Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

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

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Планування фізичних тренувань і дозвілля з урахуванням уподобань користувача
(тема)

Виконав:
Здобувач четвертого року навчання,
групи ІТШІ-21-4
Ілля Суворов
(власне ім'я, прізвище)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Освітня програма Штучний інтелект
(повна назва освітньої програми)
Керівник ст. викл. Вадим Губін
(посада, власне ім'я, прізвище)

Допускається до захисту

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

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

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

Освітня програма Штучний інтелект
(повна назва)

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Суворову Іллі Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Планування фізичних тренувань і дозвілля з урахуванням уподобань користувача

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

2. Термін подання студентом роботи до екзаменаційної комісії 19 червня 20 25 р.

3. Вихідні дані до роботи серверний застосунок, бази даних, нейрона модель. Документація із мови Python, документацыя ыз фреймворку Flask, документація із TensorFlow, література із історією нейроних мереж, рекомендпційних систем та агенті, мова програмування Python, мова маркування HTML, мова стилів CSS, мова для скріптингу у релаьноиу часі Java Script

4. Перелік питань, що потрібно опрацювати в роботі:
1) Аналіз предметної галузі
2) Обґрунтування вибору використаних технологій
3) Проектування системи

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Аналіз предметної галузі	20.05.2025	виконано
3	Нейронні мережі у рекомендаційних систем	22.05.2025	виконано
4	Постановка задачі	23.05.2025	виконано
5	Огрунтування вибору використанних технологій	24.05.2025	виконано
6	Проектування системи та опис програмної реалізації	25.05.2025	виконано
7	Оформлення пояснювальної записки	27.05.2025	виконано
8	Підготовка до захисту	14.06.2025	виконано
9	Захист перед ЕК	19.06.2925	виконано

Дата видачі завдання 19 травня 20 25 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст.викл. Вадим Губін
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 88 с., 18 рис., 3 дод., 21 джерело.

БАЗИ ДАНИХ, МАШИННЕ НАВЧАННЯ, ПРОГРАМНА РОЗРОБКА, PYTHON, SQLITE, TF.

Предмет дослідження – рекомендації за вподобаннями користувача із застосуванням методів агентних систем, машинного навчання з використанням нейронних мереж. Головна ціль це розробка програмного рішення з використанням агентів у якості покращувача взаємодії користувача із програмою, через LLM, а також використанням TensorFlow як бібліотеку для розробки нейронних, навчальних на динамічних даних моделей.

Для розробки використовується стек технологій, що включає Flask для серверної частини, SQLite для зберігання даних, а також або Tensor Flow NN для реалізації рекомендаційних моделей. Авторизація користувачів дозволяє зберігати персональні налаштування, а кожен запит на отримання рекомендацій впливає на подальші результати.

Мета дослідження – вирішення описаної вище проблеми за рахунок створення програмного рішення, здатного в прямому «ефірі» відповідати на запит користувача з тим, що він вказав на початку роботи із програмою.

Методи дослідження – у ході дослідження застосувалися методи розробки нейронних мереж з використанням прихованих шарів ReLU, а також результуючі SoftMax в рамках стандартів. Для розробки нейронних мереж використовувалася бібліотека TensorFlow, а для роботи із агентною системою ChatGPT-4o-mini LLM було використано бібліотеку LangGraph.

ABSTRACT

Bachelor’s thesis contains: 88 pp., 18 fig., 3 ann., 21 references.

AGENTS, APPLICATION DEVELOPMENT, DATA BASES, MACHINE LEARNING, PYTHON, RECOMMEND SYSTEM, SQLITE, TF.

The subject of the study is recommendations based on user preferences using agent systems, machine learning using neural networks. The main goal is to develop a software solution using agents as an enhancer of user interaction with the program, through LLM, as well as using TensorFlow as a library for developing neural, training models on dynamic data.

A technology stack is used for development, which includes Flask for the server side, SQLite for data storage, and Tensor Flow NN for implementing recommendation models. User authorization allows you to save personal settings, and each request for recommendations affects subsequent results.

The purpose of the study is to solve the problem described above by creating a software solution that can respond to the user's request live "on air" with what he indicated at the beginning of working with the program. Such an application can be useful to any representative of the middle and upper class, due to high employment and the inability to plan their lives in advance, and for those who do not want to hire a trainer and nutritionist.

Research methods – the research applied methods for developing neural networks using ReLU hidden layers, as well as the resulting SoftMax within the framework of standards. The TensorFlow library was used to develop neural networks, and the LangGraph library was used to work with the ChatGPT-4o-mini LLM agent system.

ЗМІСТ

ВСТУП

У сучасному світі темп життя значно прискорився, і люди все частіше прагнуть максимально ефективно використовувати свій вільний час. Планування дня допомагає підтримувати баланс між фізичною активністю, відпочинком і здоровим харчуванням. Проте самостійно складати такі плани буває складно, особливо якщо потрібно враховувати особисті вподобання, цілі та настрій.

Сьогодні все більшої популярності набувають персоналізовані рекомендації. Відомі сервіси, як-от Spotify для музики, Netflix для фільмів або MyFitnessPal для харчування, довели ефективність алгоритмів, які підлаштовуються під користувача. Однак ці сервіси працюють окремо, не поєднуючи різні аспекти повсякденного життя.

Запропонований веб-додаток буде унікальним тим, що об’єднає три важливі компоненти: тренування, музику та харчування. За допомогою нейронної мережі та бази даних додаток зможе аналізувати уподобання користувача і на основі цього формувати персоналізовані плани на день.

Окрім того, популяризація здорового способу життя робить цю тему ще більш актуальною. Люди хочуть бути у формі, харчуватися правильно, але при цьому не витрачати багато часу на планування. Додаток допоможе оптимізувати цей процес, створюючи зручний та індивідуальний план дня, що враховує як фізичне навантаження, так і емоційний стан через підбір відповідної музики.

Таким чином, розробка подібного веб-додатку є не лише актуальною, а й корисною для широкого кола користувачів, адже він допоможе зробити день більш продуктивним, збалансованим і приємним.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

Предметна галузь даної роботи охоплює створення сучасного веб-додатку, що надає користувачу можливість формувати персоналізований план дня, враховуючи фізичну активність, харчування та музичний супровід. У сучасних реаліях багато людей прагнуть організувати свій день так, щоб максимально поєднувати корисне з приємним: ефективні тренування, збалансоване харчування і улюблена музика, яка мотивує або допомагає розслабитися. Проте часто планування такого дня забирає чимало часу і вимагає певних знань щодо підбору вправ, раціону та навіть музичних жанрів, які підходять під настрій або тип активності.

Даний додаток покликаний вирішити цю проблему, створюючи комплексне рішення, яке враховує індивідуальні вподобання користувача. В основі лежить база даних, яка містить інформацію про різні види тренувань (кардіо, силові, функціональні тощо), страви з розрахунком калорійності та співвідношенням білків, жирів і вуглеводів, а також музичні треки, класифіковані за жанрами, ритмом і настроєм.

Система працює на основі реєстрації користувача, де він зазначає свої базові вподобання, цілі та бажаний стиль дня (наприклад, скинути вагу, підтримувати форму або просто активно провести час). На основі цих даних створюється перший базовий план дня. Подальша робота додатку стає динамічною завдяки вбудованій системі оцінювання. Користувач може оцінювати кожен елемент дня (тренування, страви, музику) за п’ятибальною шкалою. Це дозволяє додатку адаптувати рекомендації під конкретного користувача за допомогою нейронної мережі, яка поступово «навчається» і вдосконалює запропоновані варіанти.

Важливою особливістю додатку є гнучкість і можливість зміни уподобань. Якщо користувач часто ставить високі оцінки тренуванням певного типу або музиці певного жанру, система збільшує вагу цих елементів у рекомендаціях. Навпаки, якщо деякі елементи отримують низькі

оцінки, додаток поступово умовно «виключає» їх із плану через низькі оцінки, а також надає змогу повернути їжу або спортивне заняття назад до вподобань,але щоб не дратувати користувача, система більше звертає увагу на речі які користувач сказав, зо полюбляє.

Окрім цього, додаток підтримує можливість рандомних рекомендацій. Якщо користувач хоче спробувати щось нове, він може надати відповідний запрос на і отримати випадковий трек, страву або тренування. Після цього також можна залишити оцінку, щоб система врахувала нові вподобання.

Таким чином, предметна галузь охоплює поєднання кількох ключових напрямків: веб-розробку, аналіз даних, машинне навчання (нейронні мережі) та обробку природної мови для сприйняття запитів користувача. Додаток не просто рекомендує варіанти, а й створює особистий досвід для кожного користувача, адаптуючи свої рекомендації на основі історії взаємодій. Це робить систему більш гнучкою, зручною та привабливою для користувачів, які хочуть ефективно планувати свій день.

1.1 Історія створення нейронних мереж

Люди завжди думають лише про себе. Ця концепція «я» є, мабуть, найочевиднішою рисою людської природи. Існує багато теорій про природу свідомості, від метафізичних до фізіологічних. Обговорення цього питання в дебатах між філософами та теологами, фізіологами та анатомами зазнало невдачі, оскільки предмет важко вивчати. Люди, які покладаються на інтуїцію та інтуїцію, доходять висновків, які суперечать логіці фізики. З іншого боку, експериментатори виявили, що важко відстежувати мозок, і його поведінка погана. Коротше кажучи, потужні методи наукового дослідження, які змінюють наше розуміння природного світу, здається, не мають сили маніпулювати людьми.

Нейробіологи та нейроанатоми досягли великих успіхів. Завдяки зусиллям з вивчення структури та функцій нервової системи людини, вони

багато дізналися про «проводку» мозку, але мало про те, як він працює. Накопичуючи свої знання, вони виявили, що мозок – це дуже складна істота. Сотні мільярдів нейронів, кожен з яких пов'язаний із сотнями або тисячами інших, створюють систему, яка набагато складніша, ніж можна було б очікувати від суперкомп'ютера. Але мозок продовжує розширювати свій вплив в одній з найважливіших сфер людського життя. Краще розуміння того, як працює нейрон і його схеми спрацьовування, дозволило дослідникам створити математичні моделі для перевірки своїх гіпотез. Експерименти зараз проводяться на цифрових комп'ютерах і не залучають людей чи тварин, що означає багато ризиків і викликів. Попередня робота показала, що ці представлення не тільки моделюють структури мозку, але й здатні виконувати значущі завдання. Таким чином, виникли дві цілі нейронного моделювання, які існують і сьогодні: перша – зрозуміти функцію людського мозку в мозку та нервовій системі, а друга – створити обчислювальні системи для таких видів діяльності, як діяльність мозку.

Окрім досягнень у нейроанатомії та нейрофізіології, психологи розробили моделі навчання людини. Однією з цих моделей, найпліднішою, є D. Був приклад Хебба, який запропонував ідею глибокого навчання в 1949 році, що стало відправною точкою для алгоритмів навчання штучних нейронних мереж. Сьогодні, у поєднанні з багатьма іншими методами, це показало вченим того часу, наскільки потужними можуть бути нейронні мережі.

У 1950-х та 1960-х роках група дослідників, поєднавши ці генетичні та генетичні концепції, розробила перші генетичні мережі. Спочатку вони були розроблені для електронних мереж, потім перенесені в більш гнучке середовище комп'ютерного моделювання, яке залишається актуальним і сьогодні. Першим успіхом стало підвищення продуктивності та довіри. Мінський, Розенблатт, Вітроу та інші розробили одношарові мережі штучних нейронів. Вони називаються перцептронами і використовуються для таких завдань, як відстеження часу, відстеження електрокардіограми та

визначення положення. У певний момент здавалося, що таємницю свідомості розкрито, і що людський розум – це лише питання побудови змістовної мережі.

Але ця хибна думка незабаром була розвіяна. Мережі не змогли вирішити проблеми, які впливали на ті, що вже були вирішені. Ці неприємні проблеми вимагають ретельного аналізу. Мінський, використовуючи точні математичні методи, довів багато теорем про функціонування мереж.

Його дослідження призвело до статті, в якій він і Піпперт показали, що одношарові мережі використовувалися у випадках, коли вони не були оптимізовані для вирішення різноманітних проблем, включаючи операцію «виключне АБО». Мінський не погоджується з цією відмінністю:

Перцептрон здатний навчатися, незважаючи на (і завдяки) свої обмеження. Це має кілька переваг: лінійність, хороша теорема навчання та перевага моделі паралельних обчислень. Немає підстав вважати, що ці переваги будуть збережені при застосуванні до багатошарових систем. Однак ми стверджуємо, що важливим завданням науки є підтвердження (або спростування) переконання, що зміна Проте Я вважаю важливою задачею для дослідження підкріплення (або спростування) нашого інтуїтивного переконання, що такий перехід безплідний.

Можливо, буде відкрита якась могутня теорема про збіжність або знайдена глибока причина невдач дати цікаву "теорему навчання" для багатошарових машин.

Блиск і суворість аргументації Мінського, а також його престиж породили величезне довір'я до книги її висновки були незаперечливими. Розчаровані дослідники залишили поле досліджень заради більш перспективних областей, а уряди перерозподілили свої субсидії, і штучні нейронні мережі були забуті майже на два десятиріччя.

Проте декілька найбільш наполегливих вчених, таких як Кохонен, Гросберг, Андерсон продовжили дослідження. Нарівні з поганим фінансуванням і недостатньою оцінкою ряд дослідників мали труднощі з

публікаціями. Тому дослідження, опубліковані в сімдесяті і на початку вісімдесятих років, розкидані в масі різних журналів, деякі з яких маловідомі.

Поступово з'явився теоретичний фундамент, на основі якого сьогодні конструюються найбільш могутні багатошарові мережі. Оцінка Мінського виявилася зайво песимістичної, багато які з поставлених в його книзі задач вирішуються зараз мережами за допомогою стандартних процедур.

Переходячи до визначень: машинне навчання – це застосування штучного інтелекту, яке включає алгоритми, що аналізують дані, навчаються на їх основі, а потім застосовують отримані знання для прийняття обґрунтованих рішень.

Простим прикладом алгоритму машинного навчання є сервіс потокової передачі музики на вимогу, такий як Spotify.

Щоб Spotify міг прийняти рішення про те, які нові пісні чи виконавців вам рекомендувати, алгоритми машинного навчання пов’язують ваші вподобання з уподобаннями інших слухачів, які мають схожий музичний смак. Цей метод, який часто просто називають штучним інтелектом, використовується в багатьох сервісах, що пропонують автоматизовані рекомендації.

Машинне навчання підживлює всілякі завдання, що охоплюють різні галузі, від фірм із захисту даних, які виявляють шкідливе програмне забезпечення, до фінансових фахівців, які хочуть отримувати сповіщення про вигідні угоди. Алгоритми штучного інтелекту запрограмовані на постійне навчання таким чином, що імітує роботу віртуального особистого помічника – те, з чим вони справляються досить добре.

Історія машинного навчання починається в 1943 році з першої математичної моделі нейронних мереж, представленої в науковій статті «Логічне числення ідей, притаманних нервовій діяльності» Волтера Піттса та Воррена Маккалоха.

Потім, у 1949 році, була опублікована книга Дональда Гебба «Організація поведінки». Книга містила теорії про те, як поведінка пов'язана з нейронними мережами та діяльністю мозку, і згодом стала одним із монументальних стовпів розвитку машинного навчання.

У 1950 році Алан Тюрінг створив тест Тюрінга, щоб визначити, чи має комп'ютер справжній інтелект. Щоб пройти цей тест, комп'ютер повинен бути здатним обдурити людину, змусивши її повірити, що він також людина. Він представив цей принцип у своїй статті «Обчислювальна техніка та інтелект», працюючи в Манчестерському університеті. Вона починається словами: «Я пропоную розглянути питання: «Чи можуть машини мислити?»

Першу в історії програму для комп'ютерного навчання написав у 1952 році Артур Семюел. Програма була грою в шашки, і комп'ютер IBM удосконалювався в грі, чим більше він грав, вивчаючи, які ходи складають виграшні стратегії, та включаючи ці ходи у свою програму.

Потім, у 1957 році, Френк Розенблатт розробив першу нейронну мережу для комп'ютерів – перцептрон – яка моделювала розумові процеси людського мозку.

Наступний значний крок вперед у машинному навчанні відбувся лише у 1967 році, коли був написаний алгоритм "найближчого сусіда", що дозволив комп'ютерам почати використовувати дуже просте розпізнавання образів. Це можна було використовувати для прокладання маршруту для комівояжерів, починаючи з випадкового міста, але гарантуючи, що вони відвідають усі міста під час короткої екскурсії.

Дванадцять років по тому, у 1979 році, студенти Стенфордського університету винайшли «Стенфордський візок», який міг самостійно долати перешкоди в кімнаті. А в 1981 році Джеральд Деджонг представив концепцію навчання на основі пояснень (EBL), де комп'ютер аналізує навчальні дані та створює загальне правило, якого він може дотримуватися, відкидаючи неважливі дані.

У 1990-х роках робота над машинним навчанням перейшла від підходу, керованого знаннями, до підходу, керованого даними. Вчені почали створювати програми для комп'ютерів, щоб аналізувати великі обсяги даних та робити висновки – або «навчатися» – на основі результатів.

А в 1997 році Deep Blue від IBM шокував світ, перемігши чемпіона світу в шахи.

Термін «глибоке навчання» був введений у 2006 році Джеффрі Хінтоном для пояснення нових алгоритмів, які дозволяють комп'ютерам «бачити» та розрізняти об'єкти та текст на зображеннях і відео.

Чотири роки по тому, у 2010 році, Microsoft оголосила, що їхня технологія Kinect може відстежувати 20 людських рис зі швидкістю 30 разів на секунду, дозволяючи людям взаємодіяти з комп'ютером за допомогою рухів і жестів. Наступного року Watson від IBM переміг своїх конкурентів-людей у Jeopardy.

Google Brain був розроблений у 2011 році, і його глибока нейронна мережа могла навчитися виявляти та класифікувати об'єкти так само, як це робить кішка. Наступного року X Lab технологічного гіганта розробила алгоритм машинного навчання, який здатний автономно переглядати відео на YouTube, щоб ідентифікувати відео з котами.

У 2014 році Facebook розробив DeepFace – програмний алгоритм, здатний розпізнавати або перевіряти особи на фотографіях на тому ж рівні, що й люди.

Amazon запустила власну платформу машинного навчання у 2015 році. Microsoft також створила Distributed Machine Learning Toolkit, який дозволив ефективно розподіляти задачі машинного навчання між кількома комп'ютерами.

Потім понад 3000 дослідників у галузі штучного інтелекту та робототехніки, яких підтримали Стівен Гокінг, Ілон Маск та Стів Возняк (серед багатьох інших), підписали відкритого листа, в якому

попередили про небезпеку автономної зброї, яка вибирає та вражає цілі без втручання людини.

У 2016 році алгоритм штучного інтелекту Google переміг професійного гравця в китайській настільній грі Go, яка вважається найскладнішою настільною грою у світі та у багато разів складніша за шахи. Алгоритм AlphaGo, розроблений Google DeepMind, зумів виграти п'ять ігор з п'яти у змаганнях з Go.

Waymo почала тестувати автономні автомобілі в США у 2017 році, маючи лише резервних водіїв на задніх сидіннях автомобіля. Пізніше того ж року вони представили повністю автономні таксі у місті Фінікс.

У 2020 році, коли решта світу була охоплена пандемією, відкритий штучний інтелект анонсував новаторський алгоритм обробки природної мови GPT-3 з чудовою здатністю генерувати текст, подібний до людського, за запитом. Сьогодні GPT-3 вважається найбільшою та найдосконалішою мовною моделлю у світі, використовуючи 175 мільярдів параметрів та суперкомп'ютер штучного інтелекту Microsoft Azure для навчання.

1.2 Що собою представляє нейронна мережа

Оскільки у кваліфікаційній роботі використовуються декілька моделей штучного інтелекту, проллємо пролити більше світла на цю тему. У машинному навчанні нейронна мережа – це штучна математична модель, яка використовується для апроксимації складних нелінійних функцій. Її основне завдання полягає у виявленні закономірностей і взаємозв'язків у даних. Хоча ранні штучні нейронні мережі були апаратними пристроями, на кшталт електронних схем, сьогодні вони майже завжди реалізуються у вигляді програмного забезпечення. Ці програми здатні моделювати поведінку людського мозку, принаймні на концептуальному рівні, і використовуються для вирішення широкого спектра завдань – від розпізнавання образів до генерації тексту.

Нейрони у штучній нейронній мережі організовуються у шари. Інформація проходить від першого шару, який отримує вхідні дані, через один або кілька проміжних шарів, що виконують внутрішню обробку, до фінального шару, який формує результат. Така структура дозволяє мережі поступово трансформувати дані, витягуючи з них дедалі складніші особливості на кожному рівні. Сигнал, що надходить до кожного нейрона, є числом, яке обчислюється як лінійна комбінація вихідних значень нейронів попереднього шару, помножених на відповідні ваги. Потім це число подається на активаційну функцію, яка визначає остаточний вихід нейрона. Поведінка всієї мережі визначається вагою зв’язків між нейронами. Ці ваги є параметрами, які коригуються в процесі навчання з метою мінімізувати помилку між фактичними і очікуваними результатами.

Механізм навчання нейронної мережі передбачає проходження двох основних етапів. Під час прямого поширення (forward propagation) обчислюється вихід на основі поточних ваг. Потім виконується зворотне поширення помилки (backpropagation), під час якого обчислюється градієнт похибки і відбувається оновлення ваг. Цей процес ітеративно повторюється за допомогою оптимізаторів, таких як стохастичний градієнтний спуск (SGD), Adam або RMSProp, які забезпечують ефективне й швидке збіження навчання. У процесі навчання можуть виникати проблеми перенавчання, коли модель надто добре "запам’ятовує" навчальні дані і втрачає здатність узагальнювати. Для боротьби з цим застосовують техніки регуляризації, зокрема L1 та L2 штрафи, а також спеціальні підходи, такі як Dropout – випадкове вимкнення частини нейронів під час кожного епохи навчання, що запобігає надмірній залежності від окремих елементів. Також популярною є техніка раннього зупинення (early stopping), яка перериває навчання, коли показники на валідаційній вибірці починають погіршуватися.

Серед різновидів нейронних мереж існує кілька основних архітектур, кожна з яких має своє призначення. Перцептрон – це найпростіша форма

нейронної мережі, що складається лише з одного шару і використовується для лінійної класифікації. Багатошаровий перцептрон (MLP) містить кілька шарів нейронів і може моделювати більш складні функції. Згорткові нейронні мережі (CNN) спеціалізуються на обробці зображень і відео, виявляючи шаблони, такі як краї або текстури, у вхідних даних. Рекурентні нейронні мережі (RNN) створені для роботи з послідовностями – наприклад, текстами, часовими рядами або звуковими сигналами, дозволяючи мережі запам’ятовувати інформацію з попередніх кроків. Сучасні архітектури, такі як трансформери, лежать в основі моделей, здатних генерувати текст, перекладати мови та розв’язувати складні логічні задачі. Саме трансформери використовуються у відомих великих мовних моделях, таких як GPT, BERT, T5.

Активаційні функції відіграють ключову роль у поведінці нейронів, оскільки визначають, як обробляються сигнали. Серед них виділяють класичні функції, як-от сигмоїд (sigmoid) та гіперболічний тангенс (tanh), які використовувалися в перших нейромережах. Наразі найпопулярнішою стала функція ReLU (Rectified Linear Unit), яка має просту форму та ефективно працює з великими обсягами даних. Для задач багатокласової класифікації на вихідному шарі застосовується функція Softmax, яка перетворює виходи нейронів на ймовірності.

Термін «глибока нейронна мережа» вказує на моделі, які мають щонайменше три шари: вхідний, принаймні два прихованих та вихідний. Глибина мережі дозволяє їй розпізнавати дедалі складніші патерни. У міру проходження даних через шари мережі вона здатна виділяти низькорівневі ознаки (наприклад, лінії чи кольори на зображенні) і поступово переходити до високорівневих концепцій (наприклад, обличчя, об’єкти, емоції тощо).

Сфера застосування нейронних мереж є надзвичайно широкою. У медицині вони допомагають у діагностиці хвороб на основі знімків МРТ або рентгену. У фінансовому секторі – прогнозують ризики кредитування та керують портфелями інвесторів. У сфері автономного транспорту –

відповідають за розпізнавання об'єктів, планування маршрутів і ухвалення рішень у реальному часі. У побуті вони інтегровані у голосові помічники, системи рекомендацій музики, фільмів, новин. У сфері генеративного штучного інтелекту нейронні мережі створюють тексти, зображення, відео, музику, коди і навіть ідеї.

Таким чином, нейронні мережі стали невід’ємною складовою сучасного машинного навчання і цифрового світу загалом. Їх гнучкість, здатність до самоорганізації та адаптації роблять їх потужним інструментом для розв’язання як академічних, так і практичних задач у найрізноманітніших галузях науки, техніки та повсякденного життя.

1.3 Розвиток агентів

Застосунок можна вважати агентом, який допомогає людині з його побутом, а також він використує агента GPT-4o-mini для як LLM для кращої взаємодії користувача із застосунком.

LLM-и мають притаманні обмеження у знаннях та здатності до логічного мислення. AI-агенти з мовними можливостями вирішують ці проблеми, поєднуючи LLM-и з внутрішньою пам’яттю та зовнішнім середовищем, що дозволяє їм опиратися на існуючі знання або реальні спостереження.

Раніше системи мусили покладатися на створені вручну правила або навчання з підкріпленням, що ускладнювало адаптацію до нових середовищ. Мовні AI-агенти використовують вбудоване в LLM-и розуміння здорового глузду для вирішення нових завдань, зменшуючи залежність від людської анотації чи навчання методом проб і помилок.

Великі мовні моделі (LLMs, рисунок 1.1) виконують різні функції залежно від їхнього застосування:

– A – обробка тексту: у сфері обробки природної мови (NLP) LLMs отримують текст як вхідні дані та генерують текст як вихідні;

– B – мовні агенти: ці системи інтегрують LLMs у цикл зворотного зв'язку із зовнішнім середовищем, перетворюючи спостереження на текст і використовуючи LLM для прийняття рішень або виконання дій;

– C – когнітивні мовні AI-агенти: це розширені системи, які використовують LLMs не лише для взаємодії, але й для управління внутрішніми процесами, такими як навчання та міркування.

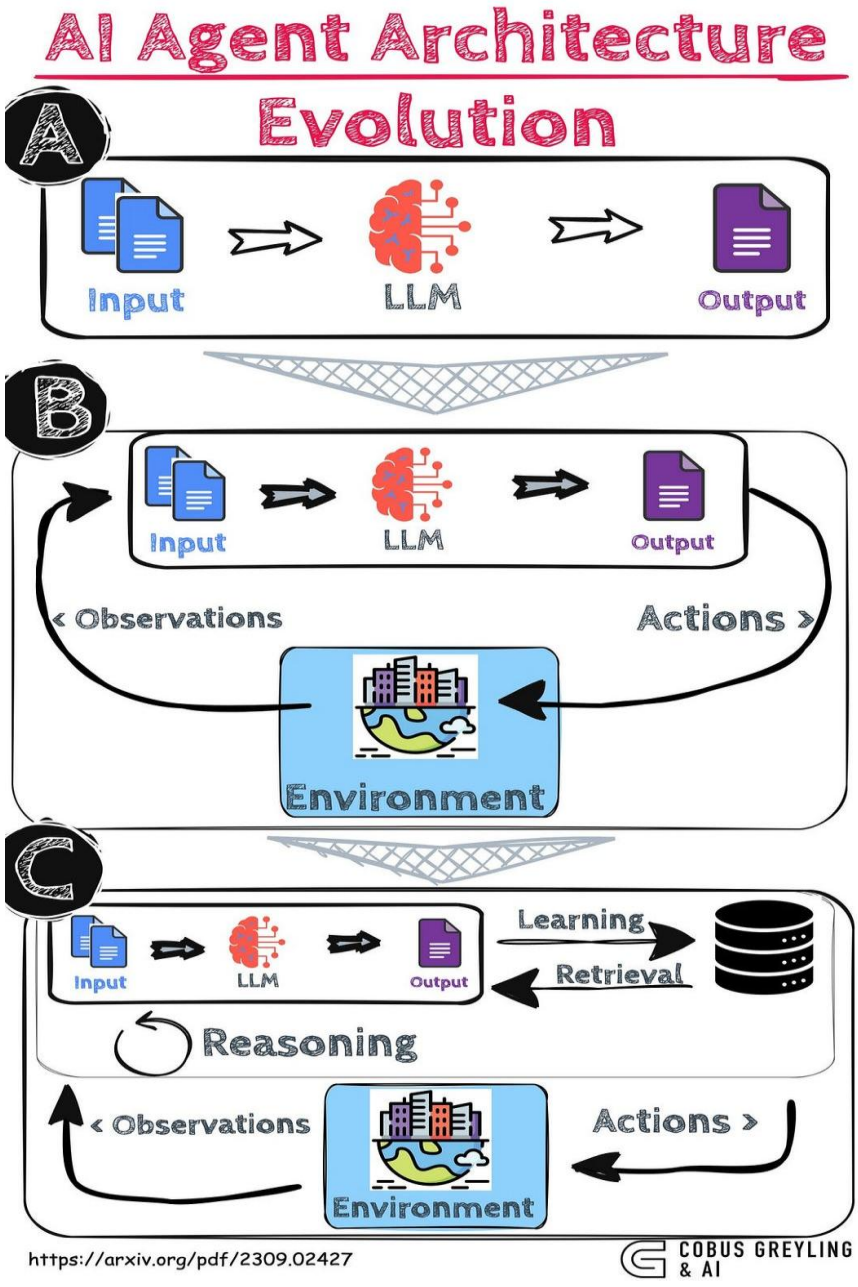

Рисунок 1.1 – Показ як для різних цілей використовуються агенти

1.3.1 Як проходив розвиток

Когнітивні архітектури включають епізодичну пам’ять, яка дозволяє зберігати та відтворювати конкретні події чи досвід, наприклад, згадування нещодавньої розмови. Семантична пам’ять містить загальні знання про світ, такі як факти та концепції. Епізодична пам’ять динамічна й залежить від контексту, тоді як семантична є більш стабільною і пов’язана з розумінням абстрактної, узагальненої інформації.

Простір дій AI-агента функціонує у двоетапній структурі. Внутрішні дії охоплюють процеси, як-от міркування, планування та оновлення його внутрішнього стану. Зовнішні дії пов’язані з взаємодією з середовищем, такими як виконання команд чи надання відповідей (рисунок 1.2).

Процес прийняття рішень AI-агента побудований як інтерактивний цикл, що складається з етапів планування та виконання. Ця ітеративна модель дає змогу аналізувати середовище, формулювати стратегію та діяти відповідно до нової інформації, удосконалюючи підхід у реальному часі.

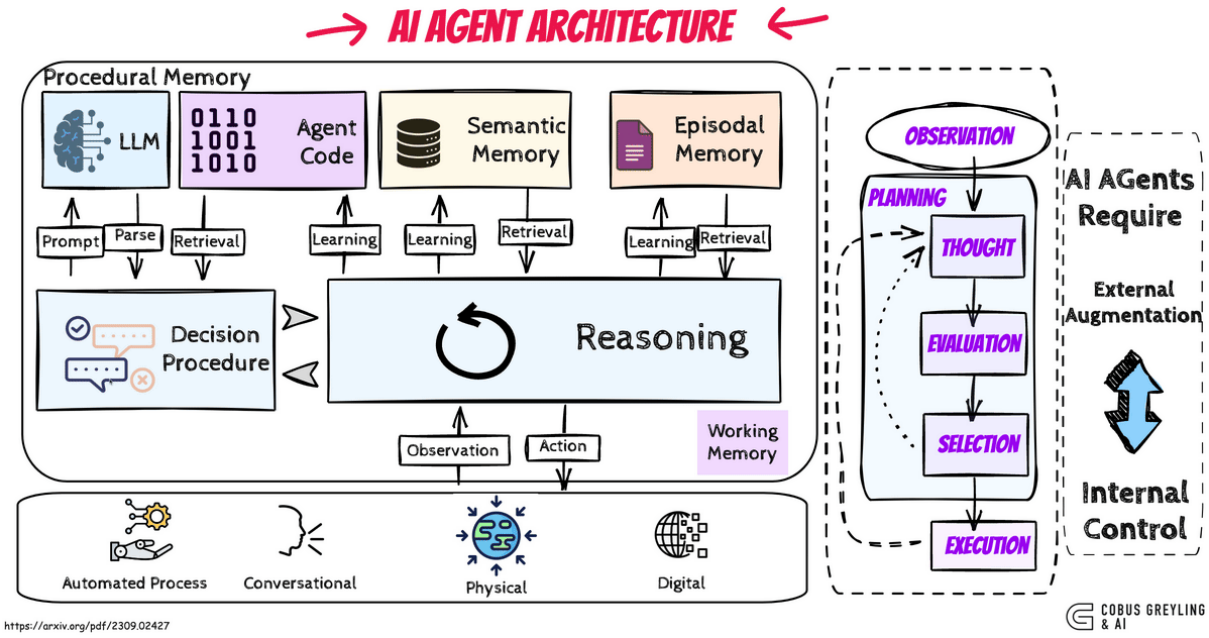

Рисунок 1.2 – Як проходить оркестрація агентів

1.3.2 Використання цифрового середовища

Цифрові середовища відіграють ключову роль у функціонуванні AI-агентів, дозволяючи їм взаємодіяти та виконувати різноманітні завдання. На сьогодні ці середовища здебільшого цифрові, включаючи мобільні та настільні операційні системи, а також інші цифрові екосистеми.

У таких умовах AI-агенти можуть працювати з іграми, API, вебсайтами та виконанням коду, використовуючи ці платформи як основу для виконання завдань та застосування знань.

Цифрові середовища пропонують ефективну й економічно вигідну альтернативу фізичній взаємодії, забезпечуючи простий спосіб розробки та оцінки AI-агентів.

1.3.3 Практичні застосування

У завданнях, пов’язаних з обробкою природної мови (NLP), цифрові API, такі як пошукові системи, калькулятори та перекладачі, часто інтегруються у операційні системи як спеціалізовані інструменти. Вони дозволяють AI-агентам виконувати конкретні завдання, що потребують зовнішніх знань або обчислень.

Зі зростанням можливостей AI-агентів їхня взаємодія у цифрових середовищах більше не буде обмежуватися статичними операціями. Вони розширюватимуть свої функції, створюючи базу для складніших систем.

Майбутнє AI-агентів полягає у їхньому фізичному втіленні, що дозволить їм працювати в реальному світі. Така трансформація відкриє нові перспективи для AI, надаючи можливість фізичної взаємодії з навколишнім середовищем, переміщення у просторі та виконання завдань у галузях, зокрема в робототехніці.

Перехід від суто цифрових середовищ до фізичних є важливим етапом розвитку AI, оскільки він потребуватиме інтеграції сенсорних даних,

фізичних дій і контекстно-орієнтованого ухвалення рішень. Це значно розширить можливості AI-агентів та їхнє практичне застосування.

1.4 Історія рекомендаційних систем

Оскільки використовуємого агента можна вважати рекомендаційною системою, що працює на ваговому принципі вподобань користувача, треба дещо розширити деталі про те, що несуть із собою рекомендаційн

Система рекомендацій (RecSys) або система рекомендацій (іноді замінюється такими термінами, як платформа, двигун або алгоритм), іноді називається лише «алгоритм» або «алгоритм», – це підклас системи фільтрації інформації, яка надає пропозиції щодо елементів, найбільш релевантних для конкретного користувача. Системи рекомендацій особливо корисні, коли людині потрібно вибрати елемент з потенційно величезної кількості елементів, які може пропонувати сервіс. Сучасні системи рекомендацій, такі як ті, що використовуються на великих сайтах соціальних мереж, широко використовують штучний інтелект, машинне навчання та пов'язані з ними методи для вивчення поведінки та вподобань кожного користувача та відповідної адаптації своєї стрічки. Як правило, пропозиції стосуються різних процесів прийняття рішень, таких як те, який продукт придбати, яку музику слухати або які онлайн-новини читати. Системи рекомендацій використовуються в різних сферах, загальновідомими прикладами яких є генератори списків відтворення для відео- та музичних сервісів, рекомендатори продуктів для інтернет-магазинів або рекомендатори контенту для платформ соціальних мереж та рекомендатори відкритого веб-контенту. Ці системи можуть працювати, використовуючи один тип вхідних даних, наприклад, музику, або кілька вхідних даних всередині та між платформами, такими як новини, книги та пошукові запити.

Існують також популярні рекомендаційні системи для певних тем, таких як ресторани та онлайн-знайомства. Рекомендаційні системи також були розроблені для дослідження дослідницьких статей та експертів, співробітників та фінансових послуг.

Платформа для пошуку контенту – це реалізована платформа рекомендацій програмного забезпечення, яка використовує інструменти рекомендаційної системи. Вона використовує метадані користувача, щоб знаходити та рекомендувати відповідний контент, одночасно зменшуючи поточні витрати на обслуговування та розробку.

Платформа для пошуку контенту надає персоналізований контент на веб-сайти, мобільні пристрої та телеприставки. Наразі існує широкий спектр платформ для пошуку контенту для різних форм контенту, починаючи від новинних статей та статей в академічних журналах і закінчуючи телебаченням. Оскільки оператори конкурують за те, щоб бути воротами до домашніх розваг, персоналізоване телебачення є ключовим фактором, що відрізняє сервіс.

Виявлення академічного контенту нещодавно стало ще однією сферою інтересів, і було створено кілька компаній, які допомагають академічним дослідникам бути в курсі актуального академічного контенту та випадково відкривати новий контент.

1.4.1 Приклад роботи рекомендаційної системи

Системи рекомендацій зазвичай використовують або спільну фільтрацію, або фільтрацію на основі контенту, або обидві, а також інші системи, такі як системи, засновані на знаннях. Підходи до спільної фільтрації будують модель на основі минулої поведінки користувача (товари, придбані або вибрані раніше, та/або числові оцінки, надані цим товарам), а також аналогічних рішень, прийнятих іншими користувачами.

Ця модель потім використовується для прогнозування товарів (або оцінок товарів), які можуть зацікавити користувача. Підходи до фільтрації на основі контенту використовують серію дискретних, попередньо позначених характеристик товару, щоб рекомендувати додаткові товари з подібними властивостями. Відмінності між спільною та контентною фільтрацією можна продемонструвати, порівнявши дві системи рекомендацій старовинної музики, Last.fm та Pandora Radio. Last.fm створює «станцію» рекомендованих пісень, спостерігаючи, які гурти та окремі треки користувач слухав регулярно, та порівнюючи їх із поведінкою прослуховування інших користувачів.

Last.fm відтворюватиме треки, які не відображаються в бібліотеці користувача, але часто відтворюються іншими користувачами зі схожими інтересами.

Оскільки цей підхід використовує поведінку користувачів, він є прикладом методу спільної фільтрації. Pandora використовує властивості пісні чи виконавця (підмножина з 400 атрибутів, наданих проектом Music Genome Project) для створення «станції», яка відтворює музику з подібними властивостями. Відгуки користувачів використовуються для уточнення результатів станції, применшуючи значення певних атрибутів, коли користувачеві «не подобається» певна пісня, та підкреслюючи інші атрибути, коли користувачеві «подобається» пісня. Це приклад підходу, заснованого на контенті. Кожен тип системи має свої сильні та слабкі сторони.

У наведеному вище прикладі Last.fm вимагає великої кількості інформації про користувача, щоб давати точні рекомендації. Це приклад проблеми холодного запуску, і це поширене явище в системах спільної фільтрації. Хоча Pandora потребує дуже мало інформації для запуску, її обсяг набагато обмеженіший (наприклад, вона може давати лише рекомендації, схожі на оригінальне початкове значення). Системи рекомендацій є корисною альтернативою алгоритмам пошуку, оскільки

вони допомагають користувачам знаходити елементи, які вони могли б не знайти інакше.

Слід зазначити, що рекомендаційні системи часто реалізуються за допомогою пошукових систем, які індексують нетрадиційні дані. Рекомендаційні системи були предметом кількох виданих патентів, і існує понад 50 програмних бібліотек, які підтримують розробку рекомендаційних систем, включаючи LensKit, RecBole, ReChorus та RecPack.

1.4.2 Історія рекомендаційних систем

Елейн Річ створила першу систему рекомендацій у 1979 році під назвою Grundy. Вона шукала спосіб рекомендувати користувачам книги, які могли б їм сподобатися. Її ідея полягала у створенні системи, яка б ставити користувачам конкретні запитання та класифікувала їх за класами уподобань, або «стереотипами», залежно від їхніх відповідей. Залежно від приналежності користувачів до стереотипів, вони отримували б рекомендації щодо книг, які могли б їм сподобатися.

Ще одна рання система рекомендацій, яка називалася «цифрова книжкова полиця», була описана в технічному звіті 1990 року Юссі Карлгреном з Колумбійського університету, а також впроваджена у великих масштабах та опрацьована в технічних звітах та публікаціях з 1994 року Юссі Карлгреном, який тоді працював у SICS, та дослідницькими групами під керівництвом Петті Мейс з MIT, Вілла Хілла з Bellcore та Пола Резніка, також з MIT, чия робота з GroupLens була нагороджена премією ACM Software Systems Award 2010 року.

Монтанер надав перший огляд систем рекомендацій з точки зору інтелектуального агента. Адомавічус надав новий, альтернативний огляд систем рекомендацій. Херлокер надає додатковий огляд методів оцінювання для систем рекомендацій, а Біл та ін. обговорили проблеми

офлайн-оцінювання. Біл та ін. також надали огляди літератури щодо доступних систем рекомендацій дослідницьких робіт та існуючих проблем.

1.4.3 Як були створені нейронні системи

Спільна фільтрація.

Один із підходів до проектування рекомендаційних систем, який широко використовується, – це колаборативна фільтрація. Колаборативна фільтрація базується на припущенні, що люди, які погодилися в минулому, погодяться в майбутньому, і що їм подобатимуться подібні типи елементів, як і в минулому. Система генерує рекомендації, використовуючи лише інформацію про профілі оцінок для різних користувачів або елементів. Знаходячи користувачів/елементи з історією оцінок, подібною до поточного користувача або елемента, вони генерують рекомендації, використовуючи цю околицю. Методи колаборативної фільтрації класифікуються як засновані на пам'яті та засновані на моделі. Відомим прикладом підходів, заснованих на пам'яті, є алгоритм, заснований на користувачеві, тоді як прикладом підходів, заснованих на моделі, є матрична факторизація (рекомендаційні системи).

Ключовою перевагою підходу колаборативної фільтрації є те, що він не спирається на контент, який можна аналізувати машиною, і тому він здатний точно рекомендувати складні елементи, такі як фільми, не вимагаючи «розуміння» самого елемента. Багато алгоритмів використовувалися для вимірювання подібності користувачів або подібності елементів у рекомендаційних системах. Наприклад, підхід k-найближчих сусідів (k-NN) та кореляція Пірсона, вперше реалізовані Алленом.

Під час побудови моделі на основі поведінки користувача часто розрізняють явні та неявні форми збору даних.

Приклади явного збору даних включають наступне:

– прохання до користувача оцінити товар за ковзною шкалою;

– прохання до користувача виконати пошук;

– прохання до користувача ранжувати колекцію товарів від улюбленого до найменш улюбленого;

– представлення користувачеві двох товарів і прохання вибрати кращий з них;

– прохання до користувача створити список товарів, які йому/їй подобаються (див. класифікацію Роккіо або інші подібні методи).

Приклади неявного збору даних включають таке:

– спостереження за товарами, які користувач переглядає в інтернет-магазині;

– аналіз часу перегляду товарів/користувачів;

– ведення обліку товарів, які користувач купує онлайн;

– отримання списку товарів, які користувач слухав або переглядав на своєму комп'ютері;

– аналіз соціальної мережі користувача та виявлення подібних вподобань та антипатій.

Підходи до колаборативної фільтрації часто мають три проблеми: холодний старт, масштабованість та розрідженість.

Холодний старт: Для нового користувача або товару недостатньо даних для надання точних рекомендацій. Примітка: одним із поширених рішень цієї проблеми є алгоритм багаторукого бандита.

Масштабованість: У багатьох середовищах, де ці системи надають рекомендації, є мільйони користувачів і товарів. Таким чином, для розрахунку рекомендацій часто потрібна велика обчислювальна потужність.

Різрідженість: Кількість товарів, що продаються на основних сайтах електронної комерції, надзвичайно велика. Найактивніші користувачі оцінили лише невелику підмножину загальної бази даних. Таким чином, навіть найпопулярніші товари мають дуже мало оцінок.

Одним із найвідоміших прикладів колаборативної фільтрації є колаборативна фільтрація товарів (люди, які купують x, також купують y), алгоритм, популяризований системою рекомендацій Amazon.com.

Багато соціальних мереж спочатку використовували колаборативну фільтрацію для рекомендації нових друзів, груп та інших соціальних зв'язків шляхом дослідження мережі зв'язків між користувачем та його друзями. Колаборативна фільтрація досі використовується як частина гібридних систем (рисунок 1.3).

Рисунок 1.3 – Як працюють колаборативні системи

Фільтрація на основі контенту.

Ще один поширений підхід до розробки систем рекомендацій – це фільтрація на основі контенту. Методи фільтрації на основі контенту базуються на описі елемента та профілі вподобань користувача. Ці методи найкраще підходять для ситуацій, коли відомі дані про елемент (назва, місцезнаходження, опис тощо), але не про користувача. Рекомендатори на основі контенту розглядають рекомендацію як проблему класифікації, специфічну для користувача, та вивчають класифікатор для вподобань та антипатій користувача на основі характеристик елемента. У цій системі ключові слова використовуються для опису елементів, а профіль

користувача створюється для позначення типу елемента, який подобається цьому користувачеві.

Іншими словами, ці алгоритми намагаються рекомендувати елементи, подібні до тих, які користувачеві подобалися в минулому або які він розглядає в даний час. Для створення цього часто тимчасового профілю не покладаються на механізм входу користувача. Зокрема, різні кандидатські елементи порівнюються з елементами, які користувач раніше оцінював, і рекомендуються найкраще відповідні елементи. Цей підхід сягає корінням у дослідження пошуку інформації та фільтрації інформації.

По суті, ці методи використовують профіль елемента (тобто набір дискретних атрибутів та ознак), що характеризують елемент у системі. Для абстрагування ознак елементів у системі застосовується алгоритм представлення елементів.

Широко використовуваним алгоритмом є представлення tf-idf (також зване векторним просторовим представленням). Система створює профіль користувачів на основі контенту на основі зваженого вектора ознак елементів. Ваги позначають важливість кожної ознаки для користувача та можуть бути обчислені з окремо оцінених векторів контенту за допомогою різних методів. Прості підходи використовують середні значення вектора оцінених елементів, тоді як інші складні методи використовують методи машинного навчання, такі як байєсівські класифікатори, кластерний аналіз, дерева рішень та штучні нейронні мережі, щоб оцінити ймовірність того, що користувачеві сподобається елемент.

Ключовою проблемою фільтрації на основі контенту є те, чи може система вивчати вподобання користувачів з дій користувачів щодо одного джерела контенту та використовувати їх для інших типів контенту. Коли система обмежена рекомендацією контенту того ж типу, що й користувач, цінність системи рекомендацій значно менша, ніж коли можна рекомендувати інші типи контенту з інших сервісів. Наприклад, корисно рекомендувати новинні статті на основі перегляду новин. Однак було б

набагато корисніше, якби музику, відео, продукти, обговорення тощо з різних сервісів можна було рекомендувати на основі перегляду новин.

Щоб подолати це, більшість систем рекомендацій на основі контенту зараз використовують певну форму гібридної системи. Системи рекомендацій на основі контенту також можуть включати системи рекомендацій на основі думок. У деяких випадках користувачам дозволено залишати текстові відгуки або відгуки про товари. Ці тексти, створені користувачами, є неявними даними для системи рекомендацій, оскільки вони є потенційно багатими ресурсами як характеристик/аспектів товару, так і оцінки/відчуття користувачів до товару. Характеристики, отримані з відгуків, створених користувачами, є покращеними метаданими товарів, оскільки, оскільки вони також відображають аспекти товару, такі як метадані, отримані характеристики широко цікавлять користувачів. Відчуття, отримані з відгуків, можна розглядати як оцінки користувачів за відповідними характеристиками.

Популярні підходи до систем рекомендацій на основі думок використовують різні методи, включаючи текстовий аналіз, пошук інформації, аналіз настроїв та глибоке навчання.

1.5 Нейронні мережі у рекомендаційних системах

У сучасному цифровому світі рекомендаційні системи відіграють ключову роль у формуванні користувацького досвіду. Вони забезпечують персоналізований підбір контенту в таких сервісах, як Netflix, Spotify, YouTube, Amazon.

Раніше такі системи працювали на основі колаборативної та контентної фільтрації:

– колаборативна фільтрація аналізує вподобання схожих користувачів;

– контентна фільтрація оцінює характеристики контенту (жанр, рейтинг тощо).

Однак ці методи мають обмеження:

– проблема холодного старту – система не знає, що рекомендувати новим користувачам;

– неможливість виявлення прихованих зв’язків між різними типами контенту;

– відсутність динамічного навчання на основі змін вподобань.

У моєму проєкті я використовую нейронну мережу, яка дозволяє усунути ці недоліки та значно покращити точність рекомендацій(нейронна мережа, див. рисунок 2.1).

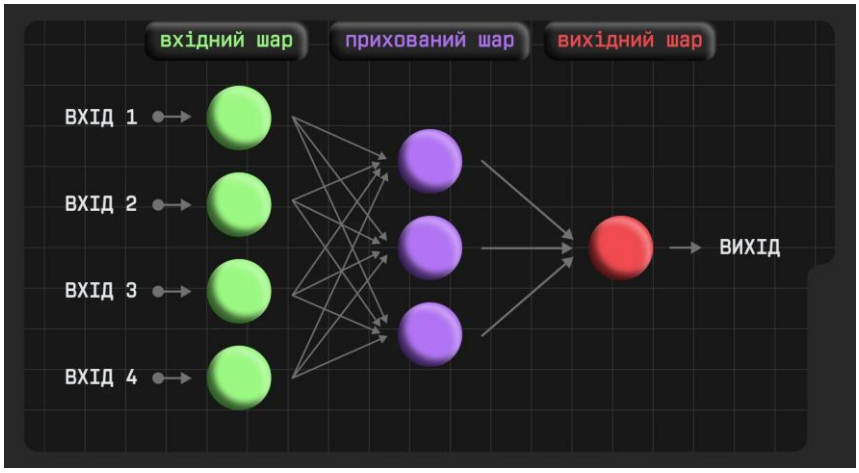

Рисунок 1.1 – Взаємодія із шарами

1.5.1 Архітектура нейронної мережі у рекомендаційних системах

Моя модель – це багатошарова нейронна мережа (MLP), що аналізує вподобання користувача та прогнозує, які фільми або треки йому можуть сподобатися.

Вона містить:

– вхідний шар, який отримує інформацію про контент (жанр, оцінку, характеристики);

– приховані шари, які виявляють приховані патерни між вподобаннями;

– вихідний шар, який повертає прогноз ймовірності, що користувачеві сподобається певний контент.

Математично кожен шар обчислюється за формулою

$$z^{(l)} = W^l * a^{l-1} + b^l \tag{1.1}$$
$$a^l = f(z^l)$$

де l-номер шару в нейроній мережі;

z^l – врівноважена сума входів на слої l до використання функції активації;

a^{l-1} – вектор активації зданього слоя;

b^l – вектор руху для шару l. Додається до кожного нейрона шару l;

a^l – вектор виходів даного шару після використання функції активації;

f – функція активації.

1.5.2 Огляд моделей в роботі

Ці моделі містять два приховані шари по 128 та 64 нейрони (лістинг 2.1) та двічі по 64 нейронів (лістинг 2.2), що дозволяє ефективно обробляти вподобання користувачів.

Лістинг 1.2 – Код моделі для спорту

```
    model_gym = models.Sequential([
    layers.Dense(128, activation='relu',
input_shape=(X_gym.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(len(y_gym.unique()), activation='softmax')
])
```

Лістинг 1.2 – Код моделі для їжі

```
 model_nutrition = models.Sequential([
    layers.Dense(64, activation='relu',
input_shape=(X_nutrition_scaled.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(len(y_nutrition.unique()),
activation='softmax')
])
```

1.5.3 Як нейронна мережа робить рекомендації

Метод Latent Factor Model (LFM) + нейромережа. Цей підхід поєднує матрицю вподобань користувачів із нейромережею. Спочатку створюється матриця оцінок користувачів (рисунок 1.2).

$$R = \begin{bmatrix} 5 & 3 & 4 & 0 \\ 4 & 0 & 5 & 2 \\ 0 & 4 & 0 & 5 \end{bmatrix}$$

Рисунок 1.2 – Матриця оцінок користувачів

Виконується розклад на приховані фактори:

$$R \approx P * Q^T \tag{1.2}$$

Ці вектори подаються на вхід нейромережі, яка вчиться прогнозувати оцінки навіть для тих комбінацій користувач-контент, яких немає у вихідних даних.

Я використовую глибоку нейронну мережу (DNN), яка отримує як вхід:

– контентні ознаки: жанр, рейтинг IMDB, ритм музики;

– поведінкові ознаки: історія прослуховувань, оцінки контенту.

Мережа прогнозує ймовірність того, що користувачеві сподобається контент (формула 1.3).

$$\gamma = \delta(W^T x + b) \tag{1.3}$$

де x – вектор ознак об`єкта;

W – Вектор ваг;

$W^T x$ – Скалярний добуток;

b – Зміщення (bias);

δ – Активаційна функція.

1.5.4 Функції активації та їхній вплив в моїй моделі

Функції активації допомагають нейромережі виявляти нелінійні залежності.

ReLU – використовується в прихованих шарах, прискорює навчання, зменшує проблему затухаючого градієнта:

$$f(x) = \max(0, x) \tag{1.4}$$

Softmax – використовується у вихідному шарі:

$$softmax(zi) = \frac{e^{zi}}{\sum_{j=1}^{n} e^{zj}}, \tag{1.5}$$

де zi – i-й елемент вхідного вектора z, що зазвичай є результатом виходу останнього шару нейронної мережі перед softmax;

e^{zi} – експоненціальне перетворення значенняzizi. Це підсилює відносні відмінності між елементами;

$\sum_{j=1}^{n} e^{zj}$ – нормалізаційний знаменник сума експонент всіх елементів вектора z;

n – кількість елементів у векторі z зазвичай дорівнює кількості класів у задачі класифікації.

Дозволяє отримати ймовірність уподобання.

Ми використовуємо Softmax, бо робимо передбачення на декількох категоріях. Якщо б ми використовували sigmoid, то він би не підійшов нам по суті програми, бо вона працює лише на одну категорію

1.5.5 Навчання моделі (Backpropagation)

Моя мережа використовує зворотне поширення помилки (Backpropagation) для оновлення ваг:

$$L(y, \gamma) = -(y \log(\gamma) + (1 - y) * \log(1 - \gamma)), \quad (1.6)$$

де y – реальна мітка класу (ground truth): 0 або 1;

γ – передбачена ймовірність від моделі, що y=1

Оновлення ваг відбувається за градієнтним спуском:

$$W = W - \mu \left(\frac{\vartheta L}{\vartheta W}\right) \quad (1.7)$$

де W – вектор або матриця ваг моделі (наприклад, в шарі нейронної мережі);

μ – швидкість навчання (learning rate) – мале додатне число, яке визначає, наскільки великий крок робити в напрямку мінімуму;

$\frac{\vartheta L}{\vartheta W}$ – градієнт функції втратLL за вагами W – показує напрямок та швидкість зміни втрат при зміні ваг.

1.6 Постановка задачі

Основним завданням даної кваліфікаційної роботи є розробка серверного застосунку, який має за собою ціль допомогти людям, які ведуть доволі активний тип життя, розроблювати для себе найбільш принагідний для них план тренувань та\або харчування. Мета проєкту є створення веб-сайту, що буде підтримувати зручний чатінг із рекомендаційною системою, що вчиться на вподобаннях людини у реальному часі.

Виходячи з результатів проведеного дослідження особливостей ствоерння такого застунку та аналізу потрібних для безпечного та зручного користування застосунком, виводимо такі вимоги:

– у програмі повинно бути реалізовано аунтефікацію та авторізацію користовучем з правильно налагодженим функціоналом для них;

– користувач має мати здатність вийти із свого облікового запису та знов увійти в нього не втративши збережені налагодження;

– сайт повинен надавати можливість користуватися чатінгом із рекомендаційною системою та отримувати валідні відповіді в правильному «напрямку» заданого агентною системою;

– сайт повинен надавати можливіть реакцій на відповіді рекмендаційної системи для кращого її налагодження;

– сайт повинен надавати можливість напряму казати вподобання та задавати кастомні ваги вправі або їжі, чи жанру музики;

– сайт повинен мати доступ до БД, щоб збирати інформацію про користувача, змінювати її, видаляти, аналізувати, тощо;

– проєкт має мати сервер, що використовує API для взамємодії із ендпоінтами, для взамаємодії із клієнтською частиною веб-застосунку. Сервер має бути написаний на мові Python із використанням фреймворку Flask;

– зберігання усіх даних провдиться через локальну БД із використанням SQLite.

Реалізація цих завдань сприятиме створенню серверного застосунку, який задовільнить потреби та очікування користувачів, що хочуть полегшити своє буття та не витрачати багато грошей на перносальних тренерів.

2 ОБҐРУНТУВАННЯ ВИБОРУ ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1 Вибір серверної розробки на Python та Flask

Вибір серверної розробки на Python та Flask для реалізації веб-дизайну був зосереджений на індивідуальному плануванні навчання та дозволі, що обумовлено низкою факторів.

По-перше, Flask – це легкий та гнучкий мікрофреймворк, який дозволяє створювати веб-додатки з мінімальними накладними витратами. Замість повноцінного фреймворку Django, Flask не нав'язує суворої структури проекту, що дозволяє розробнику краще контролювати архітектуру системи та безперешкодно інтегрувати сторонні бібліотеки та інструменти. Зокрема – це для машинного навчання та підготовки триб'юту.

Іншими словами, Python підтримує розгалужену екосистему, яка дозволяє ефективно впроваджувати інтелектуальні сервіси. Решта проекту потребуватиме роботи з окремими рекомендаціями, такими як CorystuWatch, Python у поєднанні з бібліотеками, такими як TensorFlow, Psychit-Learn або PyTorch, для оптимальних рішень. Flask легко інтегрується з такими інструментами, тому ви можете легко впроваджувати REST API або інтегрувати моделі машинного навчання з веб-скриптів.

По-третє, Flask має високу стабільність та гарну документацію, а також підтримується широким спектром розширень (flask-sqlchemy, flask-login, flask-restful тощо), що значно спростить виконання конкретних завдань – автентифікацію користувачів, архівацію, роботу з базами даних, приклади сесій тощо. Це дозволяє розробити повнофункціональну серверну шафу без будь-яких спеціальних складних деталей.

Тому ми обрали Flask як серверну технологію разом з нашим Python, що є раціональним з точки зору проекту, який вимагатиме гнучкості, інтеграції з машинним навчанням та легкої підтримки. Простота, розширюваність та активна екосистема роблять Flask ідеальним рішенням

для швидкого розвитку сучасних веб-сервісів з інтелектуальними функціями (рисунок 2.1).

Тип фреймворку	Flask	Django
Тип фреймворку	Мікрофреймворк (легкий)	Повноцінний фреймворк (все в одному)
Гнучкість	Висока (архітектура на ваш вибір, незалежна	Низька (вужча структура MVC, „Django way”)
Швидкий старт	Так (мінімум коду для першого запуску)	Так (більше налаштовань ’з коробкі”)
Вбудовані інструменти	Мінімум (база + розширювані можливостті)	Велика кількість вбудованих функцій (ORM, auth, admin)
Масштабованість	Гнучка, залежить від реалізації	Добра, але важко змінювати структуру
Навчання та вхідний поріг	Легкий для тих, хто знайомий з Python	Вищий через більше налаштованнь
Підтримка ML/AI бібліотек	Частково (орієнтований на ML--діаграми)	Підтримується (проекти з машинним навчанням)
Використання у малих проектах	Ідеально підходить (легкий та нероздутий)	Можливо (але великий за розміром)
Використання у великих проектах	Можливо (щілье розширення)	Ідеально підходить (засюапчоє все необхідне)

Рисунок 2.1 – Порівняння Flask із Django

Оскільки я використовую Flask для свого застосунку, це порівняння вельми доречне. В моєму застосунку я використовую цей фреймворк, а не Django, бо як можна бачити по таблиці та із опису вище, він не є саме добре підходящим для розробики на основі ML/DL. Саме цьому був обранний Flask. У цьому проекті використовується ML, а Flask дуже дружить із ним, а також цей проект не є таким великим, щоб використовувати більш складні фреймворки, або використовувати Python лише для бекенду.

2.2 Вибір розробки бази даних із SQLite

Обґрунтування розробки бази даних на SQLite для проекту ґрунтується на кількох ключових аспектах, які враховують характер проекту, вимоги до зберігання даних, а також простоту та ефективність розробки на початкових етапах.

По-перше, SQLite – це легка, вбудована система керування базами даних, яка не потребує окремого сервера. Це особливо зручно для малих або середніх проектів, особливо на етапі розробки MVP (мінімально життєздатного продукту) або прототипу. У випадку створення персонального помічника або програми для особистого використання, інтеграція з SQLite дозволяє значно спростити архітектуру та зменшити складність розгортання.

По-друге, SQLite є повністю безкоштовним, з відкритим вихідним кодом і не потребує окремої ліцензії чи серверного середовища. Зменшує витрати на інфраструктуру та адміністрування, що є особливим місцем для невеликих команд, стартапів або освітніх проектів.

По-третє, SQLite має чудову інтеграцію з безліччю фреймворків, зокрема Flask, що дозволяє швидко впроваджувати дані без додаткового налаштування. У поєднанні з бібліотекою ORM, такою як SQLalchemy, він забезпечує зручну та ефективну роботу з моделями даних.

По-четверте, бази даних SQLite зберігаються в одному файлі, що спрощує їх перенесення, резервне копіювання та міграцію між середовищами. Це особливо зручно у випадку, коли використання може працювати офлайн або локально без потреби в постійному мережевому з'єднанні.

На рисунку 2.2 наведено порівняння всіх окрестраторів, реляційних та нереляційних.

Рисунок 2.2 – Порівняння всіх окрестраторів, реляційних та нереляційних

Тому вибір SQLite як системи керування базами даних доцільний для проектів з невеликим навантаженням, автономних операцій або ранніх стадій розробки. Вона пропонує простоту, портативність, низьку вартість та легку інтеграцію, що дозволяє зосередитися на функціональності програми, а не на інфраструктурі. Саме цьому було прийнято рішення робити локальну базу даних, бо поки не планується робити її для когось окрім себе

3 ПРОЕКТУВАННЯ СИСТЕМИ

3.1 Структура класів системи

Дана кваліфікаційна робота було побудована на основі паттерну MVC (Model–view–controller), даний паттерн передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд та модуль керування. Візуальне відображення додатку було побудовано HTML код із використанням бібліотеки Flask

Розглянемо інтерфейси класів програми задля розуміння взаємодії з системою.

Клас SAssistant.

Інтелектуальний помічник, який використовує векторизовані дані (спорт, їжа, музика) та модель рекомендацій для створення персоналізованих планів і відповідей на запити користувача. Працює як генератор тренувань, дієти та музики – враховуючи вподобання користувача, або обирає випадково.

Функції та логіка роботи:

– extract_goal_and_period. Аналізує текстовий запит і витягує з нього фітнес-мету користувача (наприклад, «bulk», «cut» або «maintain») та кількість днів, на яку слід скласти план. Він використовує регулярні вирази для виявлення ключових слів і чисел. Якщо мету або кількість днів не вказано – підставляються значення за замовчуванням;

– generate_workout_plan. Створює програму тренувань на певну кількість днів, враховуючи ціль користувача (наприклад, нарощування маси) та його переваги щодо видів спорту, якщо вони є. Якщо переваги (ваги) надані, вони використовуються для більш точного підбору вправ. Інакше тренування формуються випадково з відповідного підрозділу датасету;

– generate_nutrition_plan. Формує харчовий план, який включає по три страви на день. Якщо користувач має улюблені продукти або надані їхні ваги – вони впливають на ймовірність появи певних страв у меню. Якщо таких даних немає, використовується випадковий вибір. Для кожної страви також генерується випадкова вага порції;

– random_meal. Просто повертає випадкову страву з датасету разом із випадково згенерованою вагою. Вона використовується як допоміжний інструмент при створенні планів харчування;

– recommend_plan. Це головна функція для створення цілісної рекомендації. Він аналізує текст користувача, витягує ціль і тривалість плану, а далі створює тренування, харчування або обидва варіанти, залежно від змісту запиту;

– random_music. Просто обирає випадкову пісню з музичного датасету, якщо у запиті згадується бажання послухати музику, без урахування вподобань користувача;

– recommend_music_by_genre. Створює персоналізовану рекомендацію музики на основі жанрових уподобань користувача. Він завантажує ці уподобання з бази, аналізує текст запиту, зіставляє з жанрами та намагається знайти найбільш релевантну пісню серед усіх доступних;

– get_user_preferences. Збирає всі наявні вподобання користувача з бази даних – від харчових звичок до улюблених жанрів музики – і повертає їх як словник, який потім використовується для точнішого налаштування відповідей;

– recommend_based_on_user. Є обгорткою над усією логікою. Він поєднує всі частини: отримує переваги користувача, визначає мету та період з тексту, і генерує персоналізований план тренувань і харчування, який максимально відповідає інтересам користувача. Якщо вподобання відсутні – використовуються загальні стратегії.

Класи БД.

Цей фрагмент коду реалізує модель бази даних для користувацької системи фітнес-помічника, яка включає класи користувача, його вподобань, а також збережених планів тренувань і харчування. Основна логіка обертається навколо взаємодії між класами User і Preference, де останній містить персоналізовані дані.

Клас User описує об'єкт користувача. Він містить унікальне ім’я користувача, хешований пароль та цілі (наприклад, «bulk» чи «cut»), а також має зв’язок «один до одного» з класом Preference. Функції set_password та check_password відповідають за створення та перевірку хешу пароля відповідно – вони використовують werkzeug.security для забезпечення безпеки.

Клас Preference пов’язаний з користувачем через зовнішній ключ user_id і зберігає детальні вподобання, такі як улюблені музичні жанри, продукти, спортивні акценти (наприклад, фокус на ноги чи спину), а також ваги – словники у форматі JSON, які визначають інтенсивність уподобань у спорті, їжі та вправах. Метод update_preferences дає змогу оновити ваги улюблених категорій у залежності від реакції користувача – оцінка конвертується у зміну ваги, що дозволяє системі адаптувати рекомендації. У випадку категорії «sport» або «food», вона відповідно оновлює текстові JSON-поля sport_focus або favorite_foods.

Методи update_sport_weight та update_food_weight виконують оновлення значень словників ваг для спорту та їжі. Якщо категорія або страва вже існує – вага збільшується на відповідну величину; інакше створюється новий запис. Вони працюють зі справжніми JSON-словниками в колонках MutableDict, а не з текстовими полями.

Методи get_sport_preferences та get_food_preferences просто повертають словники поточних переваг користувача відповідно до спортивних і харчових напрямків, що зберігаються в базі.

Метод update_user_goal формує нову мету користувача на основі його поточних вподобань. Він аналізує словник спортивних ваг і вибирає ту категорію, яка має найвищу оцінку – саме вона стає новою ціллю користувача, яка потім записується у відповідне поле goals у таблиці User.

Класи WorkoutPlan та MealPlan відповідають за зберігання планів тренувань та харчування відповідно. Вони обидва прив’язані до користувача через user_id і містять у полі plan_data структуру у вигляді JSON – цим забезпечується гнучке зберігання різноманітних планів.

Клас AgentState.

AgentState – це клас, що містить поточний стан агента під час обробки запитів. Основні поля:

– input – запит користувача;

– agent_outcome – результат взаємодії агента;

– intermediate_steps – збереження проміжних результатів обробки запиту.

3.2 Контролер системи

/api/survey [POST] приймає відповіді користувача на запитання опитування, що передаються у форматі JSON. Вони зберігаються в базі даних для подальшого аналізу. Відповіді повинні містити інформацію про їжу або вправи, які користувач любить чи не любить. Якщо формат відповіді коректний, метод зберігає ці дані в базі. У відповідь система надає статус «preferences saved», якщо збереження пройшло успішно.

/api/rate [POST] дозволяє користувачу оцінити різні елементи (їжу або вправи), змінюючи їхні ваги в базі даних. Оцінка може бути +1 або -1, і вона додається до відповідних елементів (їжа чи вправи). Після цього оновлюються відповідні категорії (food_weights або sport_weights) для цього користувача, і система відповідає статусом «ok», якщо операція була виконана успішно.

/survey [GET] відображає форму для заповнення опитування. Якщо користувач вже заповнив опитування (в базі даних є відповідні переваги), він буде перенаправлений на головну сторінку. Якщо ж користувач ще не заповнив опитування, йому буде показано форму для заповнення з початковими запитаннями.

/ [GET] є головною сторінкою додатку. Якщо користувач не авторизований, його перенаправляє на сторінку входу. Якщо ж користувач авторизований, відображається головна сторінка додатку.

/login [GET, POST] дозволяє користувачу увійти в систему. За методом GET користувач отримує форму для введення імені користувача та пароля. За методом POST система перевіряє введені дані: якщо ім'я та пароль коректні, створюється сесія користувача, і його перенаправляє на головну сторінку. Якщо введені дані неправильні, виводиться повідомлення про помилку.

/logout [GET] дозволяє користувачу вийти з системи, очищаючи дані сесії, і перенаправляє його на сторінку входу.

/register [GET, POST] дозволяє користувачу зареєструватися в системі. За методом GET показується форма для реєстрації, де користувач вводить своє ім'я, пароль, цілі, уподобання в музиці, їжі та спорті. За методом POST ці дані перевіряються, і якщо ім'я користувача вже існує в системі, виводиться помилка. Якщо все коректно, користувач створюється в базі даних, після чого йому пропонується пройти опитування для визначення його переваг.

/api/recommend [POST] приймає запит від користувача з промптом для отримання рекомендацій. Промпт передається агенту, який використовує дані користувача для формування рекомендацій. Результат повертається у вигляді JSON відповіді з текстом рекомендації.

get_recommendation(prompt, user_id=None) – це допоміжна функція, яка формує запит для агента, додаючи до промпту ID користувача для персоналізації, а потім викликає агент для отримання рекомендації.

save_initial_preferences(user_id, answers) – це метод, що зберігає відповіді користувача на запитання опитування в таблиці Preference в базі даних. Для кожної відповіді встановлюються ваги для їжі та вправ, а також визначається мета користувача, така як набір м'язової маси або схуднення.

Ці ендпоінти забезпечують основні функції для роботи з користувачами: реєстрація, авторизація, збори та збереження даних про вподобання користувача, а також надання персоналізованих рекомендацій на основі цих даних.

3.3 Агентна система та її роль у проєкті

Підстави використання агентної системи в цьому проєкті:

– автоматизація ухвалення рішень – агентна система визначає, яка відповідь буде найбільш релевантною для користувача;

– адаптивність – агент здатний змінювати поведінку залежно від вподобань користувача;

– гнучкість – система легко масштабується та розширюється за рахунок додавання нових вузлів ухвалення рішень.

Агентна система побудована на базі LangGraph, що дозволяє створювати ієрархію ухвалення рішень.

Агентна система розроблена на основі LangChain та LangGraph, що дозволяє створити динамічний механізм ухвалення рішень.

Основний алгоритм роботи:

– прийом запиту користувача – агент отримує текстовий запит;

– визначення категорії запиту – аналізує, чи стосується він музики, фільмів або фізичних тренувань;

– передача запиту у відповідний обробник – викликає модуль RecommendationModel або SAssistant;

– отримання відповіді та її адаптація – формує персоналізовану рекомендацію з урахуванням вподобань користувача;

– збереження результатів для майбутнього аналізу – фіксує взаємодію в базі даних для подальшого вдосконалення рекомендацій (лістинг 3.1).

Лістинг 3.1 – Реалізація агентної системи

```
from langchain_core.agents import AgentFinish
from langgraph.graph import END,StateGraph
from nodes import execute_tools,run_agent_reasoning
from State import AgentState
import pandas as pd
AGENT_REASON="agent_reason"
ACT="act"
def sould_cont(state:AgentState):
    if isinstance(state["agent_outcome"],AgentFinish):
        return END
    else:
        return ACT
flow=StateGraph(AgentState)
flow.add_node(AGENT_REASON,run_agent_reasoning)
flow.set_entry_point(AGENT_REASON)
flow.add_node(ACT,execute_tools)
flow.add_conditional_edges(
    AGENT_REASON,
    sould_cont,
)
flow.add_edge(ACT,AGENT_REASON)
agent=flow.compile()
```

3.4 Взаємодія з користувачем

Розробка інтерфейсу користувача базується на принципах зручності та простоти взаємодії. Головні аспекти, які було враховано при створенні веб- та десктопного застосунку:

– мінімалістичний дизайн без зайвих елементів;

– інтуїтивно зрозуміле розташування кнопок та форм введення;

– адаптивність інтерфейсу для різних пристроїв;

– чіткі підказки для користувачів у випадку помилок.

Функціональні здібності програми:

– користувач вводить запит у веб-інтерфейсі або десктопному застосунку;

– система аналізує запит та передає його в агентну систему;

– агент визначає категорію та звертається до відповідного модуля;

– генерується персоналізована рекомендація;

– результат виводиться на екран користувача з можливістю оцінки.

Передбачено систему повідомлень для користувачів у разі:

– відсутності результатів за запитом;

– некоректного введення даних;

– проблем із сервером або мережею.

3.5 Розширюваність та майбутні покращення

Проєкт має модульну архітектуру, що дозволяє легко інтегрувати нові сервіси, такі як:

– API стрімінгових сервісів (Spotify, Netflix) для розширення рекомендацій;

– голосові помічники (Google Assistant, Alexa) для голосового керування;

– соціальні мережі для персоналізованого аналізу вподобань користувачів.

3.6 UML-діаграма взамємодії сервісу та користувача

На рисунку 3.1 я показав як саме працює мій застосунок зі сторони зв’язку сервера та клієнта. А саме, спочатку людина має або зарєєструватися або залогінитися, щоб мати змогу користуватися моїм застосунком, бо без цього система персонального вдосконалення на якійсь особі працювати просто не буде, через не змогу постійно мати у «пам’яті» інформацію про

нього. Звісно, можна було би звернустися до «кукісів» , але вони також не є постійним зберігачем інформації, тому використання бази даних для цього є найбільш достовірним підходом. Оскільки кожен користувач має свії вподобання, то вони зберігаються, також як і він. Також, користувач проходить невеликий опитувач, щоб налагодити модель на перші взаємодії.

Коли ж діло доходить до роботи із рекомендаційною системою, то користувач може ставити лайки та дізлайки, щоб налагоджувати систему під себе, ці відповіді також зберігаються в базу через API, Коли користувач звертається до системи, його ID передається до ReAct агента. Агент аналізує натуральну мову, потім звертається до потрібного методу, а потім персоналізує відповідь.

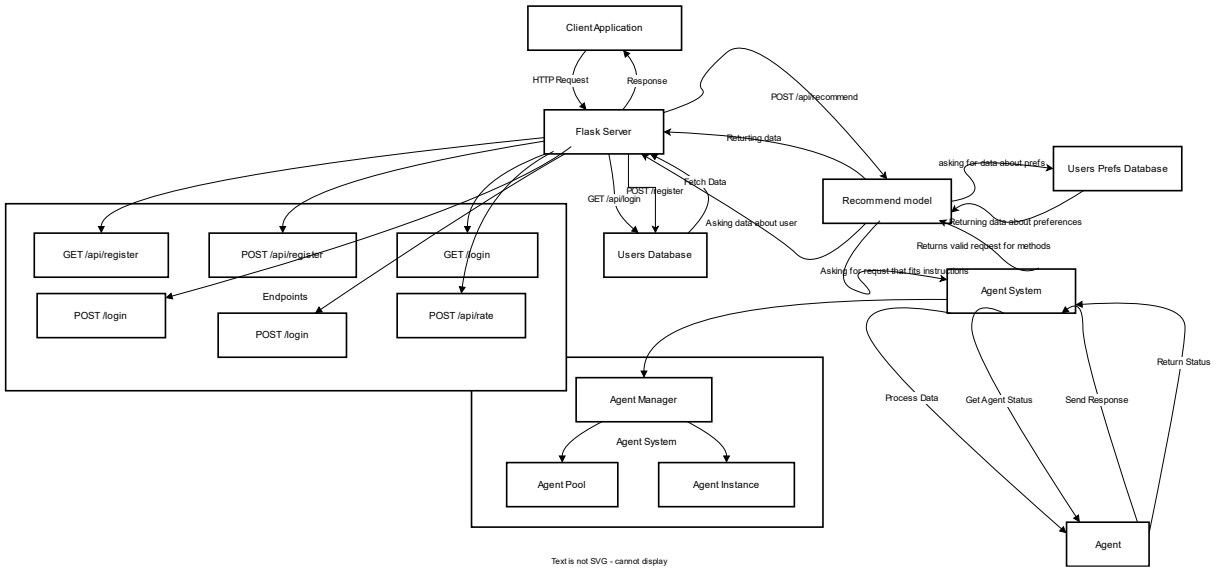

Рисунок 3.1 – UML діагарама клієнт-серверної роботи

Створені сутності для реалізації програми

а) тренажерний зал (Gym):

– Exercise: зберігає інформацію про вправи, їх типи, необхідне обладнання та м’язові групи, які вони тренують;

– id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор вправи;

– name (TEXT) – назва вправи;

– equipment (TEXT) – необхідне обладнання;

– utility (TEXT) – тип корисності вправи (основна чи допоміжна);

– mechanics (TEXT) – механіка виконання (ізольована чи складна);

– force (TEXT) – тип руху (штовхання чи тягнення);

– execution (TEXT) – опис виконання вправи;

– target_muscles (TEXT) – основні цільові м’язи;

– difficulty (INTEGER) – рівень складності (1–5);

б) музика (Music):

– Track: містить інформацію про музичні треки, їх жанр, текст та емоційні характеристики;

– id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор пісні;

– artist_name (TEXT) – ім'я виконавця;

– track_name (TEXT) – назва пісні;

– release_date (INTEGER) – рік випуску;

– genre (TEXT) – жанр музики;

– lyrics (TEXT) – текст пісні;

– danceability (FLOAT) – оцінка танцювальності (0-1);

– energy (FLOAT) – рівень енергії (0-1);

– valence (FLOAT) – оцінка настрою (0-1);

– topic (TEXT) – головна тема пісні (романтика, сум, світогляд тощо);

в) харчування (Nutrition):

– FoodItem: зберігає інформацію про їжу, її вагу, калорійність та макроелементи;

– id (INTEGER, PRIMARY KEY) – унікальний ідентифікатор продукту;

– label (TEXT) – назва продукту;

– weight (INTEGER) – вага у грамах;

– calories (INTEGER) – кількість калорій;

– protein (FLOAT) – білки (г);

– carbohydrates (FLOAT) – вуглеводи (г);

– fats (FLOAT) – жири (г);

– fiber (FLOAT) – клітковина (г);

– sugars (FLOAT) – цукри (г);

– sodium (INTEGER) – натрій (мг).

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Специфікації застосунку

Застосунок реалізований у вигляді веб-додатка з використанням Flask для серверної частини та. Він призначений для генерації персоналізованих рекомендацій щодо музики, фізичних тренувань та харчування. Реалізована структура наведена на рисунку 4.1.

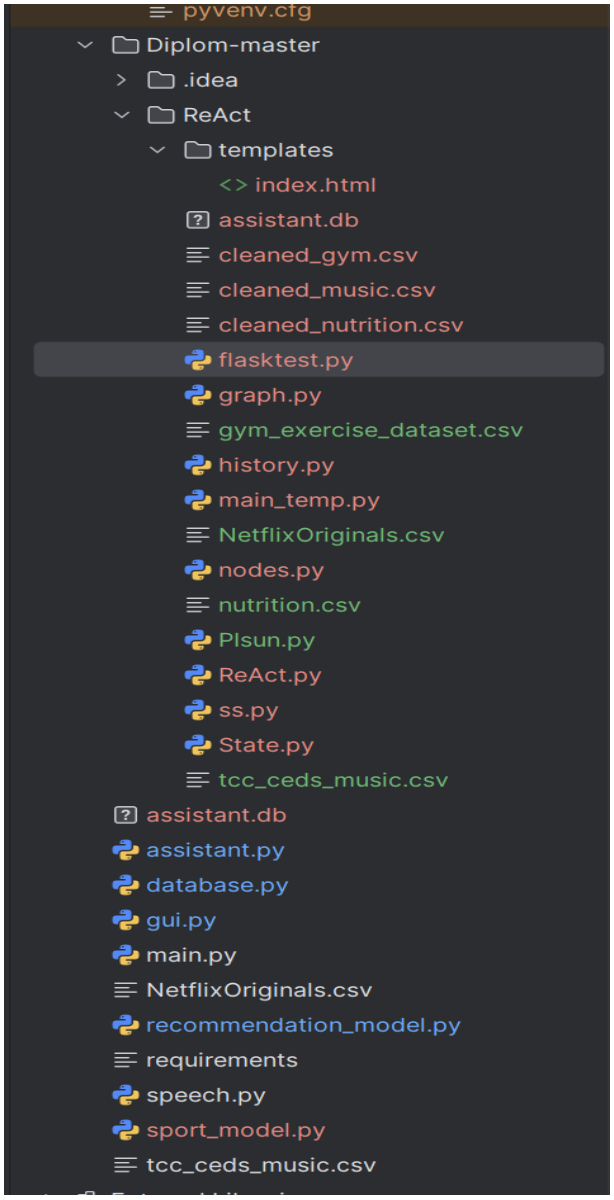

Рисунок 4.1 – Загальна структура застосунок

Функціональні можливості:

– автоматичне визначення категорії запиту (фільми, музика, харчування, тренування);

– використання агентної системи для покращення точності рекомендацій;

– можливість отримання як персоналізованих, так і випадкових рекомендацій;

– збереження історії взаємодій користувача для подальшого аналізу;

– графічний інтерфейс для зручного користування на ПК.

Технічні характеристики:

– мова програмування: Python 3.9+;

– фреймворки: Flask, PyQt6, LangChain;

– база даних: SQLite;

– модулі машинного навчання: TensorFlow (опціонально);

– API-запити: RESTful API.

4.2 Архітектура застосунку

Реалізація застосунку заснована на мікросервісному підході з використанням Python та Flask.

Основними компонентами є:

– серверна частина (Backend) – відповідає за обробку запитів, логіку обробки даних та взаємодію з базою даних;

– база даних (SQLite) – використовується для збереження історії взаємодій користувача, рекомендацій та їхніх оцінок;

– агентна система (LangChain, LangGraph) – керує обробкою запитів користувача та ухваленням рішень щодо рекомендацій;

– кожен з цих компонентів працює незалежно та взаємодіє через API-запити.

4.3 Графічне відображення системи

Візуальний інтерфейс написаний на фреймворку Flask з html. У ньому використовуються як звичайні сторінки для перегляду, так і модальні для створення та редагування сутностей.

У даній роботі використовується підхід Single Page Application (SPA) у якому перезавантаження сторінок не відбуваються у звичному значенні. При використанні даного підходу використовуються динамічні зміни однієї сторінки, де змінюються лише певні елементи сторінки. Структура сторінок базується на двох головних елементах:

– незмінному блоку header, де зберігаються усі навігаційні компоненти;

– змінній області, де відображується вся наявна на сторінці інформація на основі відповідного URL посилання;

– після цього в якості основних компонентів маємо сторінки та модальні форми для редагування даних.

Index – головна сторінка застосунку де відображається інформація про запити у чат та міститься ключові навігаційні елементи системи та статистичні розрахунки на основі активних замовлень.

4.4 Основні модулі програмного коду

Модуль обробки запитів.

Цей модуль приймає вхідні дані користувача, аналізує їх та направляє до відповідного підмодуля (лістинг 4.1).

Лістинг 4.1 – Модуль обробки запитів

```
@app.route("/api/recommend", methods=["POST"])
def recommend():
    if "user_id" not in session:
        return jsonify({"error": "Unauthorized"}), 403
    data = request.json
```

Продовження лістингу 4.1

```
        prompt = data.get("prompt", "")
        user_id = session["user_id"]
        pref =
    Preference.query.filter_by(user_id=user_id).first()
        if pref:
            pref.update_user_goal()
        recommendation = get_recommendation(prompt,
    user_id)
        return jsonify({"response": recommendation})
```

Модуль рекомендацій (лістинг 4.2).

Цей модуль обробляє запити щодо фільмів, музики та фізичних тренувань.

Лістинг 4.2 – Модуль рекомендацій

```
def get_recommendation(prompt, user_id=None):
    inputs = {"input": prompt}
    if user_id is not None:
        prompt = prompt + f" my user Id is {user_id}"
        inputs = {"input": prompt}
    response = agent.invoke(inputs)
    if isinstance(response, dict) and "agent_outcome" in
response:
        return
response["agent_outcome"].return_values["output"]
    return str(response)
```

На рисунку 4.2 зображено функцію save_initial_preferences, яка зберігає початкові вподобання користувача. Функція перевіряє, чи вже існує об’єкт переваг для поточного user_id. Якщо ні – створюється новий запис. Далі на основі отриманих відповідей формується словник з вагами для харчування (food_weights) та вправ (liked_exeicies). В залежності від того, чи користувач поставив позитивну оцінку (наприклад, «liked»: true), елемент отримує підвищену вагу. Також у фіналі оцінка по вправі «bench press» впливає на визначення спортивної цілі (bulk або cut), що зберігається в sport_weights.

```python
def save_initial_preferences(user_id, answers):  1 usage  new *
    pref = Preference.query.filter_by(user_id=user_id).first()
    if not pref:
        # Создаём новую запись, если не существует
        pref = Preference(user_id=user_id)
        db.session.add(pref)

    sport_weights = {}
    food_weights = {}
    liked_exeicies = {}
    print(f"siska :{answers}")
    for ans in answers:
        print(f"ansewr {ans}")
        if 'category' not in ans:
            continue  # Пропустить некорректные ответы

        item = ans["item"]
        liked = ans["liked"]

        if ans["category"] == "food":
            food_weights[item] = 3.5 if liked else 1.0
        elif ans["category"] == "exercise":
            liked_exeicies[item] = 3.5 if liked else 1.0

    pref.food_weights = food_weights
    pref.liked_exercises = liked_exeicies

    if pref.liked_exercises["bench press"] > 2:
        pref.sport_weights["bulk"] = 3
    else:
        pref.sport_weights["cut"] = 3

    db.session.commit()
```

Рисунок 4.2 – Метод збереження переваг користувача

На рисунку 4.3 показано реалізацію API-методу /api/rate, що дозволяє користувачу оновлювати власні вподобання після кожної взаємодії із системою. Якщо об'єкт переваг знайдено, то усі наявні ваги – як для

їжі (food_weights), так і для спорту (sport_weights) – модифікуються з урахуванням оцінки користувача (score). Це дозволяє реалізувати динамічне навчання системи, що підлаштовується під зміни вподобань у реальному часі.

```python
@app.route( rule: "/api/rate", methods=["POST"])  new *
def rate_items():
    if "user_id" not in session:
        return jsonify({"error": "Unauthorized"}), 403

    data = request.get_json()
    item = data.get("item")
    score = float(data.get("score", 0))
    user_id = session["user_id"]

    pref = Preference.query.filter_by(user_id=user_id).first()
    if not pref:
        return jsonify({"error": "No preferences found"}), 404

    if pref.food_weights:
        for food_item in pref.food_weights:
            pref.food_weights[food_item] += score

    if pref.sport_weights:
        for sport_item in pref.sport_weights:
            pref.sport_weights[sport_item] += score

    db.session.commit()
    return jsonify({"status": "ok"})
```

Рисунок 4.3 – Метод оцінки відповіді моделі

Застосунок працює у вигляді багаторівневої архітектури, де агентна система приймає запит, визначає його категорію та звертається до відповідного модуля для отримання відповіді. Всі взаємодії логуються для подальшого покращення рекомендацій.

ВИСНОВКИ

У результаті розробки даного сервісу буде створено сучасний веб-додаток, який забезпечує користувачам можливість формування персоналізованого плану дня, враховуючи їхні вподобання щодо фізичної активності, харчування та музичного супроводу.

Завдяки використанню бази даних, що містить інформацію про різні види тренувань, страви з розрахунком їхньої калорійності та співвідношенням білків, жирів і вуглеводів, а також музичні композиції, класифіковані за жанрами, ритмом і настроєм, система зможе підбирати найбільш релевантні рекомендації відповідно до індивідуальних потреб кожного користувача.

Фундаментальною особливістю розробленого додатку є його адаптивність. Користувачі отримують можливість не лише створювати базовий план дня на основі своїх початкових налаштувань, але й впливати на подальші рекомендації завдяки інтегрованій системі оцінювання. Це дозволяє штучному інтелекту постійно вдосконалюватися, аналізуючи вподобання та виключаючи нерелевантні пропозиції.

Впровадження нейронної мережі дає змогу забезпечити поступове навчання системи та її адаптацію під конкретного користувача, що сприяє підвищенню ефективності рекомендацій і зручності користування сервісом.

Одним із ключових аспектів розробки є можливість користувачів експериментувати з новими варіантами активностей, харчування та музики. Реалізація функції випадкових рекомендацій дозволяє отримувати новий досвід та розширювати спектр уподобань, що робить використання додатку більш захопливим і різноманітним.

Таким чином, результатом виконаної роботи стане веб-додаток, що поєднує в собі передові технології аналізу даних, машинного навчання та веб-розробки для створення інтуїтивно зрозумілого та корисного інструменту.

Сервіс не лише полегшує процес планування дня, а й надає можливість покращувати якість життя користувача шляхом персоналізованого підходу до вибору тренувань, харчування та музичного супроводу. Інтерактивність, адаптивність та штучний інтелект у цьому додатку забезпечують його унікальність та високу конкурентоспроможність серед аналогічних рішень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Kaggle gym exercises. URL: *https://www.kaggle.com/datasets/ambarishdeb/gym-exercises-dataset* (date of access: 26.03.2025).

2. Kaggle gym dataset. URL: *https://www.kaggle.com/datasets/edoardoba/fitness-exercises-with-animations* (date of access: 26.03.2025).

3. Contributors to Wikimedia projects. Neural network (machine learning) – Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Neural_network_(machine_learning) (date of access: 26.03.2025).

4. Hu C., Downie A. What is Speech To Text? | IBM. *IBM – United States*. URL: https://www.ibm.com/think/topics/speech-to-text (date of access: 26.03.2025).

5. Videos for gym. URL: *https://www.muscleandstrength.com/exercises* (date of access: 26.03.2025).

6. Contributors to Wikimedia projects. Biomechanics – Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Biomechanics (date of access: 26.03.2025).

7. Biomechanics – Latest research and news | Nature. *Nature*. URL: https://www.nature.com/subjects/biomechanics (date of access: 26.03.2025).

8. Нейронні мережі – Wiki ТНТУ. *Wiki ТНТУ*. URL: https://wiki.tntu.edu.ua/Нейронні_мережі (дата звернення: 19.05.2025).

9. Ahmad H. Best Practices for Training Deep Neural Networks in Deep Learning. *Medium*. URL: https://medium.com/swlh/best-practices-for-training-deep-neural-networks-in-deep-learning-8c69b6f8b140 (date of access: 19.05.2025).

10. Contributors to Wikimedia projects. Deep learning – Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Deep_learning (date of access: 19.05.2025).

11. Explained: Neural networks. *MIT News | Massachusetts Institute of Technology*. URL: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414 (date of access: 19.05.2025).

12. Galante G. A practical guide to content-based recommender systems. *Medium*. URL: https://medium.com/@guillaumegalante/a-practical-guide-to-content-based-recommender-systems-865708a8595d (date of access: 19.05.2025).

13. GeeksforGeeks. TensorFlow Recommenders – GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/tensorflow-recommenders/ (date of access: 19.05.2025).

14. GitHub – xei/recommender-system-tutorial: A step-by-step tutorial on developing a practical recommendation system (retrieval and ranking) using TensorFlow Recommenders and Keras. *GitHub*. URL: https://github.com/xei/recommender-system-tutorial (date of access: 19.05.2025).

15. Good Practices for Neural Network – AgrEGG. *AgrEGG*. URL: https://www.agregg.cloud/good-practices-for-neural-network/ (date of access: 19.05.2025).

16. Greyling C. The Evolution of AI Agents & Agentic Systems. *Medium*. URL: https://cobusgreyling.medium.com/the-evolution-of-ai-agents-agentic-systems-92259a5f5e22 (date of access: 19.05.2025).

17. Lobo D. The evolution of AI Agents. *Collect your global payments at 0% Forex Markup | Winvesta*. URL: https://www.winvesta.in/blog/agentic-ai/the-evolution-of-ai-agents (date of access: 19.05.2025).

18. Po L. Optimizing Neural Networks: Key Techniques to Boost Performance and Generalization. *Medium*. URL: