистісно забарвлені висновки. У відповіді система чітко пояснює межі своїх можливостей, наголошуючи на тому, що вона може лише допомогти знайти відповідну книжку, виходячи з уподобань користувача.

У нижньому правому куті користувач вводить запит, що не має жодного стосунку до тематики книжкового каталогу. Такий запит не несе інформації, яка могла б бути інтерпретована як запит на пошук літератури, і водночас не є питанням про контекст функціонування системи. Асистент коректно обробляє ситуацію, пояснюючи, що його функціонал обмежено лише рекомендацією книжок на основі наявного каталогу. Це демонструє здатність системи ідентифікувати нерелевантні запити та тактовно скеровувати користувача в межі дозволеного функціоналу.

Проведений аналіз демонструє, що реалізований AI-асистент здатний ефективно взаємодіяти з користувачами як у типових, так і в нетипових ситуаціях. У позитивних сценаріях система демонструє гнучкість обробки запитів різної специфіки – від узагальнених до описових чи орієнтованих на конкретні метадані. У випадках негативних сценаріїв асистент коректно обмежує сферу своєї компетенції, утримується від вигадування та надає ввічливі пояснення. Це свідчить про досягнення балансу між відкритістю до діалогу та дотриманням заданих меж роботи, що є ключовим критерієм якості взаємодії у персоналізованих системах рекомендацій.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було опрацьовано теоретичні основи побудови сучасних рекомендаційних систем, проаналізовано методи обробки природної мови та можливості інтеграції великих мовних моделей, зокрема GPT, у програмні продукти, орієнтовані на персоналізовану взаємодію з користувачем.

Вивчення предметної галузі дозволило виявити основні підходи до реалізації рекомендаційних систем та охарактеризувати їхні сильні й слабкі сторони. Особливу увагу було приділено сучасним тенденціям, що включають застосування методів обробки природної мови, аналізу настроїв, розпізнавання намірів та інтеграцію діалогових моделей. Було проаналізовано, як ці технології підвищують релевантність рекомендацій, розширюють можливості систем у контекстуальних сценаріях використання та сприяють побудові пояснюваних інтерфейсів.

Практичний компонент роботи полягав у постановці задачі розробки програмної системи для інтелектуального підбору книг на основі запитів користувача, сформульованих природною мовою. Було сформульовано вимоги до системи, описано структуру рішення та запропоновано архітектуру взаємодії її основних компонентів: інтерфейсу користувача, серверної логіки, локального сховища даних та зовнішнього GPT-модуля. Особливістю розробки є використання моделі GPT як семантичного посередника між вхідним запитом і структурованими даними, що дозволяє уникнути необхідності у традиційній логіці фільтрації або ручному налаштуванні.

Окрему цінність набули аналітичні навички, пов’язані з критичним оцінюванням джерел, порівнянням різних підходів до реалізації рекомендаційних систем і обґрунтуванням вибору архітектурного рішення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Roy D., Dutta M. A systematic review and research perspective on recommender systems. *Journal of Big Data*. 2022. Vol. 9. Article number: 59. URL: https://doi.org/10.1186/s40537-022-00592-5 (дата звернення: 22.05.2025).

2. Lops P., Gemmis M. de, Semeraro G. Content-based recommender systems: State of the art and trends. *Lecture Notes in Computer Science*. 2011. Vol. 473. P. 73–105. URL: https://www.researchgate.net/publication/226098747_Content-based_Recommender_Systems_State_of_the_Art_and_Trends (дата звернення: 22.05.2025).

3. Sabiri B., Khtira A., El Asri B., Rhanoui M. Hybrid quality-based recommender systems: A systematic literature review. *Journal of Imaging*. 2025. Vol. 11, No. 1. Article 12. URL: https://doi.org/10.3390/jimaging11010012 (дата звернення: 22.05.2025).

4. Uta M., Felfernig A., Le V., Tran T., Garber D., Lubos S., Burgstaller T. Knowledge-based recommender systems: Overview and research directions. *Frontiers in Big Data*. 2025. Vol. 7. Article 1573072. URL: https://www.frontiersin.org/journals/big-data/articles/10.3389/fdata.2025.1573072/full (дата звернення: 22.05.2025).

5. Zhao Z., Fan W., Li J., Liu Y., Mei X., Wang Y., Wen Z., Wang F., Zhao X., Tang J., Li Q. Recommender systems in the era of large language models (LLMs). *arXiv preprint*. 2023. arXiv:2307.02046. URL: https://arxiv.org/abs/2307.02046 (дата звернення: 27.05.2025).

6. Al-Hasan T. M., Sayed A. N., Bensaali F., Himeur Y., Varlamis I., Dimitrakopoulos G. From traditional recommender systems to GPT-based chatbots: A survey of recent developments and future directions. *Big Data and Cognitive Computing*. 2024. Vol. 8, No. 4. P. 36. URL: https://doi.org/10.3390/bdcc8040036 (дата звернення: 01.06.2025).

7. Pande C., Witschel H., Martin A. New hybrid techniques for business recommender systems. *Applied Sciences*. 2022. Vol. 12, No. 10. P. 4804. URL: https://www.mdpi.com/2076-3417/12/10/4804 (дата звернення: 01.06.2025).

8. Paranjape V., Nihalani N., Mishra N. Design and development of an efficient demographic-based movie recommender system using hybrid machine learning techniques. *International Journal of Computers Communications & Control*. 2024. Vol. 19, No. 4. Article 5840. URL: https://univagora.ro/jour/index.php/ijccc/article/view/5840 (дата звернення: 02.06.2025).

9. Çano E., Morisio M. Hybrid recommender systems: A systematic literature review. *arXiv preprint*. 2019. arXiv:1901.03888. URL: https://arxiv.org/abs/1901.03888 (дата звернення: 02.06.2025).

10. Uta M., Felfernig A., Le V., Tran T., Garber D., Lubos S., Burgstaller T. Knowledge-based recommender systems: Overview and research directions. *Frontiers in Big Data*. 2024. Vol. 7. Article 1304439. URL: https://www.frontiersin.org/journals/big_data/articles/10.3389/fdata.2024.1304439/full (дата звернення: 02.06.2025).

11. Pahlavan S., Akbarzadeh A., Ishikawa F. Language models for recommendation: A survey and vision. *SSRN Electronic Journal*. 2024. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4828316 (дата звернення: 09.06.2025).

12. Pasrija V., Pasrija S. The cold-start problem in recommender systems: Challenges and mitigation techniques. *International Research Journal of Modernization in Engineering Technology and Science*. 2024. Vol. 6, No. 5. P. 55701.
URL: https://www.irjmets.com/uploadedfiles/paper//issue_5_may_2024/55701/final/fin_irjmets1715656884.pdf (дата звернення: 09.06.2025).

13. Crespo R. G., Martínez O. S., Lovelle J. M. C., García-Bustelo B. C. P., Gayo J. E. L., Pablos P. O. Recommendation system based on user interaction

data applied to intelligent electronic books. *Computers in Human Behavior*. 2011. Vol. 27, No. 4. P. 1445–1449.URL: https://labra.weso.es/pdf/2011_RecommendationSystemElectronicBooks.pdf (дата звернення: 09.06.2025).

14. Creating a React App. A guide to creating your first React application. URL: https://react.dev/learn/creating-a-react-app (дата звернення: 22.06.2025).

15. React Chat UI. A library for building chat interfaces in React. URL: https://github.com/brandonmowat/react-chat-ui (дата звернення: 14.06.2025).

16. Assistants API Tools. Documentation on using tools with OpenAI Assistants API. URL: https://platform.openai.com/docs/assistants/tools (дата звернення: 22.06.2025).

17. Vite Features. Overview of core features provided by the Vite build tool. URL: https://vite.dev/guide/features.html (дата звернення: 14.06.2025).

18. Node.js API Synopsis. Introduction to the Node.js API and its basic usage. URL: https://nodejs.org/docs/latest/api/synopsis.html (дата звернення: 15.06.2025).

19. OpenAI API Realtime Sessions. Documentation on managing realtime sessions with the OpenAI API. URL: https://platform.openai.com/docs/api-reference/realtime-sessions (дата звернення: 15.06.2025).

20. Express 4.x API Reference. Comprehensive reference for the Express.js web framework. URL: https://expressjs.com/en/4x/api.html (дата звернення: 15.06.2025).