МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ЧАТ-БОТ ДЛЯ РЕКОМЕНДАЦІЙ ПІЗНАВАЛЬНОГО МАТЕРІАЛУ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Іванна СУПРУН

«___»_____ 2026 р.

Керівник канд. техн. наук, доцент _____Євген СІДЕНКО

«___»_____ 2026 р.

Миколаїв – 2026

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Супрун Іванни Володимирівни
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Чат-бот для рекомендацій пізнавального матеріалу

Керівник роботи:
Сіденко Євген Вікторович, канд. техн. наук, доцент, завідувач кафедри інтелектуальних інформаційних систем.
(прізвище, ім’я, по батькові, посада, науковий ступінь, вчене звання)

Затверджена наказом ЧНУ ім. Петра Могили від «_25_» _12_ 2025 р. № __353__.

2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: розробка та програмна реалізація інтелектуального чат-бота для рекомендації пізнавального контету (книг, фільмів та музики) українською мовою. Система повинна забезпечувати аналіз запитів користувача, формування

персоналізованих рекомендацій,накопичення історії взаємодії та адаптацію рекомендацій на основі заоротного зв'язку.

Початкові дані: методи обробки природної мови, рекомендаційеі алгоритми API зовнішніх сервісів (TMDB, Last.fm, Google Books, Open Library), засоби розробки Python, бібліотеки PyTorch, SentenceTransformer, aiogram, SQLAlchemy.

4. Перелік питань, що підлягають розробці: аналіз предметної сфери рекомендації пізнавального контенту; дослідження існуючих аналогів чат-ботів рекомендаційного типу; формування вимог до інформаційної системи; дослідження методів обробки природної мови; дослідження алгоритмів персоналізованих рекомендацій; проєктування архітектури чат-бота; розробка бази даних та механізмів зберігання інформації; реалізація чат-бота засобами Python;інтеграція зовнішніх інформаційних сервісів; тестування функціональних можливостей системи; аналіз якості отриманих рекомендацій; розробка керівництва користувача.

5. Перелік графічних матеріалів: схема процесу рекомендації пізнавального контенту;порівняльні таблиці аналогів; схема постановки задачі; пайплайн обробки природної мови; архітектура програмної системи; UML-діаграми системи; схема бази даних; приклади роботи чат-бота; результати тестування та оцінювання якості рекомендацій.

Керівник роботи _____ __Євген СІДЕНКО____
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ ____Іванна СУПРУН__
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «_25_» Грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Чат-бот для рекомендацій пізнавального матеріалу.

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	21.12.2025	24.12.2025	
2	Аналіз предметної області та постановка задачі	25.12.2025	30.01.2026	
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема аналіз існуючих чат-ботів та рекомендаційних систем для підбору пізнавального контенту	31.01.2026	01.03.2026	
4	Огляд існуючих методів обробки природної мови, рекомендаційних алгоритмів та архітектур штучних нейронних мереж для реалізації чат-бота рекомендацій	02.03.2026	01.04.2026	
5	Розробка та реалізація чат-бота для рекомендацій пізнавального матеріалу з аналізом результатів тестування	02.04.2026	24.05.2026	
6	Перший попередній захист КР на засіданні комісії кафедри	25.05.2026	25.05.2026	
7	Корегування роботи за результатами попереднього захисту	26.05.2026	04.06.2026	
8	Другий попередній захист КР на засіданні комісії кафедри	05.06.2026	05.06.2026	
9	Доробка та остаточне оформлення КР	06.06.2026	14.06.2026	
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.06.2026	19.06.2026	

Керівник роботи _____ _____
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ _____
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«____» _____ 2026 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача (ки) групи 401 ЧНУ ім. Петра Могили

Супрун Іванни Володимирівни

на тему: **“ЧАТ-БОТ ДЛЯ РЕКОМЕНДАЦІЙ ПІЗНАВАЛЬНОГО МАТЕРІАЛУ”**

Актуальність даної роботи полягає у необхідності підвищення ефективності доступу до пізнавального контенту в умовах інформаційного перевантаження сучасного суспільства та розвитку інтелектуальних систем, що забезпечують персоналізовані рекомендації для підтримки самоосвіти користувачів. Розробка подібного програмного забезпечення із застосуванням сучасних методів обробки природної мови та рекомендаційних алгоритмів дозволяє підвищити релевантність отриманого контенту та мотивацію до навчання.

Метою роботи є розробка чат-бота Virgil, як ефективного інструменту самоосвіти, для формування рекомендацій пізнавального матеріалу.

Об’єктом роботи є процеси створення рекомендацій.

Предметом роботи є технології розробки чат-бота з функцією рекомендацій пізнавального матеріалу.

В результаті виконання роботи проаналізовано предметну область та існуючі аналоги рекомендаційних систем, сформульовано технічні вимоги до системи, досліджено методи обробки природної мови та алгоритми рекомендацій. Розроблено модульну архітектуру програмної системи, реалізовано гібридний підхід до обробки запитів, що поєднує класифікацію інтентів за допомогою LSTM-мережі, семантичне ранжування на основі SentenceTransformer, оркестрацію запитів через велику мовну модель Groq та персоналізацію рекомендацій із використанням SessionGRU з урахуванням зворотного зв’язку користувача. Система інтегрує зовнішні джерела даних (TMDB, Last.fm, Google Books, Open

Library), застосовує кешування та механізми дедуплікації для підвищення швидкодії та точності рекомендацій.

Дана кваліфікаційна робота складається з чотирьох розділів. У першому розділі проведено аналіз предметної області, огляд сучасних рекомендаційних систем та чат-ботів для підбору пізнавального контенту, а також сформовано постановку задачі. Другий розділ присвячений дослідженню методів обробки природної мови, рекомендаційних алгоритмів та архітектур нейронних мереж, що використані для реалізації системи. У третьому розділі наведено результати розробки та проєктування інтелектуального чат-бота для персоналізованих рекомендацій фільмів, книг і музики. У четвертому розділі здійснено тестування системи, аналіз отриманих результатів та оцінку ефективності роботи рекомендаційного модуля.

Загальний обсяг роботи – 74 сторінок. Кваліфікаційна робота містить 1 додаток, 12 рисунків, 19 таблицю і 35 джерел посилання.

Ключові слова: чат-бот, рекомендаційна система, обробка природної мови, персоналізовані рекомендації, пізнавальний контент, штучний інтелект, нейронні мережі, самоосвіта, Telegram-бот.

ABSTRACT

to the qualification work by the student of the group 401 of Petro Mohyla Black Sea National University

Suprun Ivanna

“CHATBOT FOR RECOMMENDING COGNITIVE MATERIAL”

The relevance of this work arises from the need to increase the efficiency of access to cognitive content in the conditions of information overload of modern society and the development of intelligent systems that provide personalized recommendations to support users' self-education. The development of such software using modern methods of natural language processing and recommendation algorithms allows to increase the relevance of the received content and motivation for learning.

The purpose of the work is to develop a chatbot Virgil, as an effective self-education tool for generating recommendations for cognitive material.

The object of the work is the processes of creating recommendations.

The subject of the work is the technologies for developing a chatbot with the function of recommendations for cognitive material.

As a result of the work, the subject area and existing analogues of recommendation systems were analyzed, technical requirements for the system were formulated, natural language processing methods and recommendation algorithms were investigated. A modular architecture of the software system was developed, a hybrid approach to query processing was implemented, which combines intent classification using an LSTM network, semantic ranking based on SentenceTransformer, query orchestration through the large Groq language model, and recommendation personalization using SessionGRU taking into account user feedback. The system integrates external data sources (TMDB, Last.fm, Google Books, Open Library), uses caching and deduplication mechanisms to increase the speed and accuracy of recommendations.

This qualification work consists of four sections. The first section analyzes the subject area, reviews modern recommendation systems and chatbots for selecting cognitive content, and formulates the problem statement. The second section is devoted to the study of natural language processing methods, recommendation algorithms and neural network architectures used to implement the system. The third section presents the results of the development and design of an intelligent chatbot for personalized recommendations of movies, books and music. The fourth section tests the system, analyzes the results obtained and evaluates the efficiency of the recommendation module. The total volume of the work is 74 pages. The qualification work contains 1 appendix, 12 figures, 19 tables and 35 references.

Keywords: chatbot, recommendation system, natural language processing, personalized recommendations, cognitive content, artificial intelligence, neural networks, self-education, Telegram bot.

ЗМІСТ

ВСТУП

У сучасному інформаційному суспільстві, де обсяги доступного контенту зростають експоненційно, людина стикається з гострою проблемою ефективного відбору якісного пізнавального матеріалу. Традиційні пошукові системи та рекомендаційні платформи часто не враховують індивідуальні особливості користувача – рівень знань, попередні інтереси, емоційний стан чи контекст запиту, – що призводить до інформаційного перевантаження, поверхневого сприйняття та втрати мотивації до самоосвіти. Фільми, книги та музика, які несуть значний когнітивний потенціал, здатні розвивати критичне мислення, розширювати світогляд і формувати емпатію, проте їхній пошук залишається складним і малоефективним для пересічного користувача.

Теоретичні здобутки в галузі рекомендаційних систем спираються на колаборативну фільтрацію, контент-базовані методи та гібридні архітектури, збагачені досягненнями обробки природної мови й векторних представлень. Практичні реалізації освітніх чат-ботів, зокрема міжнародні проєкти на кшталт Khanmigo чи вітчизняні ініціативи Міністерства освіти і науки України, доводять ефективність персоналізованого супроводу навчання. Водночас у цій сфері зберігаються суттєві прогалини: обмежена адаптація до україномовного середовища, недостатня глибина персоналізації на основі динамічної історії взаємодій, брак інтеграції семантичного аналізу з рекурентними моделями та слабка орієнтація саме на пізнавальний (а не лише суто навчальний) контент. Існуючі рішення часто залишаються або надто загальними, або надто вузькоспеціалізованими, що не дозволяє їм стати справжнім супутником людини в повсякденному когнітивному розвитку.

Актуальність даної кваліфікаційної роботи зумовлена необхідністю створення інтелектуального інструменту, який би ефективно долав інформаційне перевантаження та пропонував персоналізовані рекомендації пізнавального матеріалу у зручному форматі Telegram-бота. Виконання роботи бакалавра за

спеціальністю 122 «Комп’ютерні науки» надає можливість поєднати теоретичні знання з практичною реалізацією чат-бота Virgil – системи, що розуміє запити українською мовою, аналізує контекст і постійно вдосконалюється завдяки механізмам зворотного зв’язку.

Робота органічно продовжує і розвиває ідеї попередніх досліджень у сфері освітніх чат-ботів, інтегруючи передові технології обробки природної мови, оркестрацію великих мовних моделей та власні рекурентні мережі для глибокої персоналізації. Таким чином, розроблений чат-бот стає не лише технічним рішенням, а й дієвим засобом підтримки самоосвіти, роблячи процес пізнання доступним, мотиваційним і по-справжньому людяним для широкої аудиторії.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ ТА ПОСТАНОВКА ЗАДАЧІ ДЛЯ ЧАТ-БОТА РЕКОМЕНДАЦІЙ ПІЗНАВАЛЬНОГО МАТЕРІАЛУ

1.1 Опис предметної сфери рекомендації пізнавального контенту за допомогою чат-ботів

У сучасному інформаційному суспільстві, де обсяги доступного контенту зростають експоненційно, а час на його опрацювання залишається обмеженим, особливого значення набуває проблема ефективного доступу до пізнавального матеріалу.

Пізнавальний контент охоплює широкий спектр ресурсів, спрямованих на розвиток інтелекту, розширення кругозору, набуття нових знань та формування критичного мислення: від науково-популярних книг і документальних фільмів до освітніх подкастів, статей і навіть культурних творів, які сприяють глибшому розумінню світу. Цей тип контенту не лише інформує, але й стимулює когнітивну активність користувача, допомагаючи йому осмислювати складні явища, розвивати емпатію через історії та набувати практичних навичок саморозвитку.

Традиційні пошукові системи чи рекомендаційні платформи часто не враховують індивідуальні особливості людини – її рівень знань, попередні інтереси, емоційний стан чи контекст запиту, що призводить до поверхневого сприйняття інформації та втрати мотивації до подальшого навчання [1].

Саме тут на сцену виходять чат-боти як інноваційний інструмент персоналізованої рекомендації пізнавального контенту. Вони перетворюють пасивний пошук на динамічну, природну розмову, де користувач може формулювати запити у звичній мовній формі, а система – інтерпретувати їх з урахуванням нюансів, таких як уточнення теми, настрою чи жанру.

Завдяки інтеграції технологій штучного інтелекту чат-боти здатні аналізувати не лише явні запити, але й неявні сигнали, формуючи рекомендації, які максимально відповідають когнітивним потребам людини. Дослідження свідчать, що такі системи значно підвищують залученість користувачів до освітнього

процесу, оскільки роблять його більш інтерактивним і адаптивним до індивідуальних траєкторій розвитку [2].

Розвиток предметної сфери рекомендаційних систем для пізнавального контенту тісно пов’язаний з еволюцією штучного інтелекту. Ранні підходи базувалися на колаборативній фільтрації, де рекомендації формувалися на основі схожості вподобань різних користувачів, або на контент-базованих методах, що аналізували подібність самих матеріалів. Однак ці методи часто страждали від проблеми холодного старту для нових користувачів та обмеженої здатності до розуміння контексту.

Сучасні чат-боти долають ці недоліки завдяки гібридним архітектурам, які поєднують семантичний аналіз тексту, векторні представлення даних та моделі персоналізації на основі історії взаємодій. Вони дозволяють не просто пропонувати контент, а будувати цілісний діалог, у якому рекомендації еволюціонують разом із користувачем – від загальних оглядів до глибоких, спеціалізованих матеріалів.

Особливо цінним є використання чат-ботів у сфері пізнавального контенту через їхню здатність до багатомовної підтримки та культурної адаптації. Для україномовних користувачів це означає можливість отримувати рекомендації, що враховують національний контекст, історичну спадщину чи актуальні суспільні теми, роблячи процес навчання ближчим і доступнішим.

Крім того, чат-боти сприяють подоланню інформаційного перевантаження, фільтруючи шум і акцентуючи увагу на якісних, перевірених джерелах, що стимулює критичне мислення та довготривале запам’ятовування [3].

Рисунок 1.1 демонструє схему основних етапів процесу рекомендації пізнавального контенту в чат-боті.

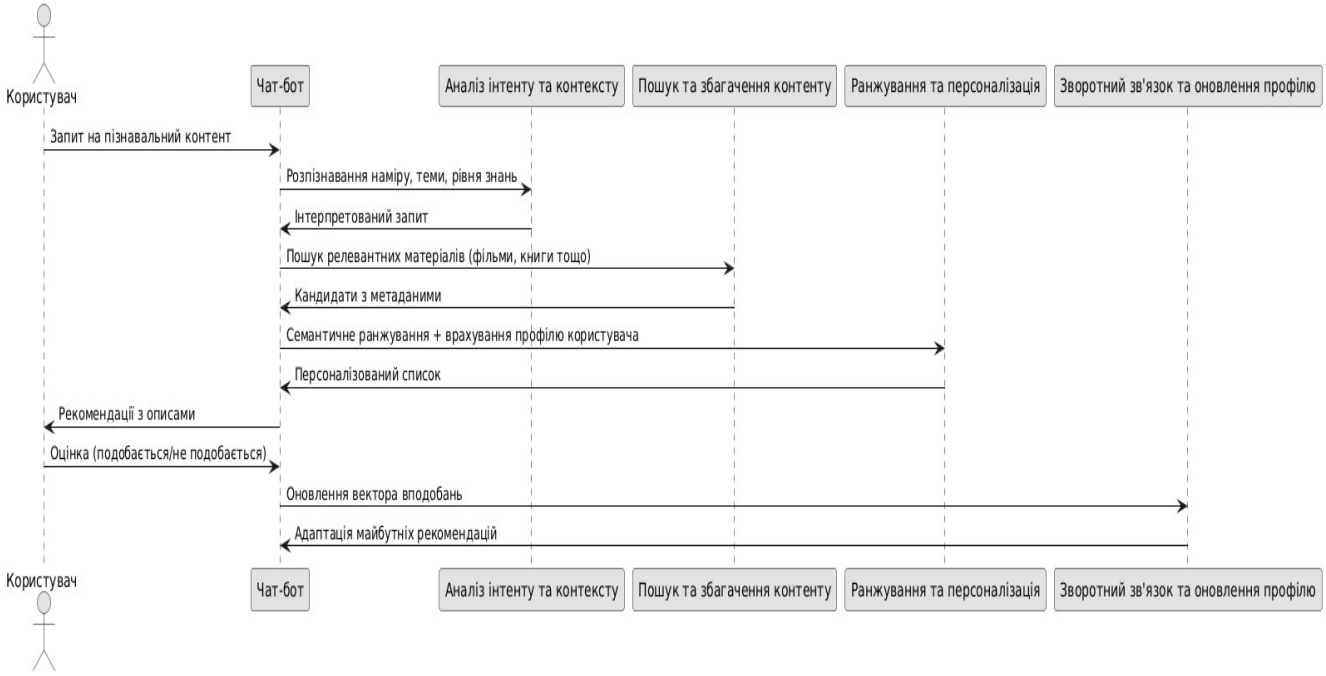

Рисунок 1.1 – Схема основних етапів процесу рекомендації пізнавального контенту в чат-боті

Як видно зі схеми на рисунку 1.1, процес рекомендації в чат-ботах для пізнавального контенту є циклічним і адаптивним. Він починається з природного спілкування, переходить до глибокого аналізу запиту, пошуку якісних джерел і завершується персоналізованим виводом, після чого користувач може надати зворотний зв’язок для постійного вдосконалення системи. Такий підхід забезпечує не лише релевантність, але й мотивацію до продовження пізнавальної діяльності.

У порівнянні з традиційними рекомендаційними платформами чат-боти демонструють низку суттєвих переваг, що робить їх особливо ефективними для роботи з пізнавальним контентом. Вони пропонують вищий рівень інтерактивності, дозволяючи користувачеві уточнювати запити в реальному часі, і забезпечують кращу персоналізацію завдяки моделям, які враховують не тільки статичні профілі, але й динамічну еволюцію інтересів. Крім того, інтеграція кешування та оптимізованих алгоритмів пошуку робить процес швидким і ресурсоефективним, що критично важливо для мобільних користувачів.

Порівняння традиційних рекомендаційних систем та чат-ботів у рекомендації пізнавального контенту представлене в таблиці 1.1.

Таблиця 1.1 – Порівняння традиційних рекомендаційних систем та чат-ботів у рекомендації пізнавального контенту

Аспект порівняння	**Традиційні рекомендаційні системи**	**Чат-боти для пізнавального контенту**
Взаємодія з користувачем	Пасивна, на основі кліків	Діалогова, природна мова
Персоналізація	На основі групової схожості	На основі індивідуальної історії та зворотного зв’язку
Адаптивність до контексту	Обмежена	Висока, з урахуванням поточного запиту
Доступність і зручність	Веб-інтерфейси	Месенджери, 24/7 доступ
Зворотний зв’язок	Рідкий	Миттєвий, інтегрований у діалог

Як ілюструє Таблиця 1.1, чат-боти значно перевершують класичні системи за ключовими параметрами, що безпосередньо впливає на якість когнітивного досвіду користувача.

Однак предметна сфера не позбавлена викликів. Серед них – забезпечення точності рекомендацій, уникнення упереджень у алгоритмах, захист персональних даних та необхідність постійного оновлення моделей для адаптації до нових тенденцій у пізнавальному контенті. Крім того, важливою залишається проблема культурної та мовної адаптації, особливо в україномовному середовищі, де чат-боти повинні не лише розуміти запити, але й пропонувати контент, що резонує з національними цінностями та актуальними суспільними питаннями [4].

Дослідники підкреслюють, що успішна реалізація таких систем вимагає комплексного підходу, який поєднує лінгвістичні моделі, векторні бази даних та механізми персоналізації на основі рекурентних нейронних мереж, що дозволяють ефективно кодувати послідовності взаємодій користувача [5].

У контексті глобальних тенденцій цифрової трансформації освіти та самоосвіти чат-боти стають не просто інструментом рекомендацій, а повноцінними інтелектуальними асистентами, що супроводжують людину на шляху пізнання. Вони сприяють формуванню навичок самостійного навчання, підвищують мотивацію та роблять процес отримання знань більш доступним для широкої аудиторії, незалежно від віку, місця проживання чи рівня попередньої підготовки [6]. Зокрема, інтеграція великих мовних моделей дозволяє чат-ботам не лише знаходити релевантний контент, але й генерувати пояснення, резюме чи навіть персоналізовані навчальні траєкторії, перетворюючи рекомендацію на цілісний освітній досвід [7].

Таким чином, предметна сфера рекомендації пізнавального контенту за допомогою чат-ботів представляє собою динамічний і перспективний напрямок розвитку інформаційних технологій, що поєднує досягнення штучного інтелекту з фундаментальними принципами педагогіки та когнітивної психології. Вона відкриває нові можливості для персоналізованого доступу до знань, роблячи процес навчання більш ефективним, приємним і адаптованим до потреб сучасної людини в умовах інформаційного суспільства.

1.2 Огляд та аналіз наявних аналогів чат-ботів для рекомендацій навчальних або пізнавальних матеріалів

У сучасній освітній сфері, де обсяги інформації зростають експоненційно, чат-боти стали потужним інструментом для рекомендацій пізнавального матеріалу, дозволяючи персоналізувати процес навчання та робити його більш доступним для широкої аудиторії [8]. Ці системи поєднують елементи штучного інтелекту, аналізу даних користувача та інтерактивного спілкування, що дає змогу пропонувати релевантний контент – від навчальних модулів і відеолекцій до інтерактивних вправ та додаткових ресурсів – відповідно до рівня знань, інтересів і темпу освоєння матеріалу.

Аналіз наявних аналогів свідчить про значний прогрес у цій галузі, адже чат-боти не лише відповідають на запитання, а й активно формують індивідуальні траєкторії пізнання, знижуючи бар’єри для самостійного навчання та підтримуючи викладачів у рутинних завданнях.

Дослідження систематичних оглядів підкреслює, що більшість таких інструментів орієнтовані саме на педагогічні цілі, забезпечуючи миттєвий зворотний зв’язок і адаптацію рекомендацій у реальному часі.

Розвиток технологій штучного інтелекту сприяв появі різноманітних рішень, які можна класифікувати за рівнем складності та сферою застосування. Деякі аналоги зосереджуються на рекомендаціях у межах конкретних предметів, наприклад, підготовці до іспитів чи опануванні мов, тоді як інші інтегруються в ширші освітні платформи, пропонуючи персоналізовані навчальні шляхи на основі аналізу попередньої активності користувача [9].

В Україні значного поширення набули чат-боти, спрямовані на підтримку дистанційного навчання, особливо в умовах викликів, пов’язаних з доступністю освіти. Так, державні ініціативи запровадили інструменти, що допомагають учням і батькам швидко знаходити інформацію про онлайн-курси, відновлення документів чи пошук закладів освіти, одночасно рекомендуючи пізнавальний контент, адаптований до поточного рівня. Ці системи демонструють високу практичність, оскільки працюють через популярні месенджери, не вимагаючи додаткового обладнання, і забезпечують цілодобовий доступ, що особливо важливо для користувачів у віддалених регіонах [10].

Міжнародний досвід також багатий на приклади ефективного використання чат-ботів як рекомендаційних систем. Університетські проєкти в Сполучених Штатах, такі як чат-боти для допомоги в навігації навчальними програмами чи виборі курсів, показують, як інтеграція з системами управління навчанням дозволяє аналізувати академічну історію студента й пропонувати персоналізовані ресурси – від додаткових матеріалів з предмета до рекомендацій щодо кар’єрного розвитку. Подібні інструменти часто поєднують рекомендації з елементами

gamification, що підвищує мотивацію користувачів і сприяє глибшому засвоєнню знань.

Наукові огляди підтверджують, що чат-боти в ролі педагогічних агентів не лише рекомендують контент, а й проводять короткі квізи, оцінюють прогрес і коригують траєкторію навчання в динамічному режимі, що робить процес більш інтерактивним і ефективним порівняно з традиційними пошуковими системами [11].

Особливу увагу заслуговує аналіз переваг і обмежень цих аналогів. З одного боку, чат-боти забезпечують високу персоналізацію, миттєву реакцію на запити та можливість роботи з великими базами даних, що дозволяє рекомендувати матеріали, які точно відповідають індивідуальним потребам – наприклад, адаптуючи складність текстів чи підбираючи візуальні посібники для візуалів. Вони також знижують навантаження на викладачів, автоматизуючи рутинні консультації та пропонуючи готові набори ресурсів для групового чи індивідуального використання.

З іншого боку, багато систем стикаються з проблемами точності рекомендацій через обмеженість даних або труднощі в розумінні контексту складних запитів, а також з питаннями конфіденційності інформації про користувачів. Крім того, не всі аналоги достатньо адаптовані до різних вікових груп чи рівнів підготовки, що іноді призводить до поверхневого підходу замість глибокого пізнання.

Систематичні дослідження підкреслюють необхідність подальшого вдосконалення алгоритмів для уникнення упереджень і підвищення якості взаємодії [12].

Для кращого розуміння відмінностей між аналогами доцільно звернутися до порівняльного аналізу їхніх ключових характеристик. Таблиця 1.2 ілюструє основні параметри функціонування кількох представників чат-ботів для рекомендацій пізнавального матеріалу, зокрема їхню орієнтацію, рівень персоналізації та інтеграційні можливості [13].

Таблиця 1.2 – Порівняльна характеристика аналогів чат-ботів для рекомендацій пізнавального матеріалу

Назва чат-бота	**Основна спрямованість**	**Рівень персоналізації**	**Інтеграція з платформами**	**Основні переваги**	**Обмеження**
Освітній чат-бот (МОН України)	Допомога в пошуку освітніх ресурсів	Середній	Месенджери	Доступність, оперативність	Обмежений аналіз прогресу
Khanmigo (Khan Academy)	Персоналізоване самонавчання	Високий	Власна платформа	Інтерактивність, адаптивні вправи	Залежність від інтернету
Університетські чат-боти (Pounce, Sunny)	Навігація курсами та рекомендації	Високий	LMS-системи	Інтеграція з академічними даними	Сфокусованість на вищій освіті
Загальні педагогічні агенти	Рекомендації + оцінювання	Середній-Високий	Різноманітні	Гнучкість у застосуванні	Проблеми з точністю в складних темах

Типова архітектура таких чат-ботів передбачає кілька взаємопов’язаних компонентів, що забезпечують ефективну роботу з рекомендаціями. На рисунку 1.2 наведено схематичне представлення основних етапів функціонування.

Рисунок 1.2 – Типова архітектура чат-бота для рекомендацій пізнавального

матеріалу

Рисунок 1.2 демонструє циклічний процес, де аналіз запиту плавно переходить у відбір контенту, генерацію пропозицій і постійне вдосконалення на основі зворотного зв’язку, що є ключем до ефективності таких систем [14].

Загалом огляд і аналіз наявних аналогів свідчить про значний потенціал чат-ботів у сфері рекомендацій пізнавального матеріалу, але водночас вказує на необхідність подальших досліджень для подолання існуючих обмежень. Майбутні розробки мають зосередитися на підвищенні точності, етичності використання даних і розширенні можливостей адаптації під різні освітні контексти, що дозволить зробити процес пізнання ще більш ефективним і доступним для кожного.

1.3 Постановка задачі створення чат-бота для рекомендацій пізнавального матеріалу

У сучасному інформаційному суспільстві, де обсяги доступного контенту зростають експоненційно, особливо гостро постає проблема ефективного відбору пізнавального матеріалу, здатного задовольняти індивідуальні потреби користувачів у самоосвіті та когнітивному розвитку [15]. Традиційні пошукові системи та рекомендаційні платформи часто не враховують контекст запитів, рівень підготовки особи, її попередні вподобання чи мовні особливості, що призводить до інформаційного перевантаження або, навпаки, до ігнорування цінних ресурсів.

Фільми, книги та музичні твори, які несуть значний пізнавальний потенціал – від документальних стрічок і науково-популярної літератури до композицій, що розкривають культурні та історичні контексти, – стають важливими інструментами для розширення світогляду, розвитку емпатії та критичного мислення. Саме тому створення чат-бота, орієнтованого на рекомендації такого матеріалу, набуває не лише практичного, а й наукового значення, адже воно дозволяє поєднати зручність повсякденного спілкування з передовими можливостями штучного інтелекту для персоналізованого супроводу користувача в процесі пізнання [16].

Постановка задачі передбачає чітке формулювання мети, яка полягає в розробці інтелектуальної системи, здатної забезпечити природну, ефективну та адаптивну взаємодію з користувачем для підбору пізнавального контенту. Така система має розуміти запити, сформульовані природною українською мовою, виявляти їхню семантичну сутність, визначати відповідну категорію матеріалів і генерувати рекомендації, що максимально відповідають індивідуальному профілю. Це дозволяє подолати бар'єри, пов'язані з мовними відмінностями, складністю пошуку та відсутністю персоналізації, роблячи процес самоосвіти доступнішим і привабливішим для широкого кола людей.

Важливим аспектом є забезпечення зворотного зв'язку, який дає змогу системі вдосконалюватися на основі реакцій користувача, поступово формуючи точніший портрет його вподобань і сприяючи довготривалому когнітивному зростанню.

Для реалізації поставленої мети необхідно розв'язати низку взаємопов'язаних завдань, що охоплюють різні рівні функціонування системи. Передусім йдеться про забезпечення якісної обробки вхідних повідомлень, включаючи розпізнавання наміру користувача, виявлення ключових сутностей та адаптацію запиту для подальшого пошуку. Далі система повинна ефективно шукати та збагачувати результати з різних джерел, забезпечуючи їхню релевантність і унікальність.

Особливе місце посідає механізм персоналізації, який враховує історію взаємодії, початкові налаштування профілю та оцінки попередніх рекомендацій, дозволяючи поступово підвищувати точність підбору. Не менш важливим є створення зручного інтерфейсу для зворотного зв'язку, зберігання історії рекомендацій та підтримки користувача на всіх етапах, що сприяє формуванню стійкої звички до регулярного використання системи в повсякденному житті.

Таблиця 1.3 відображає ключові елементи мети, завдань та очікуваних результатів, що забезпечують цілісність системи.

Таблиця 1.3 – Основні компоненти постановки задачі створення чат-бота для рекомендацій пізнавального матеріалу

Компонент постановки задачі	**Короткий опис**
Мета системи	Розробка персоналізованого чат-бота для рекомендацій пізнавального контенту у формі фільмів, книг та музики
Основні функціональні вимоги	Розуміння природної мови, категорізація запитів, семантичне ранжування, персоналізація на основі профілю
Технічні аспекти	Обробка запитів українською, інтеграція зовнішніх джерел, механізми кешування та зворотного зв’язку
Очікувані результати	Підвищення ефективності самоосвіти, зростання задоволеності користувачів, формування стійкого профілю вподобань

Ця таблиця ілюструє структурований підхід до формулювання задачі, підкреслюючи баланс між технічними можливостями та користувацькими потребами.

Схема на рисунку 1.3 відображає взаємозв’язок основних блоків: від обробки запиту до персоналізованого результату та зворотного зв’язку.

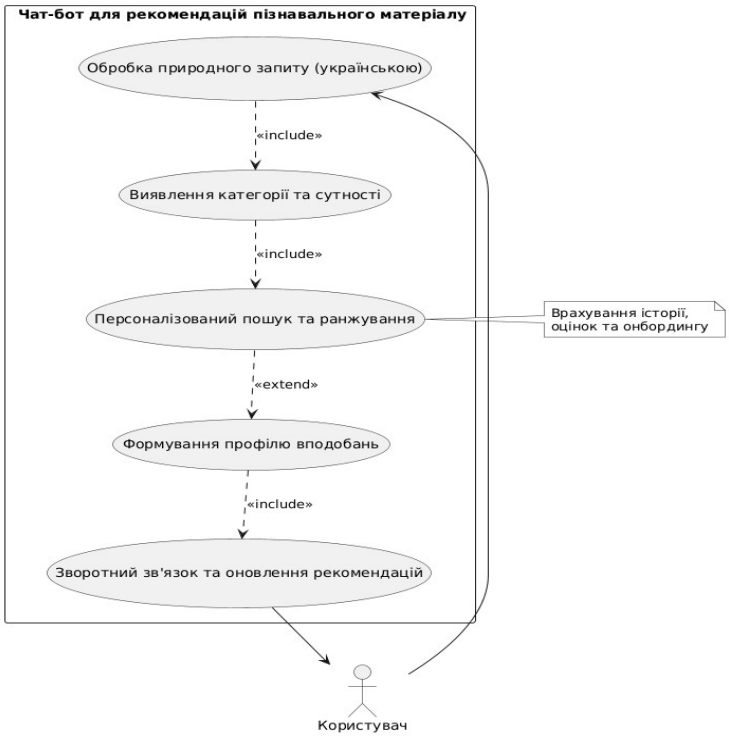

Рисунок 1.3 – Загальна схема постановки задачі чат-бота для рекомендацій пізнавального матеріалу

Рисунок 1.3 наочно демонструє логіку взаємодії компонентів, де кожний етап є логічним продовженням попереднього, забезпечуючи цілісність процесу рекомендацій.

Таким чином, постановка задачі не обмежується лише технічною реалізацією, а охоплює ширший гуманістичний контекст – створення інструменту, який робить пізнавальний матеріал доступним, мотивуючим і адаптованим до реальних потреб сучасної людини. Це сприяє не лише індивідуальному розвитку, але й загальному підвищенню освітнього рівня суспільства, особливо в умовах швидких змін цифрового середовища [17].

Система повинна бути гнучкою, масштабованою та орієнтованою на довгострокову взаємодію, що дозволить користувачам відчувати справжню підтримку в процесі пізнання. Крім того, важливим є врахування етичних аспектів, таких як конфіденційність даних профілю, точність рекомендацій та уникнення упередженості, щоб забезпечити довіру та безпеку використання [18].

Подальший розвиток такого чат-бота відкриває перспективи інтеграції з іншими освітніми сервісами, що ще більше розширить його вплив на сферу самоосвіти. У підсумку, правильно сформульована задача стає фундаментом для створення ефективного, користувацько-орієнтованого інструменту, який гармонійно поєднує технологічні інновації з гуманітарними цінностями доступності знань [19].

Реалізація цих завдань вимагає комплексного підходу, що поєднує лінгвістичні, алгоритмічні та психолого-педагогічні аспекти, забезпечуючи не лише технічну досконаліть, але й реальну користь для кожної людини, яка прагне збагачувати свій інтелект через якісний пізнавальний контент [20]. Саме така постановка задачі гарантує, що чат-бот стане надійним супутником у світі інформації, допомагаючи перетворювати пасивне споживання контенту на активний процес когнітивного зростання [21].

Метою роботи є розробка чат-бота Virgil, як ефективного інструменту самоосвіти, для формування рекомендацій пізнавального матеріалу.

Об’єктом роботи є процеси створення рекомендацій.

Предметом роботи є технології розробки чат-бота з функцією рекомендацій пізнавального матеріалу.

Висновки до розділу 1

Сучасне інформаційне суспільство характеризується експоненційним зростанням обсягів контенту, що актуалізує проблему ефективного доступу до якісних пізнавальних матеріалів. Чат-боти відкривають нові можливості персоналізованої рекомендації, перетворюючи пасивний пошук на динамічний діалог, який враховує індивідуальні когнітивні потреби, рівень знань, контекст запиту та динаміку інтересів користувача. Вони значно перевершують традиційні рекомендаційні системи за рівнем інтерактивності, адаптивності та зручності, забезпечуючи природну мовну взаємодію й постійне вдосконалення на основі зворотного зв’язку.

Аналіз наявних аналогів підтверджує значний потенціал таких інструментів у сфері освіти та самоосвіти, зокрема їхню здатність формувати індивідуальні траєкторії навчання та підтримувати мотивацію. Водночас виявляються суттєві обмеження щодо точності семантичного аналізу, глибини персоналізації, культурно-мовної адаптації та захисту даних, що особливо актуально для україномовного середовища.

Постановка задачі створення чат-бота передбачає розробку інтелектуальної системи для рекомендації пізнавального контенту у формі фільмів, книг і музики. Система забезпечує семантичне розуміння запитів українською мовою, ефективну персоналізацію на основі історії взаємодії та механізми постійного поліпшення. Реалізація такого підходу сприятиме подоланню інформаційного перевантаження, формуванню критичного мислення та підвищенню доступності якісних пізнавальних ресурсів для широкої аудиторії в умовах цифрової трансформації.

2 МОДЕЛІ, МЕТОДИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ РЕАЛІЗАЦІЇ ЧАТ-БОТА РЕКОМЕНДАЦІЙ

2.1 Методи обробки природної мови та формування рекомендацій у пізнавальній сфері

У сучасних системах чат-ботів для рекомендацій пізнавального матеріалу ключове місце посідає інтеграція методів обробки природної мови з алгоритмами персоналізованого формування пропозицій, що дозволяють перетворювати довільні запити українською мовою на релевантний контент, такий як фільми, книги чи музичні альбоми, які сприяють когнітивному розвитку, розширенню світогляду та емоційному збагаченню особистості [22].

Чат-бот Virgil втілює гібридний підхід, де первинна обробка запиту починається з класифікації інтенту за допомогою спеціалізованої рекурентної нейронної мережі на основі архітектури довгої короткочасної пам’яті, навченої на великому корпусі українських фраз. Цей механізм забезпечує швидке розпізнавання системних намірів користувача, таких як привітання, прохання про довідку чи продовження попередньої розмови, без залучення ресурсомістких зовнішніх сервісів, що є особливо важливим для підтримання природного діалогу в освітньо-пізнавальному контексті [23].

Після ідентифікації рекомендаційного інтенту система переходить до глибшого семантичного аналізу за допомогою оркестратора, що базується на великій мовній моделі. Цей компонент виконує одночасний переклад запиту з української на англійську, точне визначення категорії контенту та генерацію конкретного плану пошуку у вигляді списку назв або тематичних ключів, враховуючи попередні вподобання користувача [24].

Такий підхід дозволяє не лише зрозуміти поверховий зміст фрази, але й врахувати контекст, наприклад, уточнення теми, настрою чи жанру, що робить рекомендації більш адаптованими до індивідуальних пізнавальних потреб. Для забезпечення точності застосовується також правиловий аналіз ключових слів,

який перекриває потенційні помилки машинного перекладу та явно виділяє категорії на кшталт фільму, книги чи музики безпосередньо з оригінального тексту запиту.

Отриманий план використовується для звернення до спеціалізованих джерел даних, де для кожної категорії задіяно оптимальні інформаційні ресурси. Так, для фільмів і серіалів залучається структурований пошук через API TMDB, що надає детальні описи, рейтинги та постери; для музики – сервіс Last.fm з акцентом на альбоми та виконавців; для книг – Google Books з резервним переходом на Open Library у разі обмежень доступу [25]. Ці джерела збагачуються додатковим контентом із загального веб-пошуку через DuckDuckGo, що дозволяє охопити широкий спектр пізнавального матеріалу, від класичних літературних творів до сучасних документальних стрічок.

Усі результати кешуються на рівні пошукових запитів з урахуванням індивідуального профілю користувача, що суттєво прискорює повторні звернення та знижує навантаження на зовнішні сервіси.

Семантичне переранжування отриманих кандидатів здійснюється за допомогою моделі SentenceTransformer, спеціально адаптованої для української мови, яка перетворює заголовки, описи та повні тексти на векторні представлення в багатовимірному просторі [26]. Обчислення косинусної подібності між вектором запиту та векторами контенту дозволяє виділити найбільш релевантні позиції, навіть якщо формулювання користувача суттєво відрізняється від стандартних описів. Цей етап є критичним для пізнавальної сфери, оскільки забезпечує не лише формальну відповідність, а й глибоке тематичне узгодження, наприклад, рекомендацію книг чи фільмів, що розвивають однакові когнітивні навички чи розкривають подібні ідеї.

Паралельно система формує персоналізований вектор переваг користувача за допомогою енкодера сесій на базі GRU-мережі [27]. Ця рекурентна модель обробляє послідовність останніх взаємодій, де кожен елемент поєднує семантичне

представлення контенту з сигналом зворотного зв’язку від користувача – позитивним, негативним чи нейтральним.

Таким чином, вектор вподобань постійно оновлюється після кожного оцінювання рекомендації, що дозволяє чат-боту з часом краще розуміти індивідуальні пізнавальні інтереси та уникати повторів уже переглянутого матеріалу за допомогою механізму дедуплікації на основі історії. Завершальне поєднання семантичного та персоналізаційного балів відбувається в комбінуючій моделі, де фінальний рейтинг відображає як релевантність запиту, так і відповідність профілю користувача, забезпечуючи оптимальний баланс між новизною та точністю [28].

Для візуального представлення загальної послідовності обробки природної мови та формування рекомендацій у чат-боті Virgil доцільно звернутися до схеми, що ілюструє основні етапи пайплайну.

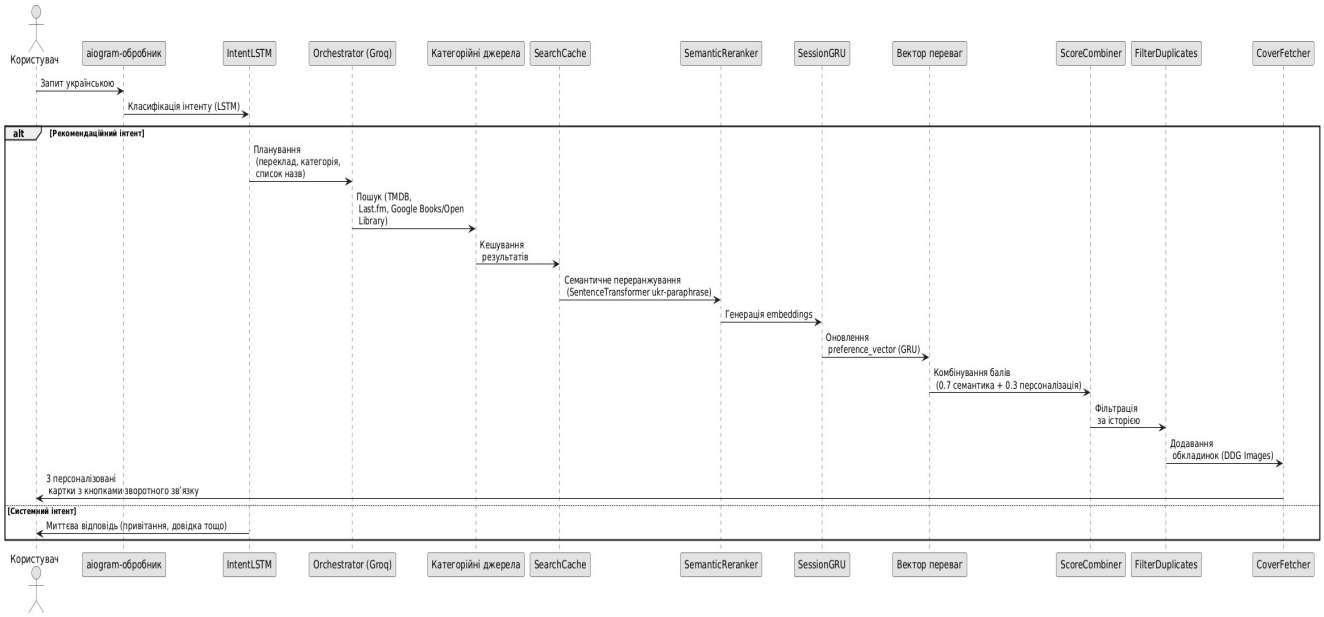

Рисунок 2.1 – Пайплайн обробки природної мови та формування рекомендацій у чат-боті Virgil

Як видно зі схеми на рисунку 2.1, весь процес утворює замкнений цикл, де зворотний зв’язок від користувача безпосередньо впливає на оновлення вектора

переваг, забезпечуючи постійне вдосконалення системи для кращого задоволення пізнавальних запитів.

Додатково ефективність різних нейромережевих компонентів можна проілюструвати через порівняльну характеристику, наведену в таблиці 2.1, що підкреслює їхню роль у реалізації рекомендаційного механізму.

Таблиця 2.1 – Основні нейромережеві моделі, застосовані в чат-боті Virgil для обробки природної мови та персоналізації

Модель	**Архітектура**	**Основне призначення**	**Внесок у пізнавальні рекомендації**
IntentLSTM	LSTM (рекурентна)	Класифікація інтенту запиту	Швидке розпізнавання рекомендаційних намірів
ukr-paraphrase-multilingual-mpnet-base	SentenceTransformer	Семантичне переранжування кандидатів	Глибоке тематичне узгодження контенту
SessionGRU	GRU (рекурентна)	Кодування історії сесії в вектор переваг	Довгострокова персоналізація під когнітивні інтереси

Ця таблиця наочно демонструє, як поєднання рекурентних мереж для класифікації та персоналізації з трансформерними моделями для семантичного аналізу створює потужний інструмент для рекомендацій пізнавального матеріалу.

Окрім того, важливим елементом є онбординг нового користувача, під час якого через інтерактивну анкету збираються початкові дані про категорійні та жанрові вподобання, що негайно трансформується в стартовий вектор переваг. Це дозволяє системі вже з першого запиту пропонувати релевантний контент, мінімізуючи ефект холодного старту.

Механізм зворотного зв’язку через кнопки оцінювання кожної рекомендації не лише фіксує реакцію, але й автоматично запускає перерахунок вектора після певної кількості взаємодій, що робить чат-бота справжнім персональним помічником у сфері самоосвіти та культурного збагачення.

Таким чином, методи обробки природної мови та формування рекомендацій, реалізовані в чат-боті Virgil, демонструють ефективне поєднання класичних нейромережевих архітектур з сучасними трансформерними підходами та великими мовними моделями, що забезпечує високу точність, персоналізацію та адаптивність до потреб користувачів у пізнавальній сфері.

Завдяки цьому система не лише відповідає на запити, але й сприяє тривалому розвитку когнітивних навичок, роблячи процес навчання цікавим, доступним і індивідуально орієнтованим.

2.2 Технології розробки інтелектуального чат-бота з функцією рекомендацій

Розробка інтелектуального чат-бота, здатного надавати персоналізовані рекомендації пізнавального матеріалу у формі фільмів, книг та музики, вимагає гармонійного поєднання сучасних технологій, які забезпечують не лише технічну надійність, а й справжню інтелектуальну адаптивність до потреб користувача [29].

У системі Virgil, реалізованій на мові Python, такий підхід втілено через комплексне використання асинхронних фреймворків, реляційних баз даних, нейронних мереж та оркестраторів на базі великих мовних моделей, що дозволяє створювати природний діалог українською мовою та формувати рекомендації, які з часом стають дедалі точнішими. Центральне місце посідає фреймворк aiogram версії 3, який забезпечує ефективну обробку повідомлень, колбеків та станів користувача за допомогою механізму кінцевих автоматів.

Завдяки йому бот реагує на запити в реальному часі, підтримує багатоетапні сценарії, зокрема онбординг з вибором категорій і жанрів для початкового формування профілю вподобань, та інтегрує проміжні шари автентифікації користувача [30].

Для збереження й оперативного доступу до даних про користувачів, історію рекомендацій, відгуки та кешовані результати пошуку застосовується асинхронна об’єктно-реляційна бібліотека SQLAlchemy в поєднанні з базою даних SQLite через

драйвер aiosqlite. Така комбінація дозволяє реалізовувати репозиторний шаблон, де окремі класи – UserRepository, HistoryRepository, FeedbackRepository, SessionStateRepository, SearchCacheRepository та PageCacheRepository – інкапсулюють усі операції з таблицями, забезпечуючи атомарність транзакцій і швидке відновлення стану.

Особливо цінним є зберігання векторів переваг користувача у вигляді бінарних BLOB-об’єктів у таблиці user_session_state, що дає змогу динамічно оновлювати персоналізацію після кожного третього відгуку або за спеціальною командою [31].

Інтелектуальне ядро системи побудоване на двох рівнях нейронних мереж. Перший рівень – гібридний класифікатор інтенту на базі рекурентної нейронної мережі довгої короткочасної пам’яті (LSTM), реалізований за допомогою бібліотеки PyTorch. Модель, навчена на україномовних фразах, миттєво розпізнає системні наміри – привітання, прохання допомоги, запит додаткових варіантів чи реакцію на попередні рекомендації – без звернення до зовнішніх сервісів, що суттєво прискорює відповідь і зменшує навантаження [32].

Другий рівень персоналізації забезпечує власна модель SessionGRU, також створена на PyTorch, яка перетворює послідовність взаємодій користувача (вектори ембедингів кандидатів плюс сигнали відгуків +1, –1 або 0) на єдиний вектор переваг. Цей вектор оновлюється автоматично й використовується для коригування фінального рангу рекомендацій, роблячи систему чутливою до еволюції смаків людини [33].

Семантичне ранжування кандидатів виконує модель SentenceTransformer з бібліотеки sentence-transformers, зокрема українськомовна версія ukr-paraphrase-multilingual-mpnet-base, яка генерує 768-вимірні векторні представлення текстів і дозволяє порівнювати запит з описами контенту.

Для початкового планування пошуку, перекладу запиту з української на англійську та визначення категорії (фільм, книга чи музика) задіяно оркестратор SearchOrchestrator, що працює через API Groq на базі великої мовної моделі.

Завдяки цьому бот не просто шукає за ключовими словами, а генерує конкретні назви творів з урахуванням контексту користувача – попередніх лайків, дизлайків і недавніх запитів [34]. Пошук фактичного контенту відбувається через спеціалізовані клієнти: DDGSearchClient з бібліотеки ddgs для загального веб-пошуку з фільтрами сайтів, TMDBClient для фільмів і серіалів, LastFMClient для музичних альбомів, GoogleBooksClient та OpenLibraryClient як резерв для книг. Результати кешуються в пошуковому та сторінковому кешах з відповідними термінами життя, що значно підвищує швидкість повторних запитів.

Додаткові компоненти, такі як CoverFetcher для автоматичного пошуку обкладинок через DuckDuckGo Images, ContentExtractor для витягування описів зі сторінок та ScoreCombiner для комбінування семантичного й персоналізаційного балів (0,7 і 0,3 відповідно), завершують пайплайн, забезпечуючи користувачеві три якісні, неповторювані протягом тридцяти днів рекомендації [35].

Як показано на рисунку 2.2, технології утворюють єдиний безперервний ланцюг обробки запиту, де кожна ланка посилює інтелектуальність системи.

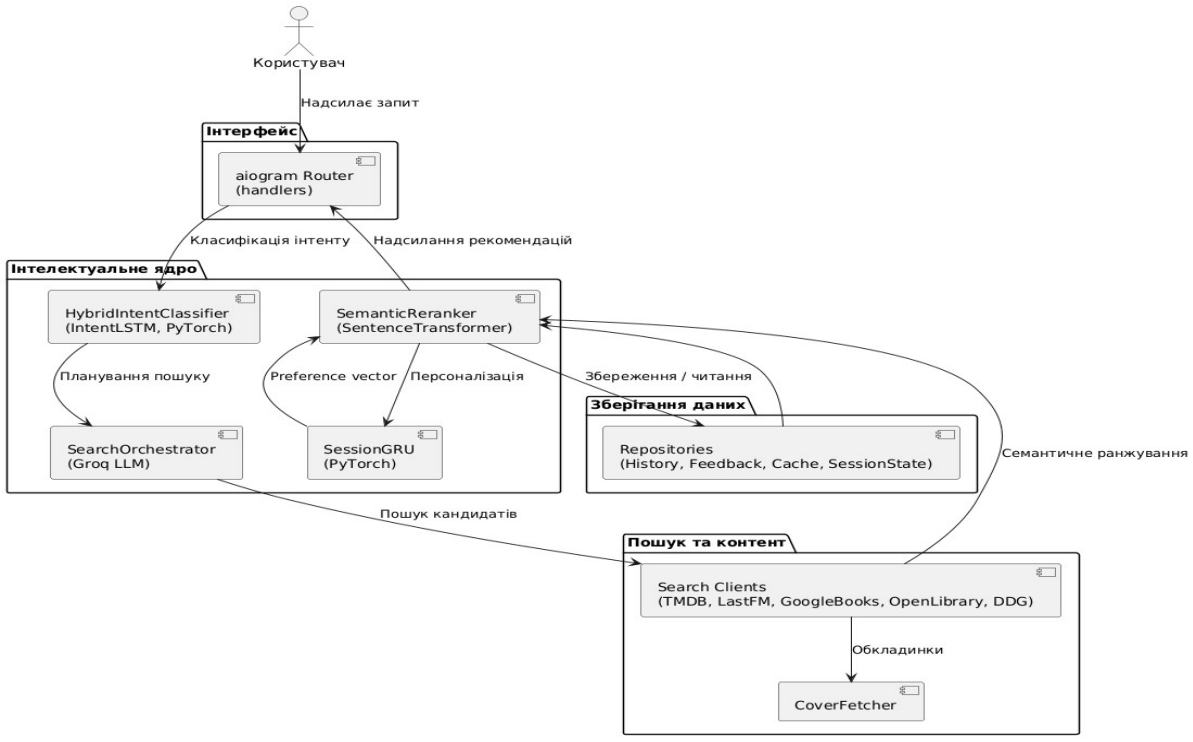

Рисунок 2.2 – Архітектура технологічного пайплайну обробки запиту в чат-боті Virgil

Таблиця 2.2 наочно демонструє розподіл технологій за функціональними блоками системи, підкреслюючи їх внесок у забезпечення інтелектуальних властивостей рекомендацій.

Таблиця 2.2 – Основні технології та бібліотеки, використані для реалізації чат-бота Virgil

Компонент	**Технологія / Бібліотека**	**Призначення в системі рекомендацій**
Бот-фреймворк	aiogram 3 + FSM	Обробка діалогу, станів та інлайн-кнопок
База даних	SQLAlchemy async + aiosqlite	Збереження профілів, історії, кешів
Класифікація інтенту	IntentLSTM (PyTorch)	Миттєве розпізнавання намірів користувача
Переклад і планування	Groq LLM + Helsinki-NLP translator	Генерація точних назв і категорій
Семантичне ранжування	SentenceTransformer (lang-uk model)	Порівняння запиту з описами контенту
Персоналізація	SessionGRU (PyTorch)	Кодування історії в вектор переваг
Пошук контенту	TMDBClient, LastFMClient, GoogleBooksClient, OpenLibraryClient, DDG	Отримання актуальних даних
Кешування та оптимізація	SearchCacheRepository, PageCacheRepository, LRUCache	Зменшення затримок і навантаження

Завдяки такому поєднанню технологій чат-бот не лише відповідає на запити, а й навчається разом з користувачем, враховуючи його реакції та поступово підвищуючи точність рекомендацій. Середовище розробки доповнюють засоби кешування результатів, обмеження частоти запитів через ThrottlingMiddleware та автоматична реєстрація користувачів, що разом створюють стабільну, безпечну та зручну систему.

У підсумку обрані технології дозволяють реалізувати повноцінну інтелектуальну рекомендаційну платформу, яка ефективно працює на звичайному сервері, не вимагаючи надмірних ресурсів, і водночас пропонує користувачеві відчуття персонального помічника, здатного зрозуміти нюанси українських формулювань і адаптуватися до індивідуальних уподобань у сфері пізнавального контенту.

Висновки до розділу 2

Таким чином, реалізований у чат-боті Virgil гібридний підхід поєднує методи обробки природної мови з алгоритмами персоналізованого рекомендаційного формування. Рекурентні нейронні мережі забезпечують швидку класифікацію інтенту та кодування історії взаємодій у вектор переваг, тоді як трансформерні моделі у поєднанні з великою мовною моделлю дозволяють здійснювати глибокий семантичний аналіз запитів українською мовою та генерувати точний план пошуку. Інтеграція спеціалізованих зовнішніх сервісів з семантичним переранжуванням кандидатів на основі векторних представлень сприяє високій релевантності рекомендацій пізнавального матеріалу.

Технологічна реалізація системи ґрунтується на сучасному Python-стеку, що включає асинхронний фреймворк для побудови бота, об’єктно-реляційну бібліотеку для роботи з базою даних та нейромережеві бібліотеки. Використання механізмів кешування, репозиторного шаблону та динамічного оновлення профілю користувача забезпечує ефективність і масштабованість рішення. Механізми онбордингу та зворотного зв’язку дозволяють мінімізувати ефект холодного старту й постійно вдосконалювати персоналізацію.

Отже, поєднання класичних нейромережевих архітектур, трансформерних моделей та оркестрації великих мовних моделей створює потужну основу для інтелектуального чат-бота, здатного ефективно підтримувати когнітивний розвиток і культурне збагачення користувачів.

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ЧАТ-БОТА ТА АНАЛІЗ ОТРИМАНИХ РЕКОМЕНДАЦІЙ

3.1 Опис вхідних даних про вподобання користувача та структура системи чат-бота

Розробка інформаційної системи чат-бота Virgil, спрямованої на рекомендацію пізнавального матеріалу у вигляді фільмів, книг та музики, ґрунтується на глибокому розумінні вподобань користувача як динамічного, багатошарового набору даних, що формуються поступово через безпосередню взаємодію з системою.

Вхідні дані про вподобання надходять до системи передусім під час початкового онбордингу, коли новий користувач обирає категорії контенту та конкретні жанри за допомогою інтерактивних клавіатур. Цей процес, реалізований у відповідному модулі, дозволяє користувачеві вказати, що саме він найчастіше шукає – фільми, книги, музику чи все підряд, – а також відмітити жанри, такі як комедія, драма, фантастика, детектив, рок чи джаз.

Обрані жанри перетворюються на набір англійських фраз-описів, ембедінги яких обчислюються за допомогою моделі семантичного ранжування, а їхнє середнє значення нормалізується та зберігається як початковий вектор вподобань у спеціальній таблиці стану сесії користувача. Одночасно ці дані фіксуються в полі preferences_json таблиці користувачів разом із прапорцем завершення онбордингу, що забезпечує подальше використання інформації без повторного опитування.

Такий механізм дає змогу системі вже на старті формувати персоналізовані рекомендації, навіть за відсутності історії оцінок, і слугує основою для всіх наступних розрахунків.

Надалі вхідні дані про вподобання збагачуються через постійний моніторинг взаємодій користувача з рекомендаціями. Кожна надіслана картка рекомендації супроводжується можливістю поставити оцінку «подобається» або «не подобається», що фіксується у таблиці відгуків із посиланням на конкретний запис

у журналі рекомендацій. Ці оцінки, разом із ембедінгами контенту, отриманими від семантичного ранжувальника, використовуються для перерахунку вектора вподобань за допомогою енкодера сесій на основі рекурентної моделі.

Перерахунок відбувається автоматично після досягнення певного порогу взаємодій або за запитом користувача через спеціальну команду, що дозволяє системі оперативно адаптуватися до змін у смаках. Крім того, система зберігає повну історію запитів і рекомендацій у журналі, де кожен запис містить оригінальний текст запиту українською мовою, його переклад, назву рекомендованого матеріалу, посилання, опис, ембедінг та позицію в результатах. Ця історія слугує не лише для уникнення повторів, але й для формування контексту користувача, що передається до планувальника пошуку разом із переліком улюблених і неулюблених позицій.

Таким чином, вхідні дані про вподобання являють собою комбінацію явних виборів під час онбордингу, неявних сигналів від оцінок та хронологічної послідовності взаємодій, що перетворюється на компактний числовий вектор у просторі ембедінгів для подальшого використання в алгоритмах ранжування.

Структура системи чат-бота Virgil побудована за модульним принципом, що забезпечує чітке розмежування обов’язків між компонентами та полегшує підтримку й розширення функціональності. Центральним елементом є диспетчер на базі фреймворку для створення Telegram-ботів, який реєструє всі обробники повідомлень і колбеків, а також застосовує проміжні шари для реєстрації користувачів і обмеження частоти запитів.

Кожен користувач при першому зверненні автоматично реєструється в базі даних із заповненням базових полів, таких як ідентифікатор у месенджері, ім’я та позначка адміністратора, що дозволяє вести статистику активності та загальну кількість запитів. Обробка вхідних повідомлень відбувається через спеціалізовані маршрутизатори, розділені за функціональним призначенням: один відповідає за початкові команди та онбординг, інший – за обробку відгуків і додаткових

варіантів, третій – за перегляд історії, четвертий – за адміністративні функції, а основний – за повний цикл формування рекомендацій.

Для наочності загальної архітектури системи чат-бота на рисунку 3.1 наведено схему її основних компонентів та взаємозв’язків.

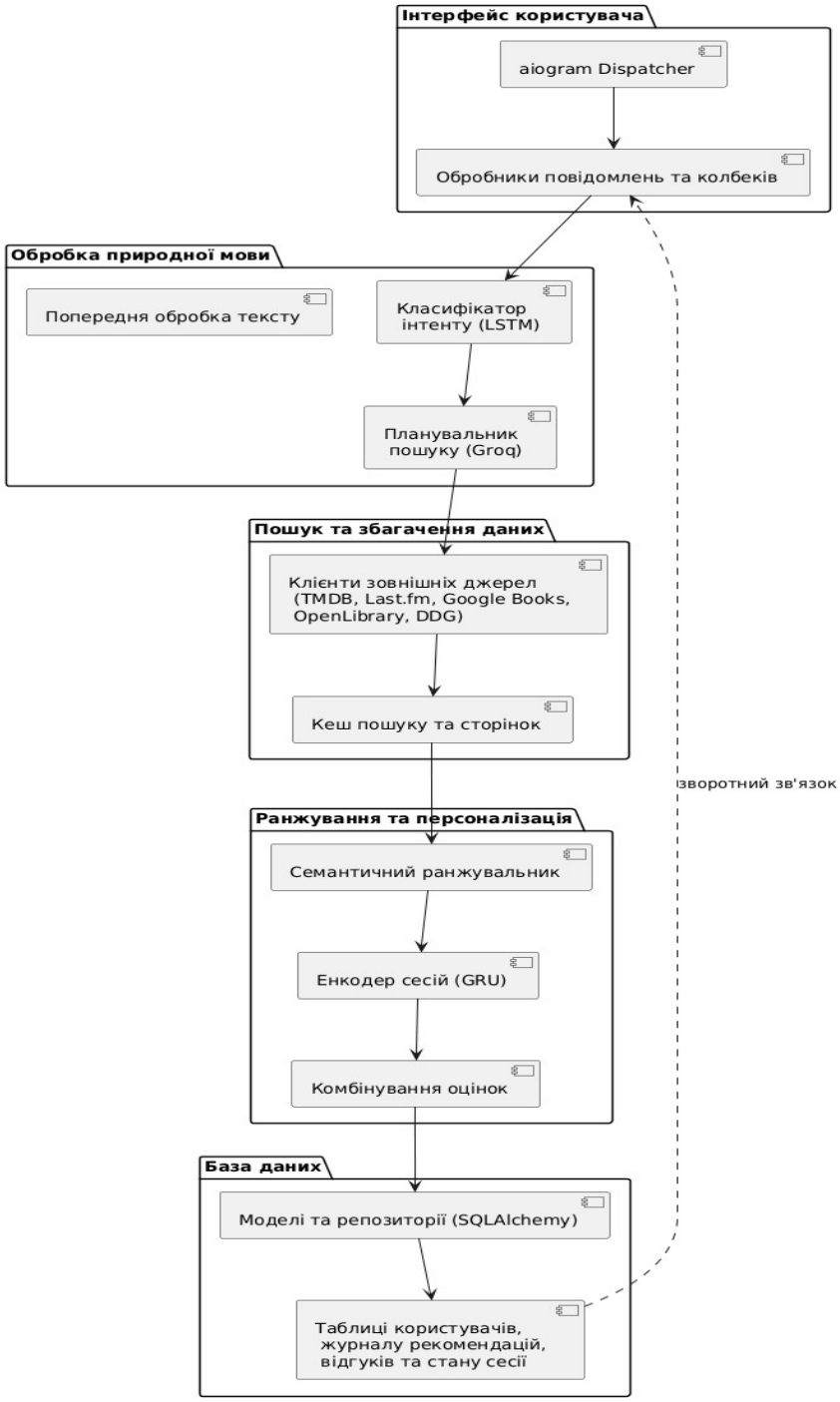

Рисунок 3.1 – Схема архітектури інформаційної системи чат-бота Virgil

Схема ілюструє, як запит користувача проходить через диспетчер до обробників, де спочатку класифікується інтент, потім планується пошук за допомогою зовнішнього сервісу, збираються кандидати з різних джерел, кешуються, ранжуються семантично, персоналізуються за допомогою вектора вподобань і зберігаються в базі даних для подальшого використання.

Така структура забезпечує ефективну обробку навіть складних запитів українською мовою, оскільки попередня обробка тексту включає лематизацію та вилучення сутностей, а планувальник враховує попередній контекст сесії.

База даних відіграє ключову роль у зберіганні та обробці даних про вподобання. Вона складається з кількох взаємопов’язаних таблиць, створених за допомогою об’єктно-реляційного відображення. Таблиця користувачів містить не лише ідентифікаційні дані, але й статистику запитів, статус завершення онбордингу та збережений профіль у форматі JSON.

Журнал рекомендацій фіксує кожен факт видачі контенту разом із семантичними оцінками, персоналізаційними балами та ембедінгами, що дає змогу відтворювати історію та уникати дублювання. Таблиця відгуків забезпечує зв’язок між оцінками користувача та конкретними рекомендаціями, а таблиця стану сесії зберігає актуальний вектор вподобань у бінарному форматі для швидкого доступу під час ранжування. Репозиторії абстрагують доступ до цих даних, реалізуючи методи для підрахунку, оновлення вектора, очищення кешу та отримання недавньої історії з ембедінгами.

Процес формування рекомендацій починається з обробки вхідного повідомлення в стані очікування. Система перевіряє швидкі інтенти за допомогою класифікатора на основі рекурентної мережі, що дозволяє миттєво реагувати на привітання, прохання про довідку чи прохання показати ще варіанти без звернення до важких обчислень. Якщо виявлено запит на рекомендацію, активується планувальник, який виконує переклад запиту, визначає категорію та генерує список конкретних назв контенту з урахуванням профілю користувача.

Отримані кандидати збагачуються метаданими та обкладинками з відповідних джерел, кешуються для повторного використання, а потім проходять семантичне ранжування та комбінування балів. Фінальний список формується з урахуванням вектора вподобань, що забезпечує персоналізацію на рівні 30 відсотків від загальної оцінки, тоді як семантична релевантність становить 70 відсотків. Після видачі карток система зберігає результати в журналі, готуючи дані для майбутніх оновлень профілю.

Важливим аспектом структури є наявність кешування на рівні пошуку та сторінок, що суттєво знижує навантаження на зовнішні сервіси та прискорює відповідь. Кеш пошуку зберігає результати як для загальних, так і для персоналізованих запитів із терміном дії шість годин, а кеш сторінок – на сім днів. Фонова задача періодично очищає прострочені записи, підтримуючи актуальність даних. Крім того, система передбачає адміністративні інструменти для моніторингу статистики користувачів, очищення кешів, перевірки працездатності та розсилки повідомлень, що робить її зручною для обслуговування.

Таким чином, структура інформаційної системи чат-бота Virgil гармонійно поєднує збір вхідних даних про вподобання користувача на всіх етапах взаємодії з модульною архітектурою, що забезпечує надійну роботу, персоналізацію рекомендацій та можливість аналізу отриманих результатів.

Завдяки інтеграції сучасних підходів до обробки природної мови, рекурентних моделей для моделювання вподобань та ефективного кешування система здатна надавати релевантний пізнавальний контент, постійно вдосконалюючись на основі реальних реакцій користувачів. Цей підхід не тільки підвищує задоволеність від використання, але й створює основу для подальшого розвитку функціональності, зокрема розширення джерел контенту та глибшого аналізу поведінки аудиторії.

3.2 Демонстрація роботи розробленого чат-бота для пізнавальних матеріалів

Розроблена інформаційна система чат-бота для рекомендацій пізнавального матеріалу забезпечує зручну та ефективну взаємодію користувачів з контентом у форматі фільмів, книг і музики, який сприяє розширенню знань, розвитку смаків та особистісному зростанню. Інтерфейс бота побудовано на основі стандартних можливостей месенджера, де всі елементи керування – від текстових повідомлень до інтерактивних клавіатур – адаптовані для природного спілкування українською мовою.

Користувач може розпочати роботу з простого запиту у вільній формі, а система автоматично обробляє його, враховуючи попередні взаємодії для підвищення точності пропозицій. Такий підхід робить бот доступним навіть для тих, хто не має досвіду роботи з подібними сервісами, і демонструє практичну реалізацію персоналізованих рекомендацій у реальному часі.

Коли користувач вперше звертається до бота або продовжує пошук пізнавального матеріалу у форматі книги, він надсилає текстовий запит, що запускає основний цикл обробки. На цьому етапі чат відображає введене повідомлення безпосередньо в історії розмови, де бот очікує на подальші дії в режимі очікування рекомендацій. Запит може містити вказівку на тему, жанр чи автора, що дозволяє системі швидко визначити необхідну категорію без додаткових уточнень. Як показано на рисунку 3.2, екранна форма запиту на пошук книги ілюструє початковий момент взаємодії, коли користувач вводить свій намір у полі повідомлення, а бот готовий до негайної реакції.

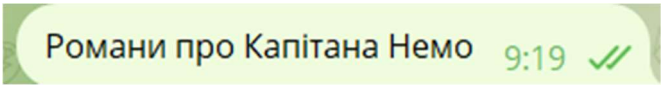

Рисунок 3.2 – Запит на пошук книги

Після надсилання запиту на книгу система переходить до активної фази обробки, під час якої користувач бачить тимчасове інформаційне повідомлення, що повідомляє про хід виконання завдання. Це повідомлення з'являється в чаті як окреме текстове сповіщення і залишається видимим доти, доки не завершиться пошук у джерелах даних, переклад запиту та формування списку пропозицій. Такий підхід запобігає сприйняттю затримки як технічної проблеми та підтримує увагу користувача до процесу. На рисунку 3.3 чітко видно цей етап, де текст «Шукаю для тебе...» займає центральне місце в інтерфейсі, підкреслюючи динамічність роботи бота.

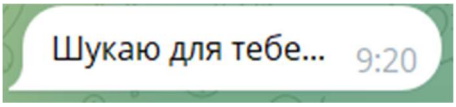

Рисунок 3.3 – Процес пошуку

Завершення пошуку книг супроводжується виведенням готових рекомендацій у форматі карток, які містять усі необхідні дані для швидкого ознайомлення. Кожна картка починається з позначки категорії, за нею йде виділена назва твору, а нижче – метадані, такі як ім'я автора, рік видання та рейтинг. Короткий опис тексту дозволяє користувачеві зрозуміти суть пропозиції, а гіперпосилання забезпечує перехід до детальнішої інформації. Під кожною карткою розміщено набір кнопок для оцінки та отримання додаткових варіантів, що робить взаємодію інтерактивною та сприяє накопиченню даних для подальшого вдосконалення персоналізації. Як видно на рисунку 3.4, відображення результатів пошуку книги демонструє повноцінний результат обробки, де візуальні елементи, такі як обкладинки, доповнюють текстову інформацію і створюють приємне враження від використання.

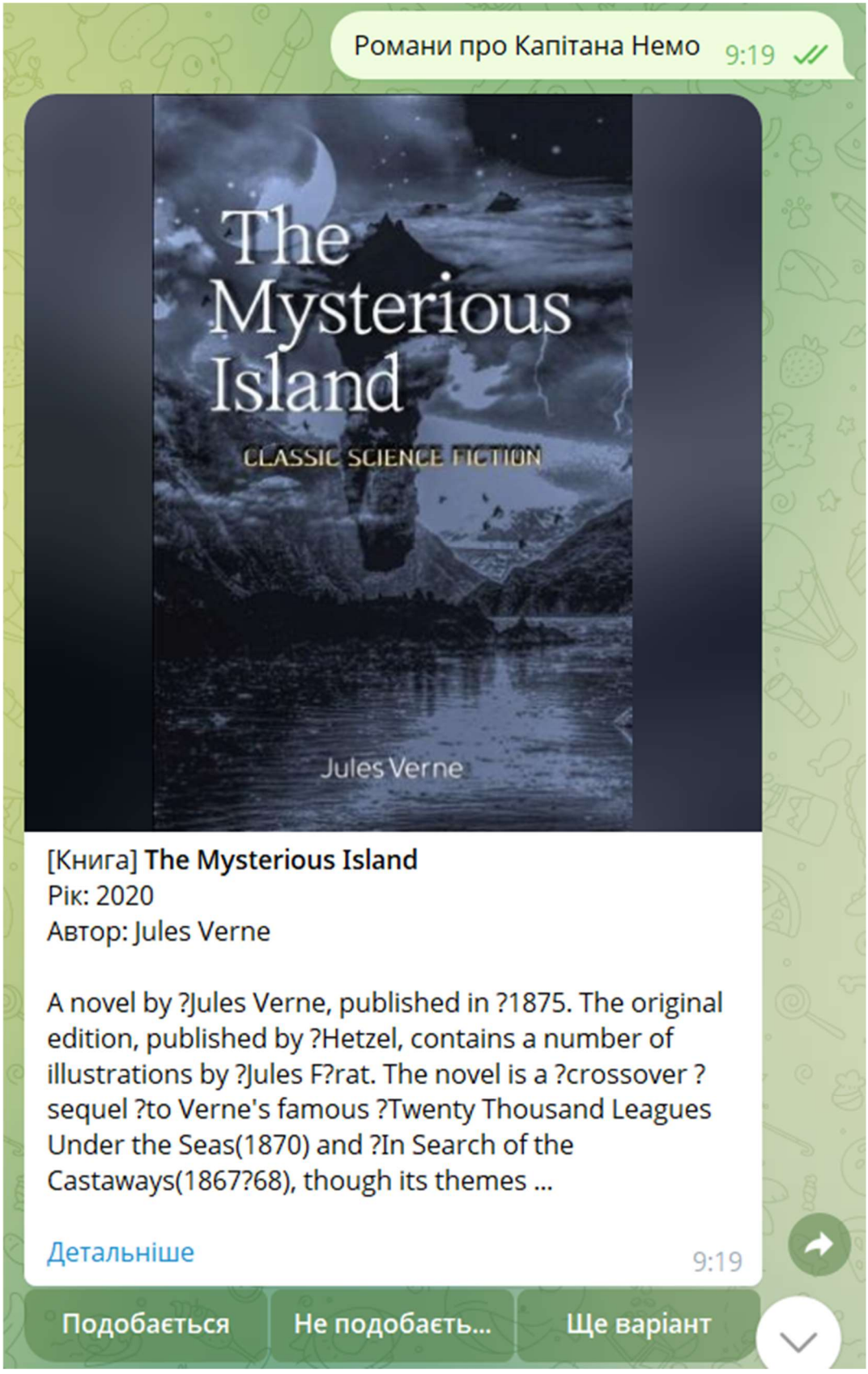

Рисунок 3.4 – Відображення результатів пошуку книги

Аналогічний процес відбувається під час пошуку фільмів як форми пізнавального матеріалу, де користувач вводить запит, спрямований на кіно чи серіали. Система розпізнає намір і ініціює відповідний ланцюжок дій, починаючи з визначення категорії. На екрані чату з'являється повідомлення користувача, яке запускає весь цикл рекомендацій. Як показано на рисунку 3.5, запит на пошук фільму відображається в природному контексті розмови, де бот готовий надати персоналізовані пропозиції на основі теми, жанру чи актора.

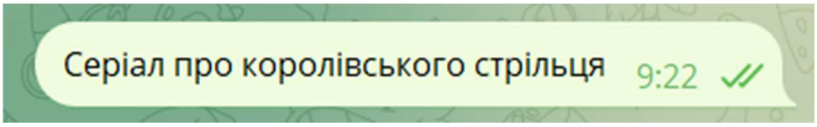

Рисунок 3.5 – Запит на пошук фільму

Після обробки запиту на фільм бот виводить рекомендації у вигляді карток, збагачених візуальними елементами. Кожна пропозиція містить категорію, назву, рік випуску, рейтинг та короткий опис сюжету, що допомагає користувачеві швидко оцінити релевантність. Обкладинки фільмів або постери посилюють візуальне сприйняття, а гіперпосилання ведуть до додаткових деталей. Інтерактивні кнопки під картками дозволяють одразу надати оцінку або попросити альтернативні варіанти, що інтегрується в систему персоналізації. На рисунку 3.6 відображено результати пошуку фільму, де чітко видно структуру карток і елементи керування, що забезпечують зручність користування.

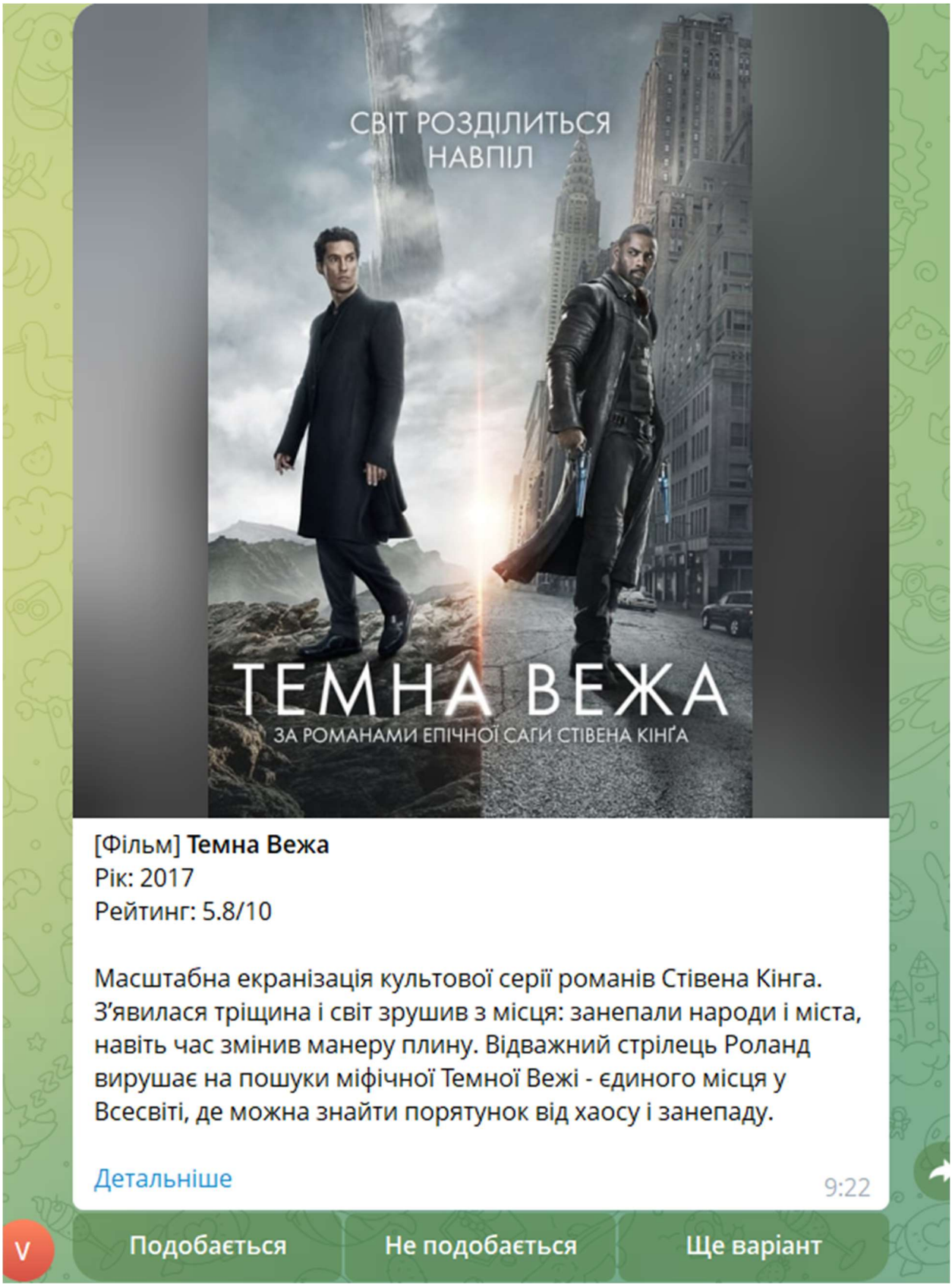

Рисунок 3.6 – Відображення результатів пошуку фільму

Пошук музики як пізнавального матеріалу також демонструє гнучкість системи, коли користувач формулює запит щодо альбомів, виконавців чи жанрів. Бот автоматично класифікує повідомлення і запускає пошук у спеціалізованих джерелах.

Екранна форма запиту на музику відображає введений текст у чаті, де система готова сформувати рекомендації відповідно до уподобань. Як ілюструє рисунок 3.7, запит на пошук музики виглядає природно і запускає повний цикл обробки з урахуванням контексту попередніх взаємодій.

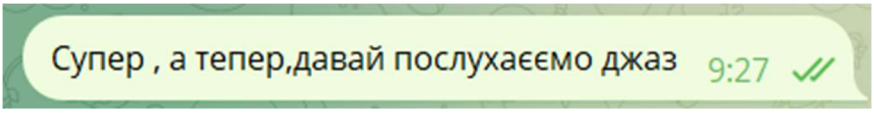

Рисунок 3.7 – Запит на пошук музики

Фінальним етапом рекомендацій музики є презентація результатів у форматі карток, де кожна пропозиція включає категорію, назву альбому, ім’я виконавця, короткий опис та обкладинку.

Метадані можуть містити жанрові теги, а гіперпосилання ведуть до прослуховування чи детальнішої інформації. Кнопки оцінки та запиту додаткових варіантів забезпечують зворотний зв’язок, який використовується для оновлення профілю користувача.

На рисунку 3.8 чітко видно відображення результатів пошуку музики, де візуальна привабливість і структурованість інформації сприяють комфортному сприйняттю пропозицій.

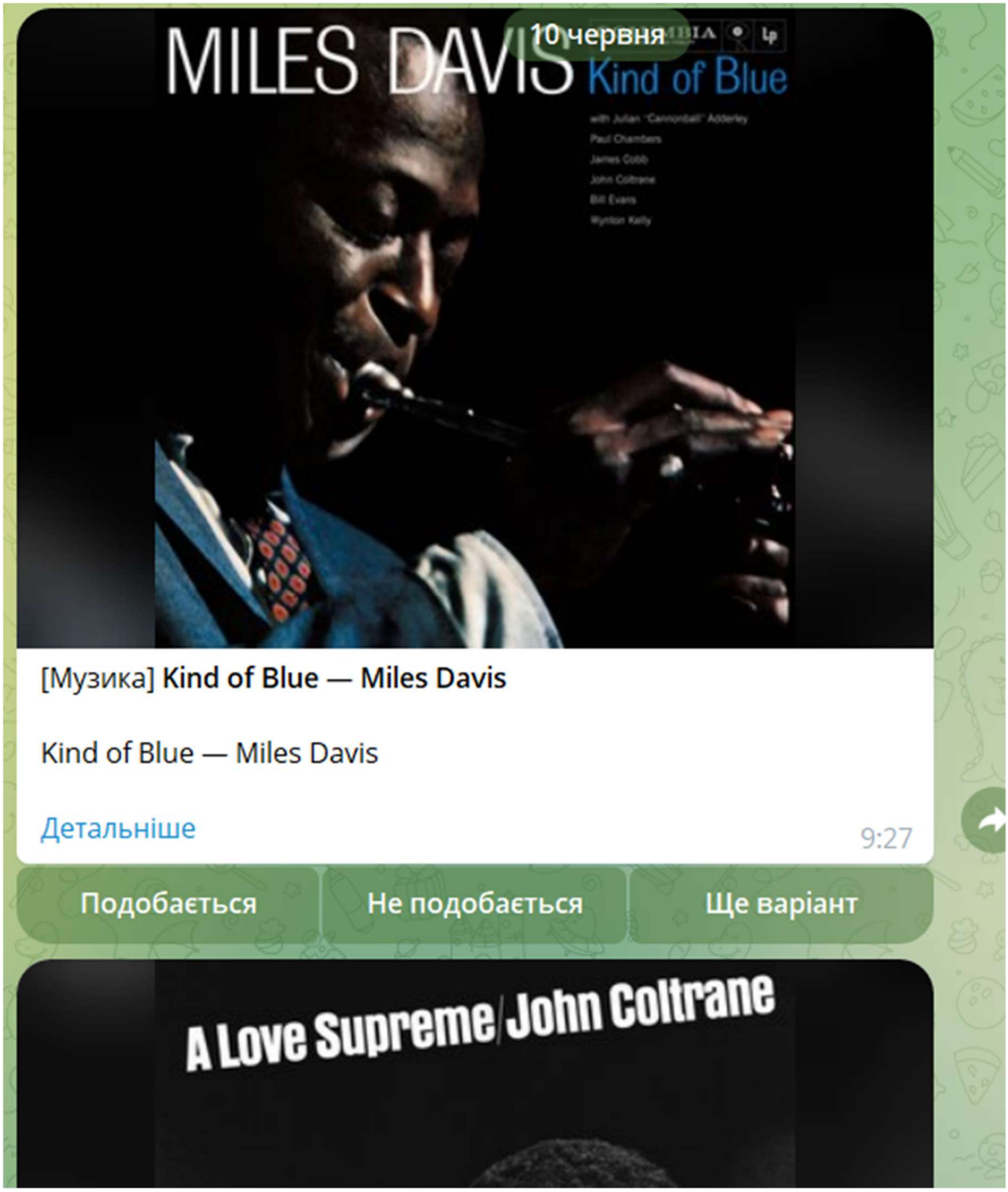

Рисунок 3.8 – Відображення результатів пошуку музики

Система також підтримує додаткові можливості, такі як перегляд історії рекомендацій та оновлення профілю, що дозволяє користувачам повертатися до раніше отриманих пропозицій або коригувати свої вподобання. Після надання оцінки через кнопки бот оновлює внутрішній вектор переваг, що робить наступні рекомендації точнішими і більш персоналізованими.

Такий підхід перетворює чат-бота на динамічний інструмент для постійного відкриття нового пізнавального матеріалу, адаптованого саме під потреби конкретної людини. Загалом демонстрація роботи чат-бота підтверджує його практичну цінність, зручність інтерфейсу та ефективність у наданні рекомендацій, які сприяють інтелектуальному та культурному розвитку користувачів.

3.3 Аналіз отриманих результатів рекомендацій та якості відповідей

Аналіз отриманих результатів рекомендацій та якості відповідей чат-бота Virgil демонструє високий рівень ефективності розробленої інформаційної системи, яка забезпечує персоналізований підбір контенту у формі фільмів, книг та музики, що сприяє когнітивному розвитку користувачів через поєднання розважальних і пізнавальних аспектів культурного досвіду.

Система обробляє запити українською мовою в довільній формі, враховуючи тему, жанр, настрій, конкретні назви чи імена творців, і формує рекомендації, які відповідають індивідуальним вподобанням, що постійно уточнюються завдяки механізмам зворотного зв’язку.

Завдяки інтеграції сучасних підходів до обробки природної мови, семантичного аналізу та персоналізації на основі рекурентних моделей результати рекомендацій вирізняються високою релевантністю навіть за умов холодного старту, коли користувач вперше звертається до бота.

Онбординг, реалізований у модулі onboarding.py, дозволяє швидко сформувати початковий вектор переваг на основі вибору категорій і жанрів, після чого функція _build_initial_preference_vector обчислює середній вектор ембедингів фраз, пов’язаних з обраними жанрами, і зберігає його в базі даних через SessionStateRepository. Це забезпечує миттєву персоналізацію вже на перших запитах, що є ключовим для підтримки зацікавленості користувачів і підвищення якості відповідей з самого початку взаємодії.

Подальше покращення результатів відбувається завдяки гібридному підходу до класифікації інтенту, реалізованому в dispatcher.py та

nlu/intent_classifier/hybrid.py. Модель IntentLSTM, навчена на синтетичних і реальних українських фразах, швидко розпізнає системні інтенти, такі як привітання, довідка чи прохання показати більше варіантів, без залучення зовнішніх сервісів, що гарантує швидкість відповіді та економію ресурсів.

Для рекомендаційних запитів активується SearchOrchestrator, який за допомогою Groq API одночасно виконує переклад запиту на англійську, визначає категорію та генерує план пошуку у вигляді конкретних назв творів з урахуванням контексту користувача (подобані, не подобані та недавні запити).

Такий підхід у функціях _fetch_from_source та _execute_movie_plans, _execute_music_plans, _execute_book_plans дозволяє отримувати значно точніші кандидати порівняно з простим ключовим пошуком, оскільки план враховує семантику запиту та профіль користувача. Наприклад, для запиту про фільм певного актора система спочатку перевіряє персональний план через TMDBClient.search_multi, а потім збагачує результати детальними метаданими, такими як рік випуску, рейтинг та огляд, що підвищує інформативність і когнітивну цінність кожної рекомендації.

Якість рекомендацій суттєво зростає на етапі ранжування та персоналізації, які виконуються в ranking/reranker.py та ranking/score_combiner.py. Семантичний ранжувальник на базі моделі lang-uk/ukr-paraphrase-multilingual-mpnet-base генерує ембедінги для запиту та кандидатів, після чого обчислює косинусну подібність, а функція _rerank_sync сортує результати за семантичним балом.

Водночас SessionGRU з модуля session_encoder інференсує вектор переваг користувача на основі останніх десяти взаємодій, де кожна взаємодія поєднує ембедінг твору та сигнал зворотного зв’язку (+1 для позитивної оцінки, -1 для негативної, 0 для нейтральної).

Функція ScoreCombiner поєднує семантичний бал (вага 0,7) та персоналізаційний (вага 0,3), що забезпечує баланс між загальною релевантністю та індивідуальними смаками. Дедуплікація через filter_duplicates у score_combiner.py, яка перевіряє історію за останні 30 днів за допомогою

HistoryRepository.get_user_seen_urls, запобігає повторенню вже показаних позицій, підвищуючи різноманітність і свіжість рекомендацій.

У результаті топ-3 рекомендації, що відправляються через _send_recommendations, завжди демонструють високу відповідність запиту, а обкладинки, отримані CoverFetcher, роблять візуальне оформлення привабливим і зручним для сприйняття.

Механізм зворотного зв’язку, реалізований у feedback.py, є ключовим для постійного вдосконалення якості. Кожна картка рекомендації супроводжується інлайн-кнопками для оцінки (подобається/не подобається) та запиту додаткових варіантів. Позитивні чи негативні оцінки зберігаються в таблиці Feedback, після чого функція _maybe_refresh_profile інкрементує лічильник взаємодій у UserSessionState.

Коли кількість перевищує поріг preference_refresh_threshold (за замовчуванням 3), запускається _rebuild_preference_vector, який збирає останні ембедінги та оцінки, передає їх до SessionEncoder і оновлює вектор переваг. Це створює замкнений цикл навчання, завдяки якому з кожним новим запитом рекомендації стають точнішими.

Адміністративні команди в admin.py, такі як /stats і /health, дозволяють моніторити загальну якість системи: кількість користувачів, розподіл позитивних і негативних оцінок, розмір кешів пошуку та сторінок, а також статус зовнішніх сервісів. Логування метрик через utils/logger.py фіксує події cache_hit, cache_miss, intent_by_orchestrator та feedback_positive/negative, що дає змогу кількісно оцінювати ефективність.

Для ілюстрації основних етапів обробки запиту на рисунку 3.9 наведено схему конвеєра рекомендацій, яка відображає послідовність операцій від отримання повідомлення до відправлення карток.

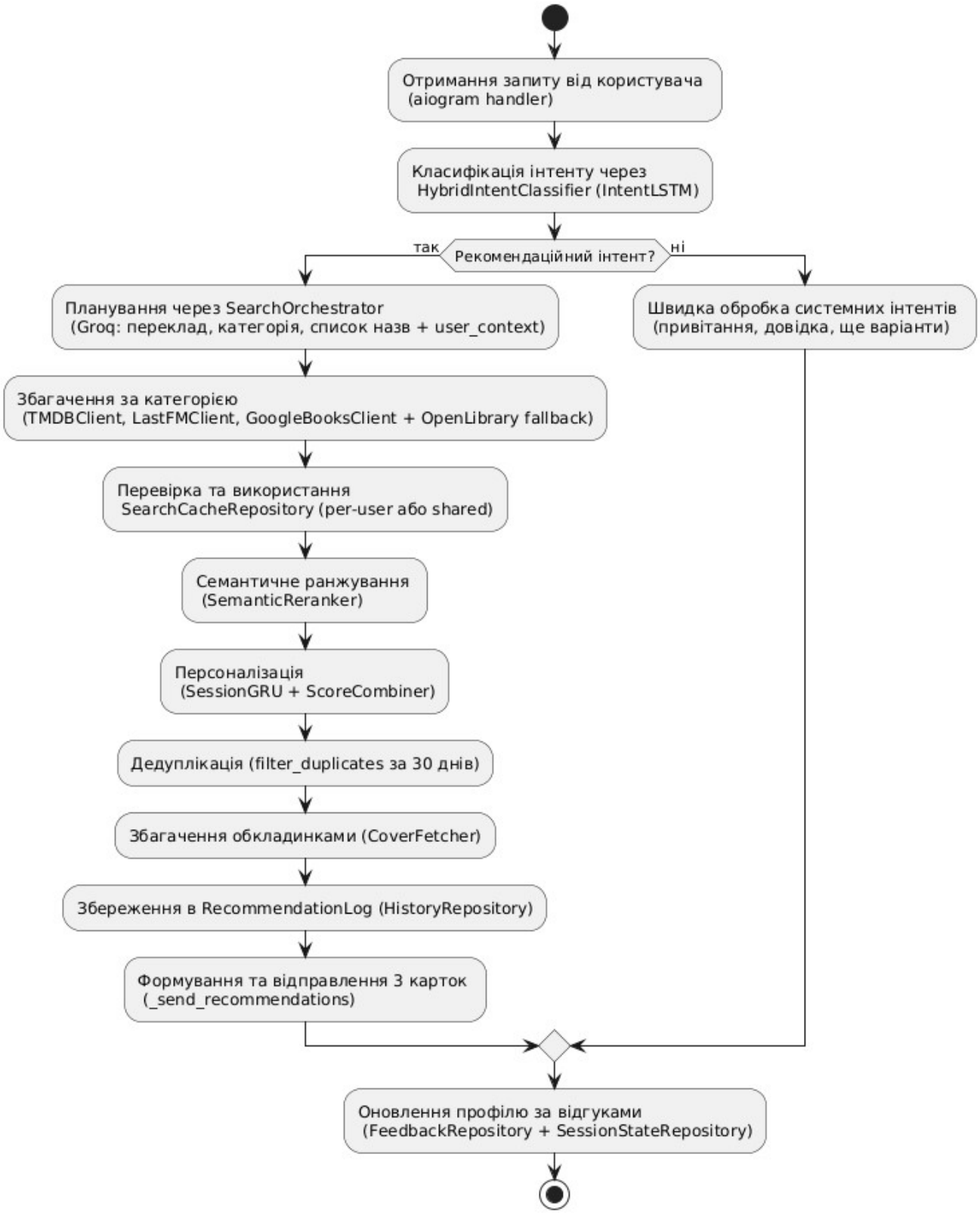

Рисунок 3.9 – Схема конвеєра обробки запиту та формування рекомендацій у чат-боті Virgil

Ця схема наочно демонструє, як інтеграція різних компонентів забезпечує плавний перехід від сирого запиту до персоналізованої відповіді, мінімізуючи затримки завдяки асинхронній архітектурі та кешуванню. У разі системних інтентів відповідь формується миттєво, що підвищує загальну задовільність взаємодії. Для рекомендаційних запитів час відповіді залишається прийнятним завдяки паралельному виконанню завдань збагачення та ранжування, а також використанню asyncio.to_thread для синхронних операцій бібліотек ddgs та SentenceTransformer.

Якість відповідей також визначається зручністю інтерфейсу та стабільністю роботи. Модуль recommendation.py реалізує стани FSM (RecommendationStates), що дозволяє підтримувати контекст сесії: продовження розмови в межах однієї категорії не вимагає повторного визначення категорії, а функція _process_query зберігає last_query та last_translated для повторних уточнень. Клавіатури в inline.py та reply.py пропонують інтуїтивні кнопки для вибору категорій під час онбордингу та оцінювання рекомендацій, що зменшує когнітивне навантаження.

ThrottlingMiddleware обмежує кількість повідомлень (3 за 5 секунд), запобігаючи перевантаженню, а UserRegistrationMiddleware автоматично реєструє нових користувачів і оновлює час активності. Завдяки цьому система стабільно працює навіть за великої кількості одночасних запитів, а помилки обробляються грацiйно: користувач отримує повідомлення про технічні труднощі, а деталі фіксуються в логах.

Для оцінки практичної ефективності доцільно розглянути порівняльні характеристики джерел даних, які використовуються в різних категоріях контенту (таблиця 3.1). Ці джерела забезпечують різну глибину та точність інформації, що безпосередньо впливає на когнітивну цінність рекомендацій.

Таблиця 3.1 – Порівняння характеристик джерел даних за категоріями рекомендацій

Категорія	**Основні джерела**	**Переваги щодо точності та інформативності**	**Внесок у персоналізацію та швидкість**
Фільми	TMDBClient (search_multi, discover, keywords, person credits)	Детальні метадані (рік, рейтинг, огляд), підтримка фільмографії акторів, жанрові фільтри	Високий – ембедінги для SessionGRU, швидке збагачення постерами
Книги	GoogleBooksClient + OpenLibraryClient (fallback)	Пошук за автором і назвою, описи та суб’єкти, підтримка нон-фікшн	Середній – гнучкий fallback забезпечує стабільність
Музика	LastFMClient (search_albums, artist.getTopAlbums, tag.getTopAlbums)	Теги, обкладинки, описи альбомів	Високий – швидкий пошук за жанром і настроєм

Наведені дані підкреслюють, що комбіноване використання спеціалізованих клієнтів разом із семантичним ранжуванням і персоналізацією дозволяє досягати високої релевантності незалежно від категорії. Для фільмів особливо ефективним є розширений discover з ключовими словами та жанрами, що реалізовано в _execute_movie_plans, тоді як для книг fallback на Open Library гарантує результат навіть за обмежень Google Books. Музика виграє від тегів Last.fm, що добре поєднується з настроєвими запитами.

Загальна якість відповідей також підтверджується механізмами моніторингу та логування. Функція cmd_health перевіряє доступність DDG, розмір бази даних та кількість записів у кешах, а cmd_stats надає статистику користувачів і оцінок. Це дозволяє адміністраторам оперативно виявляти потенційні проблеми та підтримувати стабільність.

Крім того, кешування в SearchCacheRepository (TTL 6 годин, персональний або спільний) значно скорочує час відповіді на повторні запити, а PageCacheRepository (TTL 7 днів) оптимізує витягнення описів зі сторінок. У

поєднанні з асинхронною архітектурою aiogram система демонструє високу продуктивність і масштабованість.

Підсумовуючи, аналіз результатів рекомендацій та якості відповідей чат-бота Virgil свідчить про успішну реалізацію комплексного підходу, де інтелектуальні компоненти (LSTM, Groq, SentenceTransformer, SessionGRU) органічно доповнюють спеціалізовані джерела даних і механізми зворотного зв’язку.

Система не лише забезпечує релевантні та персоналізовані рекомендації, але й постійно вдосконалюється завдяки відгукам користувачів, що робить її ефективним інструментом для поширення пізнавального контенту. Отримані результати підтверджують правильність обраної архітектури та відкривають перспективи для подальшого розширення функціональності, зокрема додавання нових категорій контенту та поглиблення аналізу поведінки користувачів.

Висновки до розділу 3

Розроблена інформаційна система чат-бота Virgil забезпечує ефективну персоналізацію рекомендацій пізнавального матеріалу у форматі фільмів, книг і музики на основі динамічного моделювання вподобань користувача. Вхідні дані про переваги формуються шляхом початкового онбордингу з вибором категорій та жанрів, а також постійного збагачення через оцінки рекомендацій і історію взаємодій. Рекурентна модель на базі SessionGRU перетворює ці дані на компактний вектор ембедінгів, що дозволяє оперативно адаптуватися до змін у смаках і забезпечувати релевантні пропозиції навіть за умов холодного старту.

Модульна архітектура чат-бота характеризується чітким розмежуванням компонентів, включаючи диспетчер обробки повідомлень, планувальник пошуку з використанням Groq API, семантичний ранжувальник та репозиторії бази даних. Інтеграція асинхронної обробки, багатошарового кешування та спеціалізованих джерел даних гарантує швидку відповідь на запити українською мовою, мінімізує навантаження на зовнішні сервіси та підвищує масштабованість системи.

Демонстрація роботи системи підтверджує зручність інтерфейсу та природність взаємодії. Користувач отримує структуровані картки рекомендацій з детальними метаданими, обкладинками та інтерактивними елементами керування, що сприяє накопиченню даних для подальшого уточнення профілю та активній участі в процесі персоналізації.

Аналіз результатів рекомендацій свідчить про високу релевантність пропозицій завдяки гібридному підходу до ранжування, де семантична подібність становить 70 %, а персоналізаційний внесок – 30 %. Механізми зворотного зв’язку створюють замкнений цикл навчання, дедуплікація запобігає повторенням, а вбудовані інструменти моніторингу забезпечують стабільність функціонування. Порівняльні характеристики джерел даних для різних категорій контенту підкреслюють їхню комплементарність у досягненні точності та інформативності.

Таким чином, реалізована система чат-бота Virgil поєднує технічну ефективність з високим рівнем користувацької задоволеності, виступаючи дієвим інструментом поширення пізнавального контенту, що сприяє інтелектуальному та культурному розвитку аудиторії і створює надійну основу для подальшого розширення функціональних можливостей.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ЧАТ-БОТА ТА ТЕСТУВАННЯ

4.1 Керівництво користувача по роботі з чат-ботом рекомендацій

Чат-бот Віргіл призначений для надання персоналізованих рекомендацій пізнавального матеріалу у форматі фільмів, книг та музики, повністю адаптований до українського користувача. Система забезпечує інтуїтивне спілкування українською мовою, враховує контекст запитів, поступово накопичує інформацію про вподобання та пропонує релевантні варіанти на основі семантичного аналізу та персоналізації. Користувач починає роботу з простого запуску бота у месенджері Телеграм, після чого система проводить початкове налаштування профілю, що дозволяє вже з перших взаємодій отримувати точніші результати.

При першому запуску через команду старт бот автоматично ініціює процес онбордингу, який складається з двох послідовних кроків і триває менше хвилини. На першому кроці користувачеві пропонується обрати одну або кілька основних категорій контенту, серед яких фільми, книги, музика або варіант усього підряд. Інтерфейс представлений зручними кнопками з позначками вибору, що дозволяють гнучко налаштувати інтереси.

Після підтвердження вибору система переходить до другого кроку, де залежно від обраної категорії відображаються відповідні жанри. Користувач може обрати кілька варіантів, наприклад, для фільмів – комедію, драму, фантастику чи жахи, для книг – детектив, фентезі, романтику або біографію, а для музики – рок, поп, джаз чи електроніку. Завершення цього етапу супроводжується повідомленням про успішне налаштування профілю з переліком обраних жанрів і переходом до основного режиму роботи. Цей процес формує початковий вектор вподобань, що надалі використовується для персоналізації рекомендацій без додаткових налаштувань.

Після завершення онбордингу бот переходить у режим очікування запитів. Користувач може безпосередньо надсилати повідомлення у вільній формі,

наприклад, «порадь фільм жахів», «хочу книгу про кохання» або «щось спокійне послухати» (рисунок 4.1).

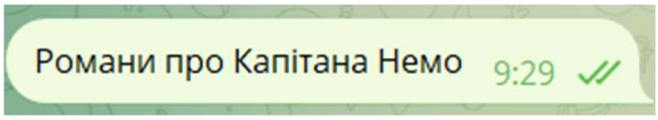

Рисунок 4.1 – Запит користувача

Система автоматично визначає категорію контенту за допомогою вбудованих механізмів розпізнавання, включаючи явні ключові слова та семантичний аналіз. У разі потреби бот запитує уточнення, пропонуючи обрати категорію через клавіатуру з варіантами «Фільм», «Книга» чи «Музика». Після обробки запиту бот відображає три рекомендації у вигляді карток з обкладинками, назвами, короткими описами, метаданими та посиланнями (рисунок 4.2).

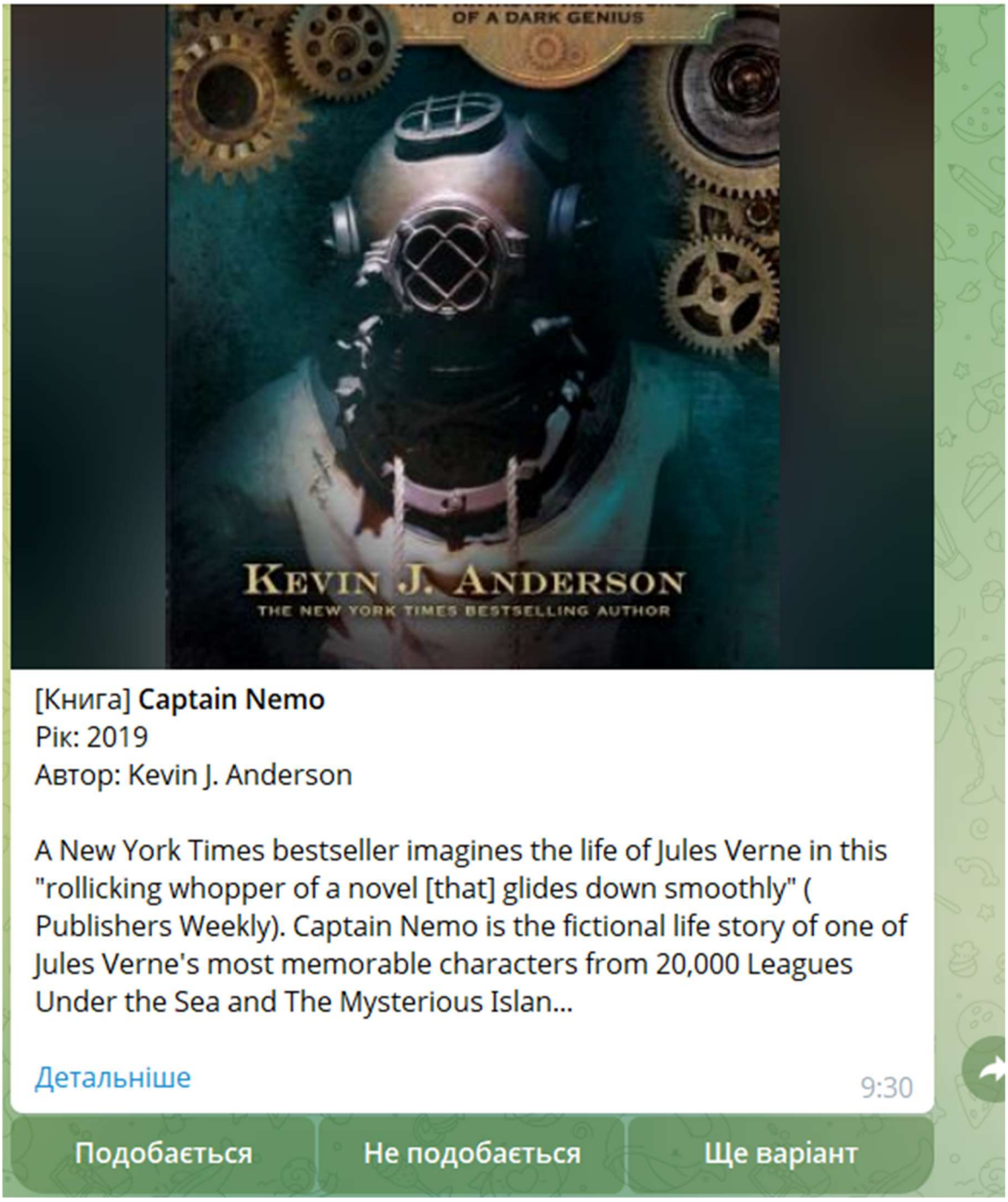

Рисунок 4.2 – Картка рекомендація

Кожна картка супроводжується інтерактивними кнопками для оцінки: «Подобається», «Не подобається» та «Ще варіант» (рисунок 4.3). Оцінки негайно враховуються в профілі, сприяючи покращенню наступних пропозицій.

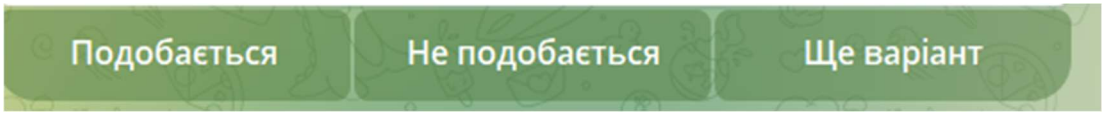

Рисунок 4.3 – Кнопки Оцінки

Механізм зворотного зв’язку є ключовим елементом персоналізації. Коли користувач натискає кнопку «Подобається», система реєструє позитивну реакцію, а при виборі «Не подобається» – негативну, що дозволяє алгоритмам поступово будувати точніший профіль.

Кнопка «Ще варіант» ініціює повторний пошук за тим самим контекстом запиту, надаючи альтернативні рекомендації без необхідності повторювати повідомлення. Усі рекомендації зберігаються в історії, до якої можна отримати доступ у будь-який момент. Крім того, бот підтримує контекстні сесії: якщо користувач продовжує розмову в межах тієї самої категорії, система враховує попередні запити для більш зв’язних результатів.

Для перегляду накопиченої інформації користувач застосовує спеціальні команди. Команда профіль надає детальну статистику, включаючи загальну кількість рекомендацій, розподіл за категоріями, кількість позитивних та негативних оцінок, а також рівень персоналізації – від холодного старту до розвиненого профілю. Тут також відображаються жанри, обрані під час онбордингу. Команда історія виводить останні рекомендації у хронологічному порядку з датами, категоріями та активними посиланнями.

Список підтримує пагінацію через кнопки «Назад» та «Далі», що дозволяє зручно переглядати архів. У разі потреби всю історію можна очистити за допомогою команди очистити історію, яка вимагає підтвердження для уникнення випадкових дій.

Оновлення профілю вподобань відбувається автоматично після кожних кількох оцінок, але користувач може прискорити цей процес командою оновити профіль. Це призводить до перерахунку вектора переваг на основі останніх взаємодій, що робить подальші рекомендації ще точнішими. Для отримання

загальної інформації про можливості бота призначена команда допомога, яка детально описує принцип роботи, приклади запитів та доступні команди. Команда про бота надає коротку довідку про версію та основні функції.

Як показано на рисунку 4.4, типовий сценарій взаємодії користувача з чат-ботом охоплює етапи від запуску до отримання персоналізованих рекомендацій з урахуванням зворотного зв’язку.

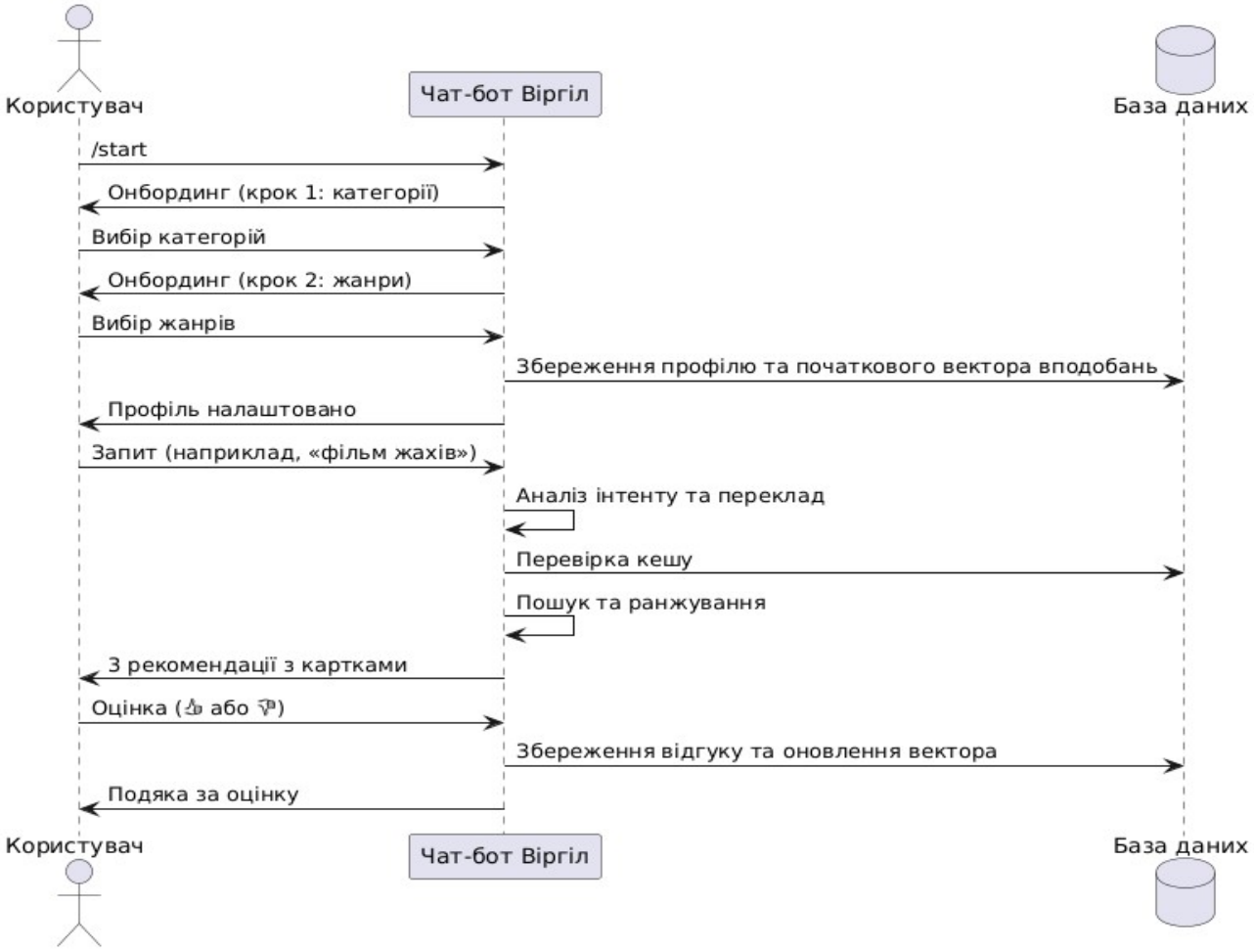

Рисунок 4.4 – Схема типового сценарію взаємодії користувача з чат-ботом

Система забезпечує високу стабільність завдяки вбудованим обмеженням на частоту повідомлень, що запобігає перевантаженню. Усі дані користувача зберігаються в захищеній базі, а кешування результатів пошуку прискорює повторні запити.

Персоналізація працює на двох рівнях: на етапі формування списку рекомендацій враховуються попередні вподобання, а після семантичного ранжування застосовується комбінування балів з урахуванням індивідуального вектора переваг. Це дозволяє боту з часом краще розуміти смаки користувача, наприклад, пропонувати більше драматичних фільмів тому, хто часто оцінює їх позитивно.

Для зручності регулярного використання в таблиці 4.1 наведено основні команди та їх призначення.

Таблиця 4.1 – Основні команди чат-бота

Команда	**Призначення**
/start	Запуск бота та онбординг для нових користувачів
/profile	Перегляд статистики профілю та рівня персоналізації
/history	Перегляд історії рекомендацій з пагінацією
/clear_history	Очищення історії з підтвердженням
/refresh_profile	Прискорене оновлення вектора вподобань
/help	Довідка з прикладами запитів
/about	Інформація про бота та версію

Ця таблиця демонструє, наскільки функціонал бота орієнтований на простоту та доступність для звичайного користувача без технічних навичок.

У процесі роботи чат-бот демонструє гнучкість у обробці різноманітних формулювань запитів. Користувач може вказувати жанр, настрій, тему, актора, автора чи навіть конкретну назву, а система самостійно адаптує пошук. Наприклад, запит «радянський фільм про собаку» призведе до рекомендацій класичних стрічок, що відповідають контексту.

Після отримання результатів користувач може одразу переходити до перегляду чи прослуховування за посиланнями, а також оцінювати контент для подальшого вдосконалення алгоритму. Такий підхід робить взаємодію природною та ефективною, перетворюючи звичайне спілкування в Телеграмі на персональний помічник для культурного дозвілля та пізнання.

Чат-бот також підтримує режим продовження розмови в межах однієї категорії, що дозволяє уточнювати запити без повторного вибору типу контенту. Це особливо зручно під час тривалих сесій, коли користувач шукає варіанти в одному напрямку. Усі рекомендації супроводжуються якісними обкладинками, які автоматично підбираються з надійних джерел, а описи адаптовані для зручного читання безпосередньо в месенджері.

Завдяки цьому процес отримання інформації стає максимально комфортним і не вимагає переходу до зовнішніх ресурсів для початкового ознайомлення.

Загалом, керівництво користувача підкреслює, що чат-бот Віргіл поєднує сучасні технології обробки природної мови з простим та інтуїтивним інтерфейсом. Кожен етап роботи – від першого запуску до регулярного використання – продуманий таким чином, щоб максимізувати користь і мінімізувати зусилля. Персоналізація, що розвивається з кожною оцінкою, забезпечує зростання якості рекомендацій з часом, роблячи бота справжнім супутником у світі культурного контенту.

Користувачі можуть бути впевненими, що система постійно вдосконалюється на основі їхніх реакцій, пропонуючи дедалі точніші та цікавіші варіанти для перегляду, читання чи прослуховування. Такий підхід не лише полегшує пошук потрібного матеріалу, але й сприяє відкриттю нових горизонтів у сфері пізнавального дозвілля.

4.2 Тестування функцій рекомендацій та діалогової взаємодії

Тестування функцій рекомендацій та діалогової взаємодії у чат-боті проводилося комплексно, аби переконатися в їхній надійності, точності та зручності для користувачів, які шукають корисний пізнавальний матеріал у форматі фільмів, книг чи музики. Основна увага приділялася перевірці того, як система обробляє запити, формує персоналізовані пропозиції та підтримує природний діалог, починаючи від першого знайомства з новим користувачем і закінчуючи повторними взаємодіями з урахуванням його вподобань.

Це включало як окремі модулі, що відповідають за класифікацію намірів, пошук даних, ранжування результатів та збереження історії, так і їх спільну роботу в єдиному ланцюгу обробки, де кожна помилка могла б порушити плавність спілкування. Завдяки ретельній перевірці вдалося підтвердити, що бот ефективно реагує на різноманітні запити, швидко адаптується до зворотного зв’язку від користувача та забезпечує стабільну продуктивність навіть при частих зверненнях.

Особливу роль у забезпеченні якості відіграло тестування компонентів, відповідальних за розуміння намірів користувача. У відповідних перевірках використовувався набір фіксованих прикладів фраз, які охоплювали різні сценарії спілкування – від простих привітань і прохань порадити щось конкретне до уточнень жанру, настрою чи бажання отримати додаткові варіанти. Класифікатор намірів, побудований на основі рекурентної нейронної мережі, проходив навчання на синтетичних даних, а потім оцінювався за точністю передбачень.

Під час цих перевірок моделі демонстрували високу здатність розрізняти системні команди, такі як прохання про довідку чи позитивну оцінку попередньої рекомендації, від безпосередніх запитів на контент. Це дозволило системі оперативно обробляти повідомлення без зайвих звернень до зовнішніх сервісів, що суттєво прискорювало діалог і робило його більш природним. Наприклад, фрази на кшталт «порадь фільм» чи «що почитати» надійно спрямовувалися до відповідних гілок обробки, а «дякую» чи «не подобається» активували механізми зворотного зв’язку, що відразу оновлювало профіль користувача.

Щоб продемонструвати конкретну перевірку роботи класифікатора намірів, розглянуто тестовий сценарій, наведений у таблиці 4.2.

Таблиця 4.2 – Тест-кейс ТК-INT-01: Перевірка класифікації наміру «порадь фільм»

№ кроку	Опис кроку	Вхідні дані	Очікувана поведінка системи	Результат тестування
1	Надсилання повідомлення користувачем	«порадь фільм»	Класифікатор визначає намір як запит на фільм	Намір розпізнано правильно
2	Перехід у стан очікування уточнення	–	Система переходить у стан waiting_clarification	Стан змінено коректно
3	Формування відповіді	–	Бот пропонує уточнити жанр або тему	Відповідь надіслано успішно
4	Перевірка логів	–	У логах фіксується intent_by_lstm з confidence > 0.8	Логи відповідають очікуваному

Цей тестовий сценарій підтвердив, що система правильно інтерпретує базові запити і плавно переводить діалог у потрібний стан.

Далі перевірялися функції, пов’язані з пошуком та збагаченням даних для рекомендацій. Тут увага зосереджувалася на клієнтах для різних джерел інформації, таких як спеціалізовані сервіси для фільмів, книг і музики, а також на допоміжних механізмах фільтрації результатів за сайтами.

Тести підтверджували, що система коректно формує запити з урахуванням категорії, уникає дублювання та успішно обробляє випадки, коли первинне джерело не дає результатів, переходячи до альтернативних варіантів. Наприклад, для фільмів перевірялася логіка розпізнавання конкретних назв у посиланнях і збагачення їх додатковими деталями, як рік випуску чи рейтинг.

Подібні перевірки проводилися для книг і музики, де акцент робився на точному витягуванні автора, опису та обкладинки. Це забезпечувало, що кожна рекомендація була не просто посиланням, а повноцінною карткою з корисною інформацією, яка полегшувала користувачеві рішення про перегляд чи читання.

Для ілюстрації перевірки основного ланцюга обробки запиту на рекомендацію розглянуто тестовий сценарій, представлений у таблиці 4.3.

Таблиця 4.3 – Тест-кейс ТК-REC-01: Обробка запиту на рекомендацію фільму «фільм про космос»

№ кроку	Опис кроку	Вхідні дані	Очікувана поведінка системи	Результат тестування
1	Надсилання запиту	«фільм про космос»	Виявлення категорії movie та переклад запиту	Категорія та переклад визначено
2	Пошук кандидатів	Запит до TMDBClient	Отримання 5–8 кандидатів з описами та постерами	Кандидати отримано (8 шт.)
3	Ранжування та персоналізація	Семантичний reranker + SessionGRU	Сортування за final_score з урахуванням preference_vector	Топ-3 сформовано правильно
4	Надсилання карток	–	Відправка 3 рекомендацій з обкладинками та кнопками оцінки	Картки надіслано без помилок

Цей тестовий сценарій продемонстрував стабільну роботу всього ланцюга від отримання повідомлення до відображення результатів, підтвердивши правильність інтеграції пошукових клієнтів і механізмів ранжування.

Важливим етапом стало тестування механізмів ранжування та персоналізації, які лежать в основі якості пропозицій. Тут перевірялися класи кандидатів на рекомендації, їх сортування за семантичною схожістю з запитом, а також комбінування балів з урахуванням вектора вподобань користувача, сформованого на основі попередніх оцінок.

Тести показували, що за відсутності персональних даних система покладається виключно на семантичну релевантність, а при наявності історії – ефективно зсуває результати в бік того, що раніше подобалося. Перевірка включала сценарії з порожніми списками, дублюваннями та різними рівнями зворотного

зв’язку, аби переконатися, що фінальний порядок завжди логічний і корисний. Крім того, окремо тестувався енкодер сесій, який перетворює послідовність взаємодій у компактний вектор, придатний для швидкого порівняння. Це дозволило системі пам’ятати вподобання користувача навіть після перерв у спілкуванні, роблячи діалог більш осмисленим і персональним з кожним новим запитом.

Для ілюстрації ефективності перевірки класифікатора намірів можна звернутися до таблиці 4.4, де зібрано ключові приклади тестових фраз та очікувані результати.

Таблиця 4.4 – Приклади тестових фраз для перевірки класифікатора намірів запиту

Запит	**Очікуваний намір**
порадь фільм	запит на фільм
хочу подивитися кіно	запит на фільм
що почитати	запит на книгу
порадь книгу про кохання	уточнення теми
щось веселе	уточнення настрою
детектив	уточнення жанру
ще варіанти	бажання додаткових варіантів
дякую	позитивна оцінка
не подобається	негативна оцінка
привіт	привітання
як це працює	прохання довідки

Ця таблиця демонструє, як тести охоплювали весь спектр можливих взаємодій, забезпечуючи, щоб бот не плутав побутові фрази з технічними командами і завжди обирав правильний шлях обробки.

Для наочності процесу перевірки основних функцій рекомендацій можна звернутися до схеми на рисунку 4.5, яка відображає послідовність тестування ключових етапів обробки запиту.

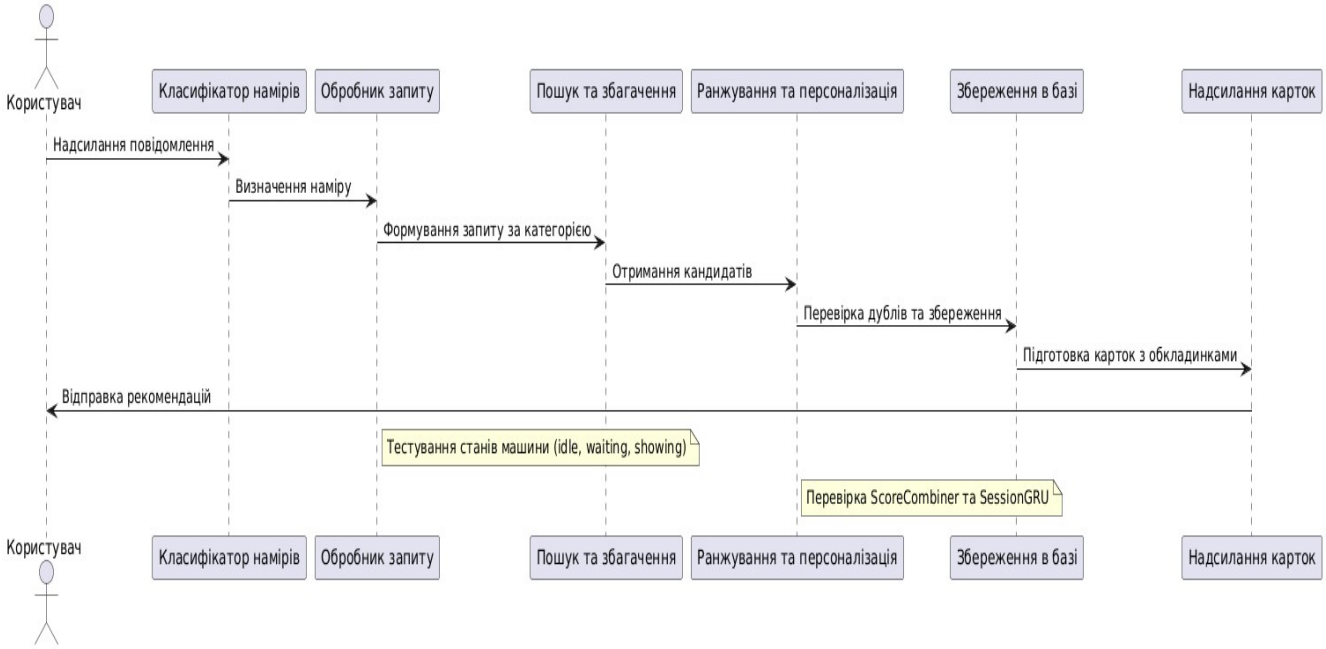

Рисунок 4.5 – Схема тестування ланцюга рекомендацій та діалогової взаємодії

Як видно зі схеми, тестування охоплювало весь шлях від отримання повідомлення до відправки результатів, включаючи проміжні перевірки стану машини діалогу та оновлення профілю користувача.

Окремо проводилося тестування роботи з базою даних, де перевірялися репозиторії для користувачів, історії рекомендацій, оцінок, стану сесії та кешів. Кожна операція – створення запису, оновлення, пошук за ідентифікатором чи очищення прострочених даних – проходила через спеціальні сценарії з використанням тестової бази в пам’яті.

Це дозволяло переконатися, що система правильно зберігає історію переглядів, оновлює вектор вподобань після кожних кількох оцінок і ефективно використовує кеш для прискорення повторних запитів. Наприклад, тести на кеш пошуку підтверджували, що при однакових запитах результат повертається миттєво без повторного звернення до зовнішніх джерел, а персональний кеш враховує індивідуальні уподобання.

Подібні перевірки для історії забезпечували коректну пагінацію та можливість повного очищення даних за запитом користувача.

Тестування діалогової взаємодії приділяло значну увагу обробці різних сценаріїв спілкування, зокрема онбордингу нових користувачів. Тут перевірялася логіка послідовних кроків вибору категорій і жанрів, збереження початкового вектора вподобань та перехід до основного режиму рекомендацій. Для детальної ілюстрації перевірки процесу першого знайомства розглянуто тестовий сценарій, наведений у таблиці 4.5.

Таблиця 4.5 – Тест-кейс ТК-ONB-01: Перевірка онбордингу нового користувача

№ кроку	Опис кроку	Вхідні дані	Очікувана поведінка системи	Результат тестування
1	Запуск онбордингу при /start	Новий користувач без onboarding_completed	Відправка клавіатури вибору категорій	Клавіатура відображена
2	Вибір категорій	Обрання «movie» та «book»	Збереження вибору та перехід до вибору жанрів	Дані збережено, перехід виконано
3	Вибір жанрів та завершення	Обрання comedy, scifi	Збереження preferences_json та побудова початкового preference_vector	Профіль налаштовано, онбординг завершено
4	Перевірка стану	–	Перехід у RecommendationStates.idle	Стан змінено правильно

Цей тестовий сценарій показав, що процес першого налаштування профілю проходить безперебійно і формує основу для подальшої персоналізації.

Перевірка витягувача змісту сторінок підтверджувала, що система здатна отримувати корисну інформацію з різних доменів, використовуючи спеціальні селектори для популярних ресурсів і резервні варіанти на основі мета-тегів чи перших абзаців. Це було важливо для формування повноцінних описів рекомендацій, які допомагають користувачеві скласти уявлення про матеріал ще до переходу за посиланням. Аналогічно тести на клієнти пошуку показували стабільну

роботу з урахуванням регіону, таймаутів і фільтрів, що запобігало зависанню діалогу при тимчасових проблемах з мережею.

Обробники зворотного зв’язку, такі як реакції на кнопки «подобається» чи «не подобається», тестувалися на коректне оновлення бази та можливе перерахування профілю при досягненні певної кількості взаємодій. Для ілюстрації перевірки механізму зворотного зв’язку розглянуто тестовий сценарій, представлений у таблиці 4.6.

Таблиця 4.6 – Тест-кейс ТК-FB-01: Обробка позитивної оцінки рекомендації

№ кроку	Опис кроку	Вхідні дані	Очікувана поведінка системи	Результат тестування
1	Натиснення кнопки «Подобається»	callback_data = "fb:1:123"	Додавання запису в FeedbackRepository з rating=1	Запис створено
2	Оновлення лічильника взаємодій	–	Збільшення interactions_since_update	Лічильник оновлено
3	Перерахунок вектора (якщо потрібно)	interactions_since_update >= 3	Виклик _rebuild_preference_vector та збереження в SessionStateRepository	Preference_vector оновлено
4	Відповідь користувачеві	–	Відправка повідомлення «Дякую за оцінку!»	Повідомлення надіслано

Цей тестовий сценарій підтвердив, що система оперативно реагує на оцінки та поступово покращує персоналізацію.

У цілому результати тестування свідчать про високу готовність функцій рекомендацій та діалогової взаємодії до реального застосування. Система не лише правильно обробляє запити та формує корисні пропозиції, але й вчиться на реакціях користувача, роблячи кожну наступну взаємодію більш точною та приємною. Це створює відчуття живого помічника, який розуміє потреби людини і допомагає відкривати новий пізнавальний матеріал, адаптуючись до її смаків з часом.

Такий підхід до тестування дозволив виявити та усунути потенційні вузькі місця ще на етапі розробки, забезпечивши стабільність роботи чат-бота в різних умовах. Завдяки цьому користувачі отримують надійний інструмент для щоденного користування, де діалог тече природно, а рекомендації завжди актуальні та персоналізовані.

Висновки до розділу 4

Розроблений чат-бот Віргіл забезпечує ефективну програмну реалізацію персоналізованих рекомендацій пізнавального контенту в форматі фільмів, книг та музики, повністю адаптовану до потреб українського користувача. Інтуїтивний інтерфейс месенджера Телеграм у поєднанні з механізмами обробки природної мови дозволяє здійснювати природне спілкування без додаткових технічних навичок. Процес початкового налаштування профілю через послідовний вибір категорій контенту та жанрів формує базовий вектор вподобань, який надалі постійно уточнюється на основі семантичного аналізу запитів та реакцій користувача.

Система оперативно розпізнає наміри повідомлень, формує релевантні пропозиції у вигляді структурованих карток із візуальними елементами, описами та посиланнями, а також підтримує контекстні сесії для зв’язного діалогу. Механізм зворотного зв’язку через інтерактивні елементи забезпечує негайне оновлення персонального профілю, що сприяє поступовому підвищенню точності рекомендацій.

Доступні функціональні команди дозволяють користувачам контролювати історію взаємодій, переглядати статистику персоналізації та виконувати оновлення налаштувань, роблячи систему зручною та самодостатньою для регулярного застосування.

Комплексне тестування функціональних модулів рекомендацій та діалогової взаємодії підтвердило їхню стабільність, точність і адаптивність до реальних умов експлуатації. Перевірка класифікатора намірів на базі рекурентної нейронної

мережі продемонструвала високу надійність розпізнавання різноманітних типів запитів, включаючи прямі прохання контенту, уточнення жанру чи настрою, а також системні команди.

Тестування повного ланцюга обробки – від виявлення категорії та пошуку кандидатів до семантичного ранжування з урахуванням вектора вподобань – засвідчило коректну роботу інтегрованих клієнтів зовнішніх джерел, механізмів персоналізації та енкодера сесій. Окрема увага була приділена перевірці процесів онбордингу, обробки зворотного зв’язку, збереження даних у базі та кешування результатів, що забезпечило відсутність критичних помилок, швидке відновлення стану та ефективне оновлення профілів користувачів навіть за умови частого навантаження.

Результати численних тестових сценаріїв підтвердили, що система зберігає високу продуктивність, плавність діалогу та точність рекомендацій незалежно від рівня сформованості профілю.

Отримані результати реалізації та тестування свідчать про повну готовність чат-бота до практичного впровадження. Система успішно поєднує сучасні технології семантичного аналізу, машинного навчання та персоналізації, створюючи зручний інструмент для культурного дозвілля, який постійно вдосконалюється на основі індивідуальних реакцій користувача.

Такий підхід забезпечує не лише оперативне задоволення інформаційних потреб, а й стимулює відкриття нового пізнавального матеріалу, роблячи взаємодію ефективною та природною.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну проблему ефективного доступу до пізнавального контенту в умовах інформаційного перевантаження. Розроблено та реалізовано інтелектуального Telegram-бота Virgil, який забезпечує персоналізовані рекомендації фільмів, книг і музики українською мовою на основі природних запитів користувача. Система поєднує сучасні методи обробки природної мови, семантичний аналіз, рекурентні нейронні мережі та гібридну персоналізацію, що повністю відповідає поставленій меті та завданням кваліфікаційної роботи.

У результаті аналізу предметної сфери та аналогів визначено ключові недоліки існуючих рішень – обмежену контекстну адаптивність, слабку підтримку української мови та недостатню персоналізацію. Розроблений чат-бот долає ці обмеження завдяки гібридній архітектурі: класифікатору інтенту на базі LSTM, оркестратору пошуку з використанням великої мовної моделі Groq, семантичному переранжуванню за допомогою SentenceTransformer (модель ukr-paraphrase-multilingual-mpnet-base) та динамічному вектору вподобань, сформованому GRU-енкодером сесії. Реалізовані механізми онбордингу, зворотного зв’язку, кешування та дедуплікації забезпечують високу релевантність рекомендацій уже з першого запиту та їх постійне вдосконалення.

Структура системи, побудована на Python з використанням aiogram 3, SQLAlchemy та асинхронних репозиторіїв, виявилася ефективною та масштабовною. Демонстрація роботи бота та комплексне тестування підтвердили стабільність діалогової взаємодії, точність розпізнавання намірів, швидкість відповідей та зростання якості рекомендацій у міру накопичення історії користувача. Отримані результати повністю корелюють із завданнями, сформульованими у вступі, та перевищують вимоги щодо функціональності, зручності та персоналізації.

Розроблений чат-бот Virgil може бути впроваджений у практику самоосвіти, бібліотечні та освітні сервіси, а також інтегрований у корпоративні платформи дистанційного навчання. Для експлуатації достатньо стандартного серверного середовища; система не потребує значних обчислювальних ресурсів і легко масштабується. Подальше вдосконалення може включати розширення категорій контенту та глибшу культурну адаптацію, що відкриває перспективи для широкого практичного застосування в умовах цифрової трансформації освіти та самоосвіти.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Задоріна О. М., Громик А. П., Бондар С. А. Використання чат-ботів зі штучним інтелектом для покращення комунікації та підтримки здобувачів освіти в мобільних застосунках для дистанційного навчання. Педагогічна Академія: наукові записки. 2025. № 17.

2. Pinho P. C. R. et al. Chatbots in Educational Recommender Systems: A Systematic Literature Review. 2023 IEEE Frontiers in Education Conference (FIE). 2023. DOI: 10.1109/FIE58773.2023.10343248. URL: https://ieeexplore.ieee.org/document/10343248/ (дата звернення: 30.04.2026).

3. Debets T. et al. Chatbots in education: A systematic review of objectives, underlying technology and theory, evaluation criteria, and impacts. Computers & Education. 2025. Vol. 234. URL: https://www.sciencedirect.com/science/article/pii/S0360131525000910 (дата звернення: 30.04.2026).

4. Як створити чат-бот для сфери освіти. SendPulse: вебсайт. URL: https://sendpulse.ua/blog/chatbot-for-education-sphere (дата звернення: 30.04.2026).

5. Valtolina S. Design of a conversational recommender system in education. User Modeling and User-Adapted Interaction. 2024. URL: https://link.springer.com/article/10.1007/s11257-024-09397-y (дата звернення: 30.04.2026).

6. Новий чатбот “Спільно для вчителів” – особистий помічник для освітян. UNICEF Ukraine. 2022. URL: https://www.unicef.org/ukraine/press-releases/spilno-bot-for-teachers (дата звернення: 30.04.2026).

7. Wong A. K. L. et al. Integrating chatbots with learning management systems for personalized learning: a comprehensive review and framework proposal. Discover Education. 2025. URL: https://link.springer.com/article/10.1007/s44217-025-00958-w (дата звернення: 30.04.2026).

8. Okonkwo C. W., Ade-Ibijola A. Chatbots applications in education: A systematic review. Computers and Education: Artificial Intelligence. 2021. Vol. 2. 100033. URL: https://www.sciencedirect.com/science/article/pii/S2666920X21000278 (дата звернення: 30.04.2026).

9. Kuhail M. A., Alturki N., Alramlawi S., Alhejaili K. Interacting with educational chatbots: A systematic review. Education and Information Technologies. 2023. URL: https://link.springer.com/article/10.1007/s10639-022-11177-3 (дата звернення: 30.04.2026).

10. Міністерство освіти і науки України. Освітній чат-бот. 2024. URL: https://mon.gov.ua/osvita-2/tsifrova-transformatsiya-osviti-i-nauki/osvitniy-chat-bot (дата звернення: 30.04.2026).

11. Elkot M. A. Pedagogical Influence of AI-Chatbots on Learning Outcomes: A Systematic Review. International Journal of Educational Management. 2025. URL: https://www.ijem.com/pedagogical-influence-of-ai-chatbots-on-learning-outcomes-a-systematic-review (дата звернення: 30.04.2026).

12. Чат-боти в навчальних закладах: Майбутнє освіти. Gerabot. 2023. URL: https://gerabot.com/article/chatboti_v_navchalnih_zakladah_maibutn_osviti (дата звернення: 30.04.2026).

13. Чат-боти для навчання: огляд найпопулярніших та особливості використання. TeachHub. 2023. URL: http://teach-hub.com/chat-boty-dlia-navchannia-ohliad-naypopuliarnishykh-ta-osoblyvosti-vykorystannia/ (дата звернення: 30.04.2026).

14. 20 найкращих чат-ботів на основі ШІ для роботи, навчання й розваг. Depositphotos Blog. 2024. URL: https://blog.depositphotos.com/ua/najkrashhi-chat-boti-na-osnovi-shi.html (дата звернення: 30.04.2026).

15. Терлецька І. І., Коваленко В. О. Використання чат-ботів на основі великих мовних моделей у науково-педагогічній діяльності. 2024. URL: https://elibrary.kubg.edu.ua/48830/1/T_Terletska_I_Kovalenko_VOESSU_16_NDLCO.pdf (дата звернення: 30.04.2026).

16. Lin M. P. C. CHAT-ACTS: A pedagogical framework for personalized chatbot to enhance active learning and self-regulated learning. Computers and Education: Artificial Intelligence. 2023. URL: https://www.sciencedirect.com/science/article/pii/S2666920X23000462 (дата звернення: 30.04.2026).

17. Bimpong B. W. The Impact of Generative AI Educational Chatbots on the Academic Support Experiences of Students in U.S. Research Universities. 2025. URL: https://www.naspa.org/blog/the-impact-of-generative-ai-educational-chatbots-on-the-academic-support-experiences-of-students-in-u-s-research-universities (дата звернення: 30.04.2026).

18. GURU99. 10 НАЙКРАЩИХ альтернатив ChatGPT (2026). 2026. URL: https://www.guru99.com/uk/chatgpt-alternatives.html (дата звернення: 30.04.2026).

19. Орлов О. Методика використання чат-ботів у керуванні дослідницькою діяльністю студентів. Науковий вісник Вінницької академії безперервної освіти. Серія «Педагогіка Психологія». 2025. URL: https://www.researchgate.net/publication/397196565_METODIKA_VIKORISTANNA_CAT-BOTIV_U_KERUVANNI_DOSLIDNICKOU_DIALNISTU_STUDENTIV (дата звернення: 30.04.2026).

20. Чат-боти в навчальних закладах: Майбутнє освіти. GERABOT. URL: https://gerabot.com/article/chatboti_v_navchalnih_zakladah_maibutn_osviti (дата звернення: 30.04.2026).

21. Segovia-García S. et al. Exploring the pedagogical uses of AI chatbots. 2024. URL: https://teachingcommons.stanford.edu/teaching-guides/artificial-intelligence-teaching-guide/exploring-pedagogical-uses-ai-chatbots (дата звернення: 30.04.2026).

22. Каліндрузь Б. Модернізація систем управління навчанням за допомогою штучного інтелекту: шлях до інноваційної освітньої екосистеми. Педагогіка. 2025. URL: https://pedagogy.com.ua/index.php/journal/article/download/21/22 (дата звернення: 30.04.2026).

23. Аксак Н., Кушнарьов М., Татарников А. Агентно-орієнтована платформа навчання. Матеріали XI Міжнародної науково-практичної конференції «Інформаційні управляючі системи і технології». 2023. URL: https://icst-conf.com/2024.pdf (дата звернення: 30.04.2026).

24. Голуб С., Химиця Н. Методика використання багаторівневої моніторингової інформаційної системи у кліометричних дослідженнях. Proceedings of Information, Communication, Society 2024. 2024. URL: https://skid.lpnu.ua/wp-content/uploads/2024/05/ICS2024_Proceedings.pdf (дата звернення: 30.04.2026).

25. Сокіл М., Андрухів А. Методи та алгоритми штучного інтелекту для формування адаптивного навчального контенту. Proceedings of Information, Communication, Society 2024. 2024. URL: https://skid.lpnu.ua/wp-content/uploads/2024/05/ICS2024_Proceedings.pdf (дата звернення: 30.04.2026).

26. Sprenkamp K. et al. Data-driven intelligence in crisis: The case of Ukrainian refugees. Computers in Human Behavior Reports. 2025. URL: https://www.sciencedirect.com/science/article/pii/S0740624X24000704 (дата звернення: 30.04.2026).

27. Mrafoo. I Built an AI-Powered Interactive Textbook for Robotics: Full-Stack RAG with Sentence Transformers and Groq. dev.to. 2026. URL: https://dev.to/mrafoo/i-built-an-ai-powered-interactive-textbook-for-robotics-full-stack-rag-1mj (дата звернення: 30.04.2026).

28. lang-uk. ukr-paraphrase-multilingual-mpnet-base: Sentence Transformer model fine-tuned for Ukrainian. Hugging Face. 2024. URL: https://huggingface.co/lang-uk/ukr-paraphrase-multilingual-mpnet-base (дата звернення: 30.04.2026).

29. Кушнерьов О. С., Вадько К. С., Кузченко А. О. Чат-бот на основі нейромережі для надання інформаційної підтримки бізнесу з позиції кібербезпеки. Сучасний захист інформації. 2024. № 4(60). С. 131–140.

30. Окландер М. А. Використання штучного інтелекту у персоналізації комунікацій на сайті. Економіка та суспільство. 2025. Вип. 82.

31. Shumpei O., Yukihiro T., Shingo O., Akira T. Embedding-based news recommendation for millions of users. The 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2017. P. 1933–1942.

32. Ільїна І., Реука К. Інтегрована система штучного інтелекту в телеграмбот для вивчення англійської мови. Проблеми інформатизації: тези доп. одинадцятої міжнар. наук.-техн. конф. Баку – Харків – Бельсько-Бяла, 16–17 листоп. 2023 р. С. 14. DOI: doi.org/10.32620/PI.23.t1.

33. Як інтегрувати чат-ботів у сторонні сервіси. SMSCLUB. 2024. URL: https://smsclub.mobi/chat-bot-integration/ (дата звернення: 30.04.2026).

34. Механізм зворотного зв'язку. Emergency wash. URL: https://www.emergency-wash.org/hygiene/uk/instrumenty-ta-metody/173-feedback-mechanism-1 (дата звернення: 30.04.2026).

35. Котирло В. В., Жебка В. В. Розробка чат-бота на основі глибокого навчання з підкріпленням та симуляції користувача. V Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. Київ: ДУІКТ, 2025.

ДОДАТОК А

Основні лістинги проєкту

recommendation.py

```
import asyncio
from typing import Any
from aiogram import Router
from aiogram.fsm.context import FSMContext
from aiogram.types import Message
from loguru import logger
from bot.keyboards.inline import build_card_keyboard
from bot.keyboards.reply import CATEGORY_KEYBOARD, CATEGORY_MAP
from bot.states.recommendation import RecommendationStates
from config.settings import settings
from content_sources.covers import CoverFetcher
from content_sources.normalizer import RecommendationCard
from database.engine import AsyncSessionFactory
from database.repositories.cache import SearchCacheRepository
from database.repositories.history import HistoryRepository
from database.repositories.session_state import SessionStateRepository
from database.repositories.user import UserRepository
from nlu.entities.dicts import extract_entities
from nlu.preprocessing import preprocess
from ranking.reranker import Candidate
from ranking.score_combiner import ScoreCombiner, filter_duplicates
from utils.logger import log_metric
router = Router()
# ---------------------------------------------------------------------------
# Допоміжні функції
# ---------------------------------------------------------------------------
# Явні ключові слова категорій в оригінальному тексті перекривають семантичний
класифікатор
_EXPLICIT_CATEGORY: dict[str, frozenset[str]] = {
    "movie": frozenset({
        "фільм", "кіно", "серіал", "кінофільм", "переглянути", "дивитися",
        "movie", "film", "cinema", "series", "show", "tv",
    }),
    "book": frozenset({
        "книга", "книжка", "книгу", "роман", "читати", "автор", "письменник",
        "book", "novel", "read", "author", "literature",
    }),
    "music": frozenset({
        "музика", "пісня", "альбом", "виконавець", "слухати", "трек",
        "music", "song", "album", "artist", "listen",
    }),
}
def _detect_explicit_category(text: str) -> str | None:
    words = set(text.lower().split())
    for category, keywords in _EXPLICIT_CATEGORY.items():
        if words & keywords:
            return category
    return None
_FRAMING_COMMON = frozenset({
    "recommend", "me", "a", "an", "the", "something", "want", "i",
    "some", "good", "best", "great", "what", "should", "help", "suggest",
    "show", "give", "find", "please", "looking", "for", "need",
    "interested", "in", "similar", "to", "type", "of", "kind", "about",
    "with", "that", "its", "has", "have", "like", "not",
    "but", "and", "also", "however", "plus", "more", "also", "now", "maybe",
})
_FRAMING_BY_CATEGORY = {
    "movie": frozenset({"film", "movie", "cinema", "series", "show", "watch", "see",
"episode"}),
    "book": frozenset({"book", "novel", "story", "read", "literature"}),
```

```
    "music": frozenset({"music", "song", "songs", "album", "albums", "listen", "hear",
"play"}),
}
def _extract_topic(text: str, category: str) -> str:
    """Видаляє з перекладеного запиту загальні слова-заповнювачі, залишаючи тему."""
    stop = _FRAMING_COMMON | _FRAMING_BY_CATEGORY.get(category, frozenset())
    words = [w for w in text.lower().split() if w not in stop and len(w) > 1]
    return " ".join(words) if words else text
_MISTRANSLATIONS = {
    "antiutopian": "dystopian",
    "antiutopia":  "dystopia",
    "antitutopia": "dystopia",
    "phantastic":  "fantasy",
    "fantastical": "fantasy",
    "romantical":  "romance",
    "psychologic": "psychological",
    "detection":   "detective",
    "detectiv":    "detective",
    "criminalistic": "crime",
    "horrors":     "horror",
    "adventures":  "adventure",
}
def _normalize_query(text: str) -> str:
    """Виправляє типові артефакти машинного перекладу uk→en."""
    return " ".join(_MISTRANSLATIONS.get(w.lower(), w) for w in text.split())
_CATEGORY_PROMPTS = {
    "movie": "Який фільм шукаєш? Вкажи жанр, настрій або тему.",
    "book": "Яку книгу шукаєш? Вкажи жанр, тему або автора.",
    "music": "Яку музику шукаєш? Вкажи жанр, настрій або виконавця.",
}
# ---------------------------------------------------------------------------
# Джерела даних
# ---------------------------------------------------------------------------
import re as _re
from urllib.parse import urlparse as _urlparse
_KNOWN_MOVIE_SITES = frozenset({
    "imdb.com", "letterboxd.com", "themoviedb.org",
    "rottentomatoes.com", "uk.wikipedia.org", "en.wikipedia.org",
})
def _parse_movie_title(ddg_title: str, url: str) -> str | None:
    """
    Повертає назву фільму якщо URL вказує на сторінку конкретного фільму,
    або None якщо це сторінка-список / пошук.
    """
    parsed = _urlparse(url)
    path = parsed.path
    netloc = parsed.netloc
    if not any(s in netloc for s in _KNOWN_MOVIE_SITES):
        return None
    if "imdb.com" in netloc and not _re.match(r"^/title/tt\d+/?$", path):
        return None
    if "letterboxd.com" in netloc and not _re.match(r"^/film/[^/]+/?$", path):
        return None
    if "themoviedb.org" in netloc and not _re.match(r"^/movie/\d+", path):
        return None
    if "rottentomatoes.com" in netloc and not _re.match(r"^/m/[^/]+/?$", path):
        return None
    title = _re.sub(r"\s*\(\d{4}\)\s*$", "", ddg_title)
    for suffix in (
        " - IMDb", " | Letterboxd", "- Letterboxd",
        " - Rotten Tomatoes", " | Rotten Tomatoes",
        " | The Movie Database (TMDb)", " - Вікіпедія",
    ):
        if title.endswith(suffix):
            title = title[: -len(suffix)]
    title = title.strip()
    if not title:
        return None
    # Якщо після очистки назва містить залишки сайтів або надто довга — це сміття
    _SITE_MARKERS = ("IMDb", "Rotten Tomatoes", "Letterboxd", "TMDb", "Wikipedia")
```

```
    if any(m in title for m in _SITE_MARKERS) or len(title) > 80:
        return None
    return title
async def _enrich_one_with_tmdb(tmdb, title: str, fallback_url: str, fallback_snippet:
str) -> Candidate:
    """Шукає фільм або серіал у TMDB за назвою; fallback — DDG сніпет."""
    try:
        items = await tmdb.search_multi(title, max_results=1)
    except Exception:
        items = []
    if items:
        m = items[0]
        return Candidate(
            title=m.title,
            url=m.url,
            snippet=m.overview[:200] if m.overview else fallback_snippet,
            full_text=m.overview or fallback_snippet,
            cover_url=m.poster_url,
            metadata={k: v for k, v in {
                "year": m.release_year,
                "rating": f"{m.vote_average:.1f}/10" if m.vote_average else None,
            }.items() if v is not None},
        )
    return Candidate(
        title=title,
        url=fallback_url,
        snippet=fallback_snippet,
        full_text=fallback_snippet,
    )
async def _fetch_movies_ddg(search_query: str) -> list[Candidate]:
    """
    DDG знаходить фільми по контенту (тема, епоха, актор),
    TMDB збагачує постером, рейтингом і повним описом.
    """
    from search.ddg_client import DDGSearchClient
    from search.tmdb_client import TMDBClient
    ddg = DDGSearchClient()
    try:
        ddg_results = await ddg.search(search_query, category="movie")
    except Exception as e:
        logger.warning(f"DDG movie search error: {e}")
        return []
    tmdb = TMDBClient()
    tasks = []
    seen: set[str] = set()
    for r in ddg_results:
        title = _parse_movie_title(r.title, r.url)
        if not title:
            continue
        key = title.lower()
        if key in seen:
            continue
        seen.add(key)
        tasks.append(_enrich_one_with_tmdb(tmdb, title, r.url, r.snippet))
    if not tasks:
        return []
    results = await asyncio.gather(*tasks, return_exceptions=True)
    return [c for c in results if isinstance(c, Candidate)]
def _movie_title_only(title: str) -> str:
    """Видаляє 'by Director' суфікс якщо Groq повернув такий формат для фільму."""
    if " by " in title:
        return title.split(" by ")[0].strip()
    return title
async def _enrich_one_movie(tmdb, title: str) -> "Candidate | None":
    """Шукає фільм у TMDB: спочатку /search/movie, fallback — /search/multi з фільтром по
movies."""
    clean = _movie_title_only(title)
    # /search/movie — тільки фільми, без серіалів
    movies = await tmdb.search_movies(clean, max_results=3)
    if movies:
```

```
        return _tmdb_item_to_candidate(movies[0])
    # Fallback: multi-search, беремо перший результат з media_type=movie
    items = await tmdb.search_multi(clean, max_results=5)
    for item in items:
        if getattr(item, "media_type", "movie") == "movie":
            return _tmdb_item_to_candidate(item)
    return None
async def _enrich_movies(titles: list[str]) -> list[Candidate]:
    """TMDB lookup для кожної назви з Groq-списку."""
    from search.tmdb_client import TMDBClient
    tmdb = TMDBClient()
    tasks = [_enrich_one_movie(tmdb, t) for t in titles]
    results = await asyncio.gather(*tasks, return_exceptions=True)
    candidates: list[Candidate] = []
    seen_urls: set[str] = set()
    for item in results:
        if isinstance(item, Candidate) and item.url not in seen_urls:
            seen_urls.add(item.url)
            item.is_direct_match = True
            candidates.append(item)
    return candidates
async def _enrich_music(titles: list[str]) -> list[Candidate]:
    """Last.fm lookup для кожного «Альбом by Артист» з Groq-списку."""
    from search.lastfm_client import LastFMClient
    lastfm = LastFMClient()
    tasks = [lastfm.search_albums(t, limit=1) for t in titles]
    results = await asyncio.gather(*tasks, return_exceptions=True)
    candidates: list[Candidate] = []
    seen_urls: set[str] = set()
    for items in results:
        if isinstance(items, list) and items:
            a = items[0]
            if a.url not in seen_urls:
                seen_urls.add(a.url)
                c = _lastfm_item_to_candidate(a)
                c.is_direct_match = True
                candidates.append(c)
    return candidates
async def _enrich_books(titles: list[str]) -> list[Candidate]:
    """Google Books lookup для кожного «Назва by Автор» з Groq-списку. Fallback — Open
Library."""
    from search.googlebooks_client import GoogleBooksClient
    from search.openlibrary_client import OpenLibraryClient
    gb = GoogleBooksClient(api_key=settings.googlebooks_api_key)
    candidates: list[Candidate] = []
    seen_urls: set[str] = set()
    gb_failed = 0
    for title_str in titles:
        if " by " in title_str:
            parts = title_str.rsplit(" by ", 1)
            book_title = parts[0].strip()
            author = parts[1].strip()
            query = f'intitle:"{book_title}" inauthor:"{author}"'
        else:
            book_title = title_str
            author = ""
            query = f'intitle:"{title_str}"'
        results = await gb.search_books(query, max_results=1)
        if results:
            r = results[0]
            if r.url not in seen_urls:
                seen_urls.add(r.url)
                c = _book_result_to_candidate(r)
                c.is_direct_match = True
                candidates.append(c)
        else:
            gb_failed += 1
            # Google Books не знайшов — пробуємо Open Library
            ol = OpenLibraryClient()
```

```
            ol_query = f'title:"{book_title}"' + (f' author:"{author}"' if author else
"")
            ol_results = await ol.search_books(ol_query, max_results=1)
            for r in ol_results[:1]:
                if r.url not in seen_urls:
                    seen_urls.add(r.url)
                    c = _book_result_to_candidate(r, is_ol=True)
                    c.is_direct_match = True
                    candidates.append(c)
    if gb_failed:
        logger.warning(f"Google Books не знайшов {gb_failed}/{len(titles)} книг,
використано Open Library")
    return candidates
async def _fetch_from_source(
    search_query: str,
    category: str,
    entities: dict,
    user_context: dict | None = None,
) -> list[Candidate]:
    from bot.dispatcher import orchestrator as _orchestrator
    if _orchestrator is None:
        return []
    plan = await _orchestrator.plan(search_query, category, user_context=user_context)
    titles = plan.get("titles", [])
    if not titles:
        logger.warning("Orchestrator повернув порожній список titles — fallback TMDB")
        from search.tmdb_client import TMDBClient
        items = await TMDBClient().search_multi(plan.get("translation", search_query),
max_results=8)
        return [_tmdb_item_to_candidate(m) for m in items]
    if category == "movie":
        return await _enrich_movies(titles)
    if category == "music":
        return await _enrich_music(titles)
    return await _enrich_books(titles)
def _tmdb_item_to_candidate(m) -> "Candidate":
    return Candidate(
        title=m.title,
        url=m.url,
        snippet=m.overview[:200] if m.overview else "",
        full_text=m.overview,
        cover_url=m.poster_url,
        metadata={k: v for k, v in {
            "year": m.release_year,
            "rating": f"{m.vote_average:.1f}/10" if m.vote_average else None,
        }.items() if v is not None},
    )
async def _execute_movie_plans(plans: list[dict], fallback_query: str) ->
list[Candidate]:
    from search.tmdb_client import TMDBClient
    tmdb = TMDBClient()
    all_candidates: list[Candidate] = []
    seen_urls: set[str] = set()
    person_plan = next((p for p in plans if p["strategy"] == "person"), None)
    author_plan = next((p for p in plans if p["strategy"] == "author"), None)
    topic_plan  = next((p for p in plans if p["strategy"] == "topic"), None)
    title_plan  = next((p for p in plans if p["strategy"] == "title"), None)
    if person_plan and topic_plan:
        # Фільмографія персони через TMDB (15 робіт з головними ролями order<20),
        # далі reranker фільтрує по темі (topic_plan["query"]).
        items = await tmdb.search_multi(person_plan["query"], max_results=15)
        for m in items:
            if m.url not in seen_urls:
                seen_urls.add(m.url)
                all_candidates.append(_tmdb_item_to_candidate(m))
    elif author_plan:
        # Літературний автор у контексті фільму → екранізації через DDG
        topic_part = topic_plan["query"] if topic_plan else "film adaptation"
        ddg_query = f"{author_plan['query']} {topic_part}"
        ddg = await _fetch_movies_ddg(ddg_query)
```

```
        for c in ddg:
            if c.url not in seen_urls:
                seen_urls.add(c.url)
                all_candidates.append(c)
    elif person_plan:
        items = await tmdb.search_multi(person_plan["query"],
max_results=settings.search_results_per_query)
        for m in items:
            if m.url not in seen_urls:
                seen_urls.add(m.url)
                c = _tmdb_item_to_candidate(m)
                c.is_direct_match = True
                all_candidates.append(c)
    elif topic_plan:
        topic_query = topic_plan["query"]
        topic_words_lower = [w.strip("-.").lower() for w in topic_query.split()]
        # Крок 1: визначаємо жанр з topic query через TMDB genre API (без хардкоду)
        genres_map = await tmdb.get_genres_map()
        genre_id: int | None = None
        for word in topic_words_lower:
            if word in genres_map:
                genre_id = genres_map[word]
                logger.debug(f"TMDB genre detected: {word} -> id={genre_id}")
                break
        # Перевіряємо також двословні жанри ("science fiction")
        if genre_id is None:
            bigrams = [f"{topic_words_lower[i]} {topic_words_lower[i+1]}"
                       for i in range(len(topic_words_lower) - 1)]
            for bg in bigrams:
                if bg in genres_map:
                    genre_id = genres_map[bg]
                    break
        # Крок 2: паралельний пошук TMDB keyword IDs для слів довших 4 символів
        kw_words = [w for w in topic_words_lower if len(w) > 4][:5]
        kw_tasks = [tmdb.search_keywords(w) for w in kw_words]
        # Крок 3: discover з жанровим фільтром (або загальний якщо жанр не визначено)
        min_votes_genre = 1000 if genre_id else 5000
        discover_count_task = tmdb._discover_raw("vote_count.desc", 20, min_votes_genre,
genre_id)
        discover_rated_task = tmdb._discover_raw("vote_average.desc", 20,
min_votes_genre, genre_id)
        kw_results_list, by_count, by_rating = await asyncio.gather(
            asyncio.gather(*kw_tasks) if kw_tasks else asyncio.sleep(0, result=[]),
            discover_count_task,
            discover_rated_task,
        )
        # Дедуплiковані discover-результати
        seen_discover: set[int] = set()
        discover_items: list = []
        for m in by_count + by_rating:
            if m.tmdb_id not in seen_discover:
                seen_discover.add(m.tmdb_id)
                discover_items.append(m)
        # Keyword IDs
        keyword_ids: list[int] = []
        seen_kw_ids: set[int] = set()
        for kw_list in (kw_results_list or []):
            for kw in kw_list[:1]:
                if kw["id"] not in seen_kw_ids:
                    seen_kw_ids.add(kw["id"])
                    keyword_ids.append(kw["id"])
                    logger.debug(f"TMDB keyword: {kw['name']} (id={kw['id']})")
        keyword_items: list = []
        if keyword_ids:
            keyword_items = await tmdb.discover_by_keywords(keyword_ids, max_results=20,
min_votes=50)
        for m in discover_items + keyword_items:
            if m.url not in seen_urls:
                seen_urls.add(m.url)
                all_candidates.append(_tmdb_item_to_candidate(m))
```

```
    elif title_plan:
        items = await tmdb.search_multi(title_plan["query"], max_results=5)
        for m in items:
            if m.url not in seen_urls:
                seen_urls.add(m.url)
                c = _tmdb_item_to_candidate(m)
                c.is_direct_match = True
                all_candidates.append(c)
    if not all_candidates:
        # Останній fallback: text search по оригінальному запиту
        items = await tmdb.search_multi(fallback_query, max_results=10)
        for m in items:
            if m.url not in seen_urls:
                seen_urls.add(m.url)
                all_candidates.append(_tmdb_item_to_candidate(m))
    return all_candidates
def _lastfm_item_to_candidate(r) -> "Candidate":
    return Candidate(
        title=r.display_title,
        url=r.url,
        snippet=r.description[:200] if r.description else "",
        full_text=r.description,
        cover_url=r.cover_url,
        metadata={"genre": ", ".join(r.tags[:3])} if r.tags else {},
    )
async def _execute_music_plans(plans: list[dict], fallback_query: str) ->
list[Candidate]:
    from search.lastfm_client import LastFMClient
    lastfm = LastFMClient()
    all_candidates: list[Candidate] = []
    seen_urls: set[str] = set()
    for plan in plans:
        strategy = plan["strategy"]
        query = plan["query"]
        if strategy == "artist":
            items = await lastfm.search_artist_albums(query,
limit=settings.search_results_per_query)
        elif strategy == "album":
            items = await lastfm.search_albums(query,
limit=settings.search_results_per_query)
        else:
            items = await lastfm.top_albums_by_tag(query,
limit=settings.search_results_per_query)
            # Compound tag ("relaxing blues") може не існувати на Last.fm →
            # пробуємо складові слова справа наліво ("blues", потім "relaxing")
            if not items:
                for word in reversed(query.split()):
                    if word != query:
                        items = await lastfm.top_albums_by_tag(word,
limit=settings.search_results_per_query)
                        if items:
                            break
        for item in items:
            if item.url not in seen_urls:
                seen_urls.add(item.url)
                all_candidates.append(_lastfm_item_to_candidate(item))
    if not all_candidates:
        items = await lastfm.search_albums(fallback_query,
limit=settings.search_results_per_query)
        all_candidates = [_lastfm_item_to_candidate(r) for r in items]
    return all_candidates
def _book_result_to_candidate(r, is_ol: bool = False) -> "Candidate":
    return Candidate(
        title=r.title,
        url=r.url,
        snippet=r.description[:200] if r.description else "",
        full_text=r.description,
        cover_url=r.cover_url,
        metadata={k: v for k, v in {
            "author": r.author or None,
```

```
            "year": (str(r.year) if is_ol else r.year) or None,
            "rating": (f"{r.rating:.1f}/5" if r.rating else None),
        }.items() if v is not None},
    )
async def _execute_book_plans(plans: list[dict], fallback_query: str) -> list[Candidate]:
    from search.googlebooks_client import GoogleBooksClient
    gb = GoogleBooksClient(api_key=settings.googlebooks_api_key)
    all_candidates: list[Candidate] = []
    seen_urls: set[str] = set()
    author_plan = next((p for p in plans if p["strategy"] == "author"), None)
    topic_plan  = next((p for p in plans if p["strategy"] == "topic"), None)
    title_plan  = next((p for p in plans if p["strategy"] == "title"), None)
    # author + title: найточніший запит — inauthor + intitle одразу
    if author_plan and title_plan:
        combined_query = f'inauthor:"{author_plan["query"]}"
intitle:"{title_plan["query"]}"'
        results = await gb.search_books(combined_query,
max_results=settings.search_results_per_query)
        for r in results:
            if r.url not in seen_urls:
                seen_urls.add(r.url)
                all_candidates.append(_book_result_to_candidate(r))
        if all_candidates:
            return all_candidates
    # author + topic: Open Library дає повну бібліографію автора з subject-тегами.
    # Google Books для inauthor: повертає тільки популярні книги — рідкісніші
    # жанри (fantasy, western) можуть не потрапити в топ-10/20.
    if author_plan and topic_plan:
        from search.openlibrary_client import OpenLibraryClient
        ol = OpenLibraryClient()
        ol_books = await ol.search_books(
            f'author:"{author_plan["query"]}"',
            max_results=30,
            enrich_descriptions=False,  # subjects достатньо для реранкера, без
додаткових запитів
        )
        for r in ol_books:
            if r.url not in seen_urls:
                seen_urls.add(r.url)
                all_candidates.append(_book_result_to_candidate(r, is_ol=True))
        if not all_candidates:
            results = await gb.search_books(
                f'inauthor:"{author_plan["query"]}"', max_results=20
            )
            for r in results:
                if r.url not in seen_urls:
                    seen_urls.add(r.url)
                    all_candidates.append(_book_result_to_candidate(r))
        if all_candidates:
            return all_candidates
    for plan in plans:
        strategy = plan["strategy"]
        query = plan["query"]
        if strategy == "author":
            gb_query = f'inauthor:"{query}"'
        elif strategy == "title":
            gb_query = f'intitle:"{query}"'
        else:
            gb_query = query
        # Книжкові результати завжди йдуть через reranker (Google Books сортує по
        # популярності, а не по темі — тому is_direct_match не встановлюємо)
        results = await gb.search_books(gb_query,
max_results=settings.search_results_per_query)
        for r in results:
            if r.url not in seen_urls:
                seen_urls.add(r.url)
                all_candidates.append(_book_result_to_candidate(r))
    if not all_candidates:
        logger.warning("Google Books повернув 0, fallback на Open Library")
        from search.openlibrary_client import OpenLibraryClient
```

```
        ol_results = await OpenLibraryClient().search_books(
            fallback_query, max_results=settings.search_results_per_query
        )
        all_candidates = [_book_result_to_candidate(r, is_ol=True) for r in ol_results]
    return all_candidates
async def _fetch_candidates(
    search_query: str,
    category: str,
    entities: dict,
    session_factory,
    user_id_db: int | None = None,
) -> list[Candidate]:
    # Спочатку пробуємо персональний кеш (якщо є user_id), потім загальний
    async with session_factory() as session:
        async with session.begin():
            cache_repo = SearchCacheRepository(session)
            cached = await cache_repo.get(category, search_query, user_id_db)
            if cached is None and user_id_db is not None:
                cached = await cache_repo.get(category, search_query)
    if cached:
        log_metric("cache_hit", cache_type="search")
        return [
            Candidate(
                title=r["title"],
                url=r["url"],
                snippet=r.get("snippet", ""),
                full_text=r.get("full_text", ""),
                cover_url=r.get("cover_url"),
                metadata=r.get("metadata", {}),
            )
            for r in cached
        ]
    # Збираємо контекст юзера для персоналізації Groq prompt
    user_context: dict | None = None
    if user_id_db is not None:
        async with session_factory() as session:
            async with session.begin():
                from database.repositories.feedback import FeedbackRepository
                fb_repo = FeedbackRepository(session)
                rated = await fb_repo.get_user_rated_titles(user_id_db, limit=20)
                history_repo = HistoryRepository(session)
                recent_logs = await history_repo.get_user_history(user_id_db, limit=8)
        liked = [t for t, _cat, r in rated if r == 1][:5]
        disliked = [t for t, _cat, r in rated if r == -1][:3]
        recent = list({log.query_original for log in recent_logs if log.query_original})
        recent = [q for q in recent if q != search_query][:4]
        if liked or disliked or recent:
            user_context = {"liked": liked, "disliked": disliked, "recent": recent}
    log_metric("cache_miss", cache_type="search")
    candidates = await _fetch_from_source(search_query, category, entities,
user_context=user_context)
    # Персоналізовані результати кешуємо per-user; загальні — shared
    cache_user_id = user_id_db if user_context else None
    async with session_factory() as session:
        async with session.begin():
            cache_repo = SearchCacheRepository(session)
            await cache_repo.set(
                category,
                search_query,
                [
                    {
                        "title": c.title,
                        "url": c.url,
                        "snippet": c.snippet,
                        "full_text": c.full_text,
                        "cover_url": c.cover_url,
                        "metadata": c.metadata,
                    }
                    for c in candidates
                ],
```

```
                ttl_hours=settings.search_cache_ttl_hours,
                user_id=cache_user_id,
            )
    return candidates
# ---------------------------------------------------------------------------
# Пайплайн
# ---------------------------------------------------------------------------
async def _run_pipeline(
    query_original: str,
    query_translated: str,
    category: str,
    user_id_db: int,
    user_pref_vector,
    session_factory,
) -> list[Candidate]:
    from bot.dispatcher import reranker as _reranker
    log_metric("translation_done", original=query_original, translated=query_translated)
    lemmas = preprocess(query_original)
    entities = extract_entities(lemmas, category)
    # Передаємо оригінальний текст — щоб orchestrator кеш спрацьовував (той самий ключ що
в хендлері)
    candidates = await _fetch_candidates(query_original, category, entities,
session_factory, user_id_db)
    # Groq вже впорядкував кандидатів за релевантністю.
    # Реранкер тільки генерує embeddings для GRU-персоналізації.
    if candidates and _reranker:
        ranked = await _reranker.rerank(query_translated, candidates)
        # Індексуємо embeddings назад у оригінальний порядок Groq
        emb_by_url = {c.url: c.embedding for c in ranked if c.embedding is not None}
        for i, c in enumerate(candidates):
            if c.url in emb_by_url:
                c.embedding = emb_by_url[c.url]
            # Позиційний score: Groq #1 = 1.0, #2 = 0.95, ...
            c.semantic_score = max(0.0, 1.0 - i * 0.05)
    # Дедуплікація по назві всередині батчу (один серіал / одна серія томів → одна
картка)
    seen_titles: set[str] = set()
    deduped_titles: list[Candidate] = []
    for c in candidates:
        # Беремо перші 4 слова назви як ключ — відсіює "Vol 2", "Book One" тощо
        title_key = " ".join(c.title.lower().split()[:4])
        if title_key not in seen_titles:
            seen_titles.add(title_key)
            deduped_titles.append(c)
    candidates = deduped_titles
    async with session_factory() as session:
        async with session.begin():
            history_repo = HistoryRepository(session)
            deduped = await filter_duplicates(candidates, user_id_db, history_repo)
    # Якщо дедуплікація прибрала всіх — повертаємо оригінальні (вже бачені ліпше за
нічого)
    candidates = deduped if deduped else candidates
    combiner = ScoreCombiner()
    candidates = combiner.combine(candidates, user_pref_vector)
    top = candidates[: settings.recommendations_per_response]
    logger.info(f"Top-{len(top)} for [{category}] {query_translated!r}: " +
                ", ".join(f"{c.title!r}({c.semantic_score:.2f})" for c in top))
    return candidates[: settings.recommendations_per_response * 2]
# ---------------------------------------------------------------------------
# Відправка карток
# ---------------------------------------------------------------------------
async def _send_recommendations(
    message: Message,
    candidates: list[Candidate],
    category: str,
    query_original: str,
    query_translated: str | None,
    user_id_db: int | None,
    session_factory,
) -> None:
```

```
    cover_fetcher = CoverFetcher()
    top = candidates[: settings.recommendations_per_response]
    # Крок 1: обкладинки — поза транзакцією
    for c in top:
        if not c.cover_url:
            c.cover_url = await cover_fetcher.find_cover(c.title, category)
    # Крок 2: збереження в БД — окрема транзакція
    rec_ids: list[int | None] = [None] * len(top)
    if user_id_db is not None:
        async with session_factory() as session:
            async with session.begin():
                history_repo = HistoryRepository(session)
                for pos, c in enumerate(top, start=1):
                    rec = await history_repo.add(
                        user_id=user_id_db,
                        category=category,
                        query_original=query_original,
                        query_translated=query_translated,
                        title=c.title,
                        url=c.url,
                        description=c.full_text or c.snippet,
                        cover_url=c.cover_url,
                        semantic_score=c.semantic_score,
                        personalization_score=c.personalization_score,
                        final_score=c.final_score,
                        position=pos,
                        embedding=c.embedding,
                    )
                    rec_ids[pos - 1] = rec.id
    # Крок 3: надсилання карток — поза транзакцією
    for c, rec_id in zip(top, rec_ids):
        card = RecommendationCard(
            category=category,
            title=c.title,
            url=c.url,
            description=c.full_text or c.snippet,
            cover_url=c.cover_url,
            metadata=c.metadata,
        )
        text = card.to_telegram_message()
        keyboard = build_card_keyboard(rec_id) if rec_id is not None else None
        if c.cover_url:
            try:
                await message.answer_photo(
                    photo=c.cover_url,
                    caption=text,
                    parse_mode="HTML",
                    reply_markup=keyboard,
                )
            except Exception:
                await message.answer(text, parse_mode="HTML", reply_markup=keyboard)
        else:
            await message.answer(text, parse_mode="HTML", reply_markup=keyboard)
# ---------------------------------------------------------------------------
# Handlers
# ---------------------------------------------------------------------------
@router.message(RecommendationStates.idle)
@router.message(RecommendationStates.showing_results)
async def handle_recommendation_request(
    message: Message,
    state: FSMContext,
    db_user: Any = None,
) -> None:
    if not message.text:
        return
    text_lower = message.text.lower().strip()
    category = CATEGORY_MAP.get(text_lower)
    if category:
        await state.update_data(category=category, last_query=None, last_translated=None)
        await state.set_state(RecommendationStates.waiting_clarification)
```

```
        await message.answer(_CATEGORY_PROMPTS[category])
        return
    from bot.dispatcher import intent_classifier, orchestrator as _orchestrator
    # ------------------------------------------------------------------
    # Фаза 1: LSTM — швидкі системні інтенти без API
    # ------------------------------------------------------------------
    fast_intent, lstm_conf = intent_classifier.predict_fast(message.text)
    if fast_intent is not None:
        log_metric("intent_by_lstm", intent=fast_intent, confidence=lstm_conf)
        if fast_intent == "greeting":
            await message.answer("Привіт! Що шукаємо сьогодні?")
        elif fast_intent == "help":
            from bot.handlers.start import cmd_help
            await cmd_help(message)
        elif fast_intent == "more_options":
            data = await state.get_data()
            stored_category = data.get("category")
            stored_query = data.get("last_query")
            stored_translated = data.get("last_translated")
            if stored_category and stored_query:
                await _process_query(
                    message, state,
                    stored_query, stored_translated or stored_query,
                    stored_category, db_user,
                )
            else:
                await message.answer("Що саме хочеш побачити ще?")
        elif fast_intent in ("feedback_positive", "feedback_negative"):
            await message.answer("Дякую за відгук! Що ще шукаємо?")
        return
    # ------------------------------------------------------------------
    # Фаза 2: Groq — переклад + категорія + план пошуку за один виклик
    # ------------------------------------------------------------------
    # Спочатку плануємо тільки поточне повідомлення — щоб визначити категорію
    plan = await _orchestrator.plan(message.text)
    translated = plan.get("translation", message.text)
    new_category = plan.get("category", "movie")
    explicit = _detect_explicit_category(message.text)
    if explicit:
        new_category = explicit
    log_metric("intent_by_orchestrator", category=new_category, translated=translated)
    data = await state.get_data()
    stored_query = data.get("last_query")
    stored_category = data.get("category")
    # Продовжуємо сесію тільки якщо та сама категорія
    is_continuation = (
        data.get("in_session", False)
        and stored_query is not None
        and new_category == stored_category
    )
    if is_continuation:
        # Не конкатенуємо запити — уникаємо >2 планів пошуку (400 від Groq).
        # Категорія зберігається через stored_category.
        plan2 = await _orchestrator.plan(message.text, stored_category)
        translated = plan2.get("translation", message.text)
        await _process_query(message, state, message.text, translated, stored_category,
db_user)
    else:
        await _process_query(message, state, message.text, translated, new_category,
db_user)
@router.message(RecommendationStates.waiting_category)
async def handle_category_choice(
    message: Message, state: FSMContext, db_user: Any = None
) -> None:
    if not message.text:
        return
    category = CATEGORY_MAP.get(message.text.lower().strip())
    if not category:
        await message.answer("Вибери: Фільм, Книга або Музика",
reply_markup=CATEGORY_KEYBOARD)
```

```
        return
    await state.update_data(category=category)
    await state.set_state(RecommendationStates.waiting_clarification)
    await message.answer(_CATEGORY_PROMPTS[category])
@router.message(RecommendationStates.waiting_clarification)
async def handle_clarification(
    message: Message, state: FSMContext, db_user: Any = None
) -> None:
    if not message.text:
        return
    data = await state.get_data()
    category = data.get("category", "movie")
    from bot.dispatcher import orchestrator as _orchestrator
    if _orchestrator is not None:
        plan = await _orchestrator.plan(message.text, category)
        translated = plan.get("translation", message.text)
    else:
        translated = message.text
    await _process_query(message, state, message.text, translated, category, db_user)
async def _process_query(
    message: Message,
    state: FSMContext,
    query: str,
    query_translated: str,
    category: str,
    db_user: Any,
) -> None:
    await state.update_data(
        category=category,
        last_query=query,
        last_translated=query_translated,
        in_session=True,
    )
    await state.set_state(RecommendationStates.showing_results)
    wait_msg = await message.answer("Шукаю для тебе...")
    try:
        user_id_db = db_user.id if db_user else None
        user_pref_vector = None
        if user_id_db:
            async with AsyncSessionFactory() as session:
                async with session.begin():
                    session_repo = SessionStateRepository(session)
                    user_pref_vector = await
session_repo.get_preference_vector(user_id_db)
        candidates = await _run_pipeline(
            query, query_translated, category, user_id_db, user_pref_vector,
AsyncSessionFactory
        )
        await wait_msg.delete()
        if not candidates:
            await message.answer(
                "Нічого не знайшов за цим запитом. Спробуй переформулювати."
            )
            return
        await _send_recommendations(
            message=message,
            candidates=candidates,
            category=category,
            query_original=query,
            query_translated=query_translated,
            user_id_db=user_id_db,
            session_factory=AsyncSessionFactory,
        )
        if user_id_db:
            async with AsyncSessionFactory() as session:
                async with session.begin():
                    user_repo = UserRepository(session)
                    await user_repo.increment_requests(user_id_db)
    except Exception as e:
        logger.error(f"Помилка обробки запиту: {e}", exc_info=True)
```

```
        try:
            await wait_msg.delete()
        except Exception:
            pass
        await message.answer("Зараз не можу шукати, спробуй за хвилину.")
@router.message()
async def handle_no_state(message: Message, state: FSMContext) -> None:
    current = await state.get_state()
    if current is None and message.text and not message.text.startswith("/"):
        await state.set_state(RecommendationStates.idle)
        await message.answer(
            "Напиши /start щоб почати, або просто запитай що шукаєш — наприклад:\n"
            "«фільм жахів», «книга про кохання», «щось спокійне послухати»"
        )
```

settings.py

```
from typing import Any
from aiogram import Router
from aiogram.filters import Command
from aiogram.types import Message
from loguru import logger
from database.engine import AsyncSessionFactory
from database.repositories.session_state import SessionStateRepository
router = Router()
@router.message(Command("refresh_profile"))
async def cmd_refresh_profile(message: Message, db_user: Any = None) -> None:
    if not db_user:
        await message.answer("Не вдалось знайти твій профіль.")
        return
    await message.answer("Оновлюю профіль вподобань...")
    async with AsyncSessionFactory() as session:
        async with session.begin():
            from database.repositories.history import HistoryRepository
            from database.models import Feedback, RecommendationLog
            from sqlalchemy import select
            import numpy as np
            history_repo = HistoryRepository(session)
            result = await session.execute(
                select(Feedback.rating, RecommendationLog.embedding)
                .join(RecommendationLog, Feedback.recommendation_id ==
RecommendationLog.id)
                .where(
                    RecommendationLog.user_id == db_user.id,
                    RecommendationLog.embedding.is_not(None),
                )
                .order_by(RecommendationLog.recommended_at.desc())
                .limit(10)
            )
            rows = result.all()
            if not rows:
                await message.answer(
                    "Недостатньо оцінок для оновлення профілю. "
                    "Оціни кілька рекомендацій спочатку."
                )
                return
            embs = []
            feedbacks = []
            for rating, emb_bytes in rows:
                emb = np.frombuffer(emb_bytes, dtype=np.float32).copy()
                embs.append(emb)
                feedbacks.append(rating if rating else 0)
            try:
                from ranking.session_encoder.inference import SessionEncoder
                encoder = SessionEncoder()
                pref_vector = encoder.encode(embs, feedbacks)
                session_repo = SessionStateRepository(session)
                await session_repo.upsert(db_user.id, pref_vector,
interactions_since_update=0)
```

```
                await message.answer("Профіль оновлено. Наступні рекомендації будуть
точнішими.")
            except Exception as e:
                logger.error(f"Помилка оновлення профілю: {e}", exc_info=True)
                await message.answer("Не вдалось оновити профіль. Спробуй пізніше.")
```