МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: Розробка чат-бота з використанням Generative AI Service на платформі Orcale Cloud Infrastructure

Здобувача вищої освіти першого (бакалаврського) рівня,
IV курсу, групи КБ-21-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології (за галузями)»

_____ Ботвинка Олександра Олеговича
(підпис) (прізвище, ім'я, по батькові)

Науковий керівник _____ Редич Олександр Володимирович, к.е.н., доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-професійної програми _____ Омельчук Антон Анатолійович, к.т.н.
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних технологій і систем протокол від «13» травня 2025 № 18

Завідувач кафедри _____ Філоненко Михайло Миколайович, к.ф.-м.н., доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь – 2025

Реєстрація № 45 23.05.2025 _____
(дата) (підпис відповідальної особи)

Результати перевірки До захисту 23.05.2025 _____
(до захисту, на доопрацювання) (дата) (підпис)

Результат захисту 5 (відмінно) 04.06.2025 _____
(оцінка) (дата) (підпис)

Голова комісії: _____ В.М. Франчук
(підпис)

Члени комісії: _____ М.М. Філоненко
(підпис)

_____ С.М. Грищенко
(підпис)

_____ А.А. Омельчук
(підпис)

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії,секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	29		Редич О.В. Франчук В.М. Філоненко М.М. Грищенко С.М. Омельчук А.А. Суховецька Л.П.
2. Дотримання вимог щодо оформлення	від 0 до 10	9		
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	5		
4. Рецензент	від 0 до 5	5		
РАЗОМ БАЛІВ	від 0 до 50	48		

Оцінювання кваліфікаційної роботи науковим керівником – 48 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалоюЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
96	A	5 (відмінно)	04.06.2025	Франчук В.М. Філоненко М.М. Грищенко С.М. Омельчук А.А. Суховецька Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань 12 «Інформаційні технології»
Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Інформаційні управляючі системи і технології (за галузями)»

Затверджую
Завідувач кафедри _____ Філоненко М.М.
«15» листопада 2024

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Ботвинко Олександру Олеговичу

1. Тема кваліфікаційної роботи ***«Розробка чат-бота з використанням Generative AI Service на платформі Oracle Cloud Infrastructure»***
Затверджена наказом ректора від «13»листопада 2024 № 1717
2. Строк здачі закінченої роботи *«5» травня 2025*
3. Мета роботи: *розробка чат-бота з використанням Generative AI Service на платформі Oracle Cloud Infrastructure.*

Об'єкт дослідження: *технології Generative AI.*
Предмет дослідження: *сервіси на платформі Oracle Cloud Infrastructure.*
Орієнтовна база дослідження: *відкриті дані про криптовалюти, блокчейну, та децентралізовані фінанси.*

4. Вхідні дані до роботи: *документація Oracle Cloud Infrastructure, інструменти та технології Generative AI.*
5. Перелік графічного матеріалу: *рисунки інтерфейсів додатку, структурні схеми, алгоритми, лістинги програм.*
6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ АСПЕКТИ GENERATIVE AI;
1.1. Поняття технології та історія Generative AI
1.2. Огляд інструментів Generative AI
1.3. Порівняльний аналіз інструментів генерації: OpenAI, Cohere, Anthropic,Google
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ЧАТУ
2.1. Побудова RAG-архітектури: pipeline з embedding, retrieval і LLM
2.2. Взаємодія елементів через API Cohere та Oracle DB;
2.3. Підготовка та векторизація бази знань для чат-боту
РОЗДІЛ 3. РОЗРОБКА ВЕБ-ІНТЕРФЕЙСУ СИСТЕМИ
3.1. Фреймворк Streamlit для швидкої реалізації користувацького інтерфейсу в браузері
3.2. Тестування відповідей чат-бота та оцінка ефективності впровадження на безкоштовному рівні OCI;

3.3. Перспективи розширення функціональності чат-бота та розрахунок економічного ефекту розробки;

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційноїроботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	до 01.11.2024	«01» 11 2024 р.
2	**І розділ** (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	30.01.2025	«30» 01 2025 р.
3	**ІІ розділ** (аналіз експериментальної частини про методики досліджень)	28.02.2025	«28» 02 2025 р.
4	**ІІІ розділ** (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	30.04.2025	«30» 04 2025 р.
5	**Висновки**	05.05.2025	«05» 05 2025 р.
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	25.05.2025	«25» 05 2025 р.
7	Отримання відгуку керівника	25.05.2025	«25» 05 2025 р.
8	Отримання зовнішньої рецензії	28.05.2025	«28» 05 2025 р.

8. Дата видачі завдання «15» листопада 2024 р.

9. Керівник кваліфікаційної роботи _____ Редич О.В.
(підпис)

10. Керівник освітньо-професійної програми _____ Омельчук А.А.
(підпис)

11. Завдання до виконання отримав, здобувач вищої освіти _____ Ботвинко О.О.
(підпис)

АНОТАЦІЯ

Розробка чат-бота з використанням Generative AI Service на платформі Oracle Cloud Infrastructure.

Мета роботи полягає в розробці чат-бота з використанням Generative AI Service на платформі Oracle Cloud Infrastructure.

Завданнями роботи є:

1. Аналіз літератури та документації предметної області.
2. Дослідження можливостей моделей обробки природної мови та архітектури RAG.
3. Розробка архітектури, бази знань та інтерфейсу системи.
4. Реалізація прототипу генеративного чат-боту.
5. Тестування розробленої системи.

Предметом дослідження є сервіси на платформі Oracle Cloud Infrastructure.

Об'єктом дослідження є технології Generative AI.

Базою дослідження виступає відкриті дані про криптовалюти, блокчейн та децентралізовані фінанси.

За результатами дослідження розроблено прототип генеративного чат-бота на архітектурі генерації з доповненням через пошук.

Одержані результати можуть бути використані для надання інформації про криптовалюти, блокчейн та децентралізовані фінанси та створення адаптивного, персоналізованого чат-бота.

Кваліфікаційна робота містить 63 сторінок, 11 рисунків, список використаних джерел містить 41 найменувань та 4 додатків.

Ключові слова: штучний інтелект, генеративний штучний інтелект, великі мовні моделі, база знань, векторна база даних, чат-бот, моделі вбудовування.

Рік виконання кваліфікаційної роботи: 2025

Рік захисту роботи: 2025

ABSTRACT

Development of a chatbot using Generative AI Service on Oracle Cloud Infrastructure platform.

The aim of this work is to develop a chatbot using Generative AI Service on Oracle Cloud Infrastructure platform.

The tasks of the work are as follows:

1. Analysis of literature and documentation in the subject area.
2. Investigation of the capabilities of natural language processing models and RAG architecture.
3. Development of system architecture, knowledge base, and interface.
4. Implementation of a generative chatbot prototype.
5. Testing of the developed system.

The subject of the research is Oracle Cloud Infrastructure platform services.

The object of the research is Generative AI technologies.

The research is based on open data about cryptocurrencies, blockchain, and decentralized finance.

The results of the research include the development of a generative chatbot prototype based on Retrieval-Augmented Generation architecture.

The obtained results can be used to provide information about cryptocurrencies, blockchain, and decentralized finance and develop adaptive, personalized chatbots.

The qualification work consists of 63 pages, 11 images, conclusions, the list of 41 references and 4 appendices.

Keywords: artificial intelligence, generative artificial intelligence, large language models, knowledge base, vector database, chatbot, embedding models.

Year of qualification work: 2025

Year of work defense: 2025

ЗМІСТ

ВСТУП 3
РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ АСПЕКТИ РОЗВИТКУ GENERATIVE AI 6
1.1. Поняття технології Generative AI та огляд етапів його розвитку 6
1.2. Огляд інструментів Generative AI на базі Oracle 11
1.3. Порівняльний аналіз інструментів генерації: OpenAI, Cohere, Anthropic, Google 15
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ЧАТУ 20
2.1. Побудова RAG-архітектури: pipeline з embedding, retrieval і LLM 20
2.2. Взаємодія елементів через API Cohere та Oracle DB за допомогою фреймворку LangChain 25
2.3. Підготовка та векторизація бази знань для чат-боту 29
РОЗДІЛ 3. РОЗРОБКА ВЕБ-ІНТЕРФЕЙСУ СИСТЕМИ 33
3.1. Фреймворк Streamlit для швидкої реалізації користувацького інтерфейсу в браузері 33
3.2. Тестування відповідей чат-бота та оцінка ефективності впровадження на безкоштовному рівні OCI 37
3.3. Перспективи розширення функціональності чат-бота та розрахунок економічного ефекту розробки 41
ВИСНОВКИ 47
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 49
ДОДАТКИ 52

ВСТУП

У сучасному світі стрімкий розвиток штучного інтелекту, хмарних технологій та обчислювальних платформ відкриває нові можливості для створення інтелектуальних інформаційних систем, здатних ефективно обробляти великі обсяги даних та надавати користувачам релевантну інформацію в режимі реального часу. Одним із перспективних напрямів є використання генеративних моделей штучного інтелекту, які здатні не лише аналізувати дані, а й генерувати зв'язну, логічну і контекстуально релевантну інформацію. Особливу увагу привертають інноваційні розробки в галузі нейронних мереж та глибинного навчання, що значно покращують здатність моделей до розуміння контексту та генерації релевантних відповідей [38]. Водночас важливим аспектом розвитку мовних моделей є підвищення їхньої обчислювальної ефективності та оптимізації без втрати якості генерації, що демонструють дослідження в цій галузі [14]. Також даною проблематикою займалися вітчизняні науковці та розробників, такі як Майстренко С. Я., Литвин В.В., Шевченко А.І., Винничук С.Д., Бісікало О.В, а також зарубіжні: Turing A.M., McCarthy J., Minsky M.L., Schmidhuber J., Sutskever I., Brown T.B., Thrun S., Vaswani A., Devlin J. та Caldarini G.

Хмарна платформа Oracle Cloud Infrastructure пропонує сучасний набір інструментів для реалізації систем на базі Generative AI, включаючи попередньо навчені моделі, API для обробки природної мови, а також можливості інтеграції з базами даних та інтерфейсами користувача. Інтеграція цих технологій дозволяє створити чат-бота нового типу − не просто як скриптову систему запит–відповідь, а як інтелектуального помічника, який може адаптуватися до контексту, навчатися з даних та надавати гнучкі інтерпретації користувацьких запитів.

Метою кваліфікаційної роботи є розробка чат-бота з використанням Generative AI Service на платформі Oracle Cloud Infrastructure.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1) здійснити аналіз наукових публікацій та документації з тематики Generative AI і сервісів Oracle Cloud Infrastructure;

2) дослідити архітектуру та функціональні можливості Generative AI Service, зокрема моделі, що підтримують обробку природної мови;

3) проаналізувати відкриті джерела даних з тематики криптовалют, блокчейну та децентралізованих фінансів, які будуть використані як база знань;

4) розробити архітектуру чат-бота із використанням сервісів Oracle та великими мовними моделями Cohere.

5) реалізувати прототип чат-бота, який виконує пошук релевантної інформації за допомогою векторного представлення текстів;

6) провести тестування функціональності чат-бота на реальних запитах та оцінити ефективність роботи системи;

7) надати рекомендації щодо розширення функціональності чат-бота та оцінити можливий економічний ефект від його впровадження.

Об’єктом дослідження є технології Generative AI.

Предметом дослідження є сервіси на платформі Oracle Cloud Infrastructure.

Орієнтовною базою дослідження є відкриті дані про криптовалюти, блокчейн та децентралізовані фінанси, які використані для побудови системи питань і відповідей на базі векторного пошуку.

Для реалізації поставленої мети в роботі використано комплекс методів дослідження:

1) аналіз наукової літератури для формування теоретичної основи дослідження;

2) системний аналіз функціональних можливостей Generative AI Service;

3) метод моделювання для побудови архітектури чат-бота;

4) прототипування для реалізації та перевірки ефективності запропонованого рішення;

5) експериментальне тестування та оцінка якості надання відповідей.

Практичне значення роботи полягає в розробці чат-бота, який забезпечує інтелектуальний доступ до вибраних даних із використанням Generative AI Service та базами даних Oracle. Систему можна адаптувати до різних предметних областей. Такий чат-бот може використовуватися як засіб комунікації з користувачами, платформа для навчання персоналу або інструмент для швидкого доступу до актуальної інформації, яку легко оновлювати та розширювати.

Новизна дослідження полягає у побудові RAG-системи з використанням Generative AI Service на базі OCI, що поєднує генеративні моделі, векторний пошук та інтеграцію з базою даних Oracle як сховищем embedding векторів. Розроблене рішення забезпечує ефективну взаємодію з користувачем у форматі чат-бота та відповідає вимогам сучасного інформаційного середовища.

Результати дослідження апробовано на II Міжнародній науково-практичній конференції молодих вчених «Фінансові механізми забезпечення відновлення економіки України в сучасних умовах» (м. Ірпінь 26 лютого 2025 р.).

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ АСПЕКТИ РОЗВИТКУ GENERATIVE AI

1.1. Поняття технології Generative AI та огляд етапів його розвитку

Сучасний стрімкий розвиток штучного інтелекту базується на наукових працях Алана Тюрінга, Марвіна Мінського, Джона Маккарті, Герберта Саймона та інших дослідників, а також на розроблених ними методологіях і алгоритмах у XX столітті. У той час розвиток штучного інтелекту було обмежено недостатньою обчислювальною потужністю, малим обсягом пам'яті та високою вартістю апаратного забезпечення. Натомість сьогодні ці бар'єри практично подолані завдяки еволюції процесорних технологій, машинного і глибокого навчання, розвитку хмарних обчислень та доступності великих обсягів даних. Сукупність цих факторів, а також успішні приклади застосування генеративних моделей, зокрема компанії OpenAI, створили сприятливі умови для відродження і стрімкого зростання досліджень у сфері штучного інтелекту.

Щоб краще зрозуміти сучасний стан розвитку штучного інтелекту, варто розглянути основні етапи його становлення від середини XX століття до сьогодення. Одним із перших дослідників, хто теоретично обґрунтував можливість створення мислячих машин, став британський математик Алан Тюрінг. У своїй концепції обчислювальної машини він припустив, що машинна зможе вийти за межі початкового програмування та розширити свої початкові функції. Попри технічні обмеження того часу, ідеї Тюрінга заклали основу для подальшого розвитку штучного інтелекту. Він також запропонував метод перевірки здатності машини до мислення – імітаційну гру, що нині відома як тест Тюрінга. Значним кроком у формуванні галузі став Літній семінар у Дартмутському коледжі у 1956 році, ініційований американським вченим Джоном Маккарті. Саме тут вперше було офіційно озвучено припущення, що будь-який аспект навчання або інтелекту можна змоделювати настільки точно, що машину

можна буде змусити імітувати людські розумові процеси. Саме під час підготовки до цього семінару Маккарті запропонував термін «штучний інтелект», який став фундаментом для подальших досліджень і розвитку цілої галузі. Також 1959 році він опублікував статтю "Програми зі здоровим глуздом", заклавши основи для подальшої ідеї загального штучного інтелекту (AGI) [35].

Після Дартмутської конференції з’явилися перші практичні приклади застосування технологій штучного інтелекту, серед яких реалістичний чат-бот та автономний робот. Одним із таких проривів стала програма ELIZA, створена Джозефом Вайзенбаумом у Массачусетському технологічному інституті у 1966 році. ELIZA стала першим чат-ботом, здатним імітувати розмову із психологом, перепрофільовуючи відповіді користувача у вигляді запитань для продовження діалогу. Цікаво, попри свою простоту, багато користувачів були впевнені, що спілкуються з реальною людиною. Наступним важливим досягненням того часу став мобільний робот Shakey, розроблений у 1966–1972 роках у Стенфордському дослідницькому інституті. Оснащений телекамерою та датчиками, Shakey міг самостійно орієнтуватися у складних середовищах. Хоча його можливості за сучасними мірками були обмеженими, проєкт заклав основи розвитку таких напрямів, як комп’ютерний зір, побудова маршрутів і маніпуляція об'єктами у просторі.

Попри всі ці досягнення, у 1970-х роках галузь зіткнулася з кризою, яку назвали «зимою штучного інтелекту». Основною причиною стало те, що науковці та інженери переоцінили можливості тодішніх комп’ютерів і зосередилися у бік символьного ШІ та експертних систем. Дані системи базувалися на створенні баз знань за визначеними практичними галузями із заздалегідь прописаними правилами логічного виведення. Після критичної доповіді британського математика Джеймса Лайтгілла 1973 року фінансування наукових досліджень у сфері ШІ значно скоротилося, оскільки уряди та компанії розчарувалися у повільному прогресі. Утім, «зима ШІ» підштовхнула дослідників до глибшого

вивчення фундаментальних проблем і розроблення нових підходів, як-от машинне навчання та еволюційні алгоритми. У результаті наприкінці 1990-х років галузь відродилась [3].

Відродження інтересу до штучного інтелекту наприкінці 1990-х років ознаменувалося появою нових підходів до обробки даних, що суттєво розширило можливості інтелектуальних систем. У цей період особливо активного розвитку набули обробка природної мови (NLP) і комп'ютерне бачення, що стало початком сучасної епохи в дослідженнях ШІ. У 1980-х роках експертні системи все ще були дуже обмеженими, оскільки працювали переважно зі структурованими даними та базувалися на фіксованих правилах. У 1990-х роках завдяки стрімкому розвитку машинного навчання дослідники почали відмовлятися від правил і перейшли до використання статистичних методів, які дозволяли моделювати закономірності безпосередньо з даних. У сфері обробки природної мови ключовим проривом стало впровадження прихованих марковських моделей (ПММ), які забезпечили ймовірнісне моделювання текстів. Це призвело до значного прогресу в розпізнаванні мовлення, машинному перекладі та автоматичній класифікації текстів. У галузі комп’ютерного бачення важливим досягненням стала поява згорткових нейронних мереж (CNN), що підвищили ефективність розпізнавання об'єктів на зображеннях. Поява цих технологій дозволила штучному інтелекту вперше ефективно працювати з неструктурованими даними, що істотно розширило межі його застосування – від автономних транспортних засобів до систем медичної візуалізації [1].

Період із 1990-х до сьогодення є часом стрімкого розвитку машинного навчання та глибинного навчання, що докорінно змінило практику досліджень і розробок у сфері штучного інтелекту. Вибухове зростання обчислювальної потужності, поява великих обсягів даних і вдосконалення алгоритмів забезпечили небачене раніше прискорення інновацій у науці, бізнесі та техніці. Одним із визначальних проривів стала поява у 2009 році проєкту ImageNet, який надав

дослідникам мільйони анотованих зображень для навчання глибоких нейронних мереж і закріпив легітимність попереднього навчання великих моделей. Надалі розвиток глибоких нейронних мереж призвів до суттєвих досягнень у комп'ютерному баченні, обробці природної мови та робототехніці. Ключовим етапом стало створення архітектури Transformer у 2017 році, розробленої дослідниками Google, яка здійснила справжню революцію в обробці природної мови завдяки використанню механізму самоуваги (self-attention). Завдяки цьому підходу великі мовні моделі (LLM) отримали здатність глибоко розуміти контекст текстів і працювати із величезними обсягами даних. У 1990-х роках парадигма символічного ШІ поступово поступилася місцем конекціонізму, що базується на моделюванні обчислювальних процесів у великих мережах простих вузлів, подібних до нейронних мереж. Хоча ідеї конекціонізму були закладені ще в 1950-х роках Розенблаттом, їхнє широке практичне застосування стало можливим лише завдяки розвитку глибинного навчання. Сьогодні основні досягнення в штучному інтелекті пов'язані саме з розвитком великих мовних моделей та генеративного штучного інтелекту, які поєднують конекціоністські підходи, машинне навчання та обробку великих даних [17].

Після десятиліть розвитку штучного інтелекту, сучасний етап у цій еволюції позначений появою генеративного штучного інтелекту (Generative AI) – технології, яка дозволяє не лише аналізувати дані та надавати передбачення, а й створювати новий контент. На відміну від традиційних алгоритмів, що зосереджуються на класифікації чи прогнозуванні, генеративні моделі навчаються на великих масивах інформації, виявляючи приховані закономірності, щоб самостійно генерувати тексти, зображення, аудіо або відео, яких раніше не існувало. Особливість генеративного штучного інтелекту полягає у здатності самонавчання, де моделі автоматично формують уявлення про дані та вчаться створювати змістовний, логічно зв'язаний вихід. Цей напрям став визначальним трендом останніх років, зокрема завдяки стрімкому розвитку великих мовних

моделей, таких як GPT від OpenAI, Command від Cohere Gemini від Google та інших. У результаті генеративний ШІ відкрив нову еру в інтелектуальних системах, де машини виступають співавторами інновацій у різних галузях. В його основі лежать низка фундаментальних технологій (рис. 1.1) [4].

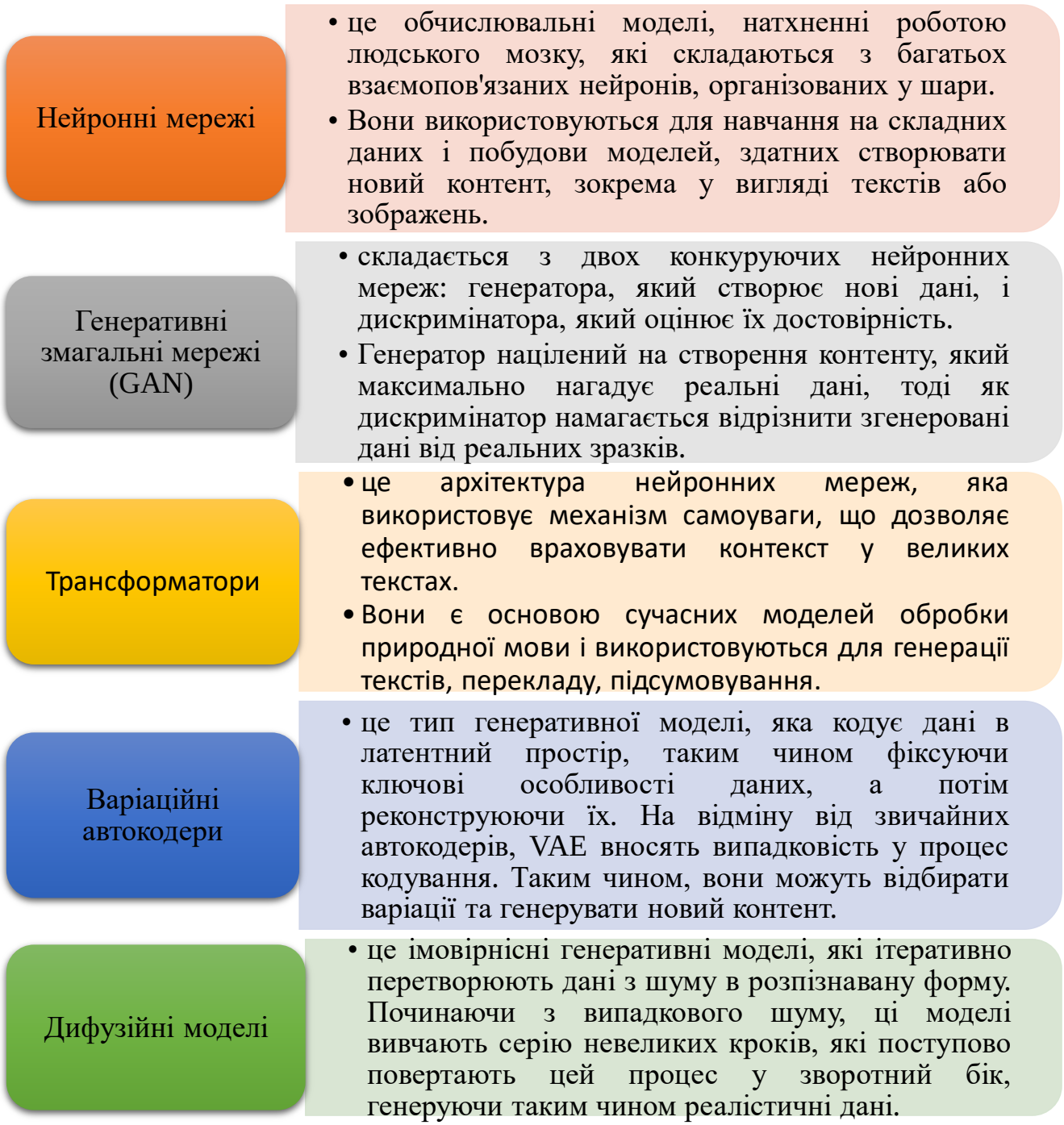


Рис. 1.1. Основні технології з яких складається Generative AI

Джерело: [6].

У підсумку, розвиток штучного інтелекту пройшов складний і водночас вражаючий шлях від теоретичних основ Алана Тюрінга до сучасних генеративних моделей, здатних створювати новий зміст у текстовій, візуальній чи аудіоформі. Це стало можливим завдяки поєднанню нейронних мереж, машинного та глибокого навчання, нових архітектур, появі потужних хмарних інфраструктур і відкритих даних. Генеративний ШІ сьогодні виступає як кульмінація багатьох десятиліть наукових пошуків, відкриваючи нові горизонти у сфері креативності, аналітики та автоматизації.

1.2. Огляд інструментів Generative AI на базі Oracle

Компанія Oracle є одним із світових лідерів у сфері корпоративних інформаційних технологій, зокрема в галузях керування базами даних, хмарних обчислень і аналітики. З моменту створення Oracle постійно впроваджує інновації, реагуючи на динамічні зміни IT-ринку та зростаючі потреби бізнесу. У контексті стрімкого розвитку штучного інтелекту, а особливо генеративного ШІ, компанія активно розвиває відповідні сервіси, інтегруючи їх у свою хмарну екосистему. Використовуючи багаторічний досвід у сфері зберігання, обробки та аналізу даних, Oracle вже забезпечує основу для розгортання сервісів на основі штучного інтелекту [24].

У 2023 році Oracle оголосила про стратегічну співпрацю з компаніями Cohere та Meta, одними із провідних розробників великих мовних моделей, з метою інтеграції генеративного штучного інтелекту в сервіси Oracle Cloud Infrastructure (OCI). Завдяки цій колаборації Oracle надала користувачам доступ до масштабованих та оптимізованих LLM-моделей, а також моделей для створення embedding-представлень тексту, безпосередньо в межах власного хмарного середовища. Такі моделі дозволяють конвертувати текстові дані у векторну форму для пошуку схожих документів, класифікації та побудови RAG-систем. Моделі спеціалізуються на обробці природної мови, аналітиці та генерації текстів, стали

основою для створення сервісу OCI Generative AI, який підтримує мультиінстанційність, ізоляцію даних і високий рівень безпеки – критично важливі аспекти для корпоративного використання.

У структурі Oracle Cloud Infrastructure аналітика та штучний інтелект представлені широким спектром сервісів, згрупованих у категорію Analytics & AI. Користувачі мають змогу працювати з різними інструментами для побудови моделей машинного навчання, обробки великих обсягів даних, потокового аналізу, а також безпосередньо взаємодіяти з сервісами генеративного ШІ та налаштовувати їх для себе. На зображенні нижче представлено інтерфейс OCI з доступними модулями, які охоплюють більшість інструментів для роботи з AI, тобто блоки: машинного навчання, аналітики, штучного інтелекту та озера даних, де зберігаються зібрані неструктуровані дані (рис. 1.2).

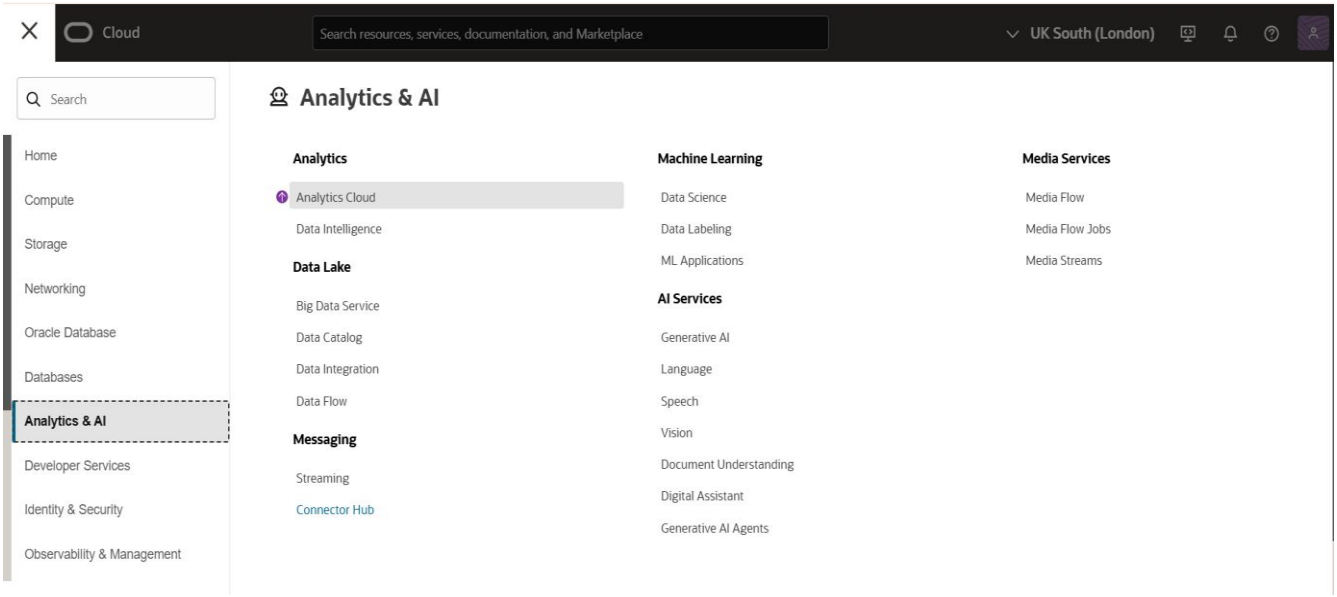


Рис. 1.2. Інтерфейс розділу Analytics & AI в OCI з переліком доступних сервісів

Варто зазначити, що платформа підтримує модель доступу до сервісів за трьома принципами Free Tier, Always Free та платну версію. Версія Free Tier надає можливість новим користувачам ознайомитися можливостями всього сервісу, проте даний доступ обмежений у часі 30 днів та доступ до платних

сервісів виконується за допомогою безкоштовних 300$ хмарних кредитів. Після закінчення терміну у 30 днів форма доступу змінюється на Always Free, а всі дані та сервіси видаляються, попередження про дану подію відправляють на електронну адресу. Тому щоб поновити повноцінний доступ до генеративного ШІ та пов'язаних з ним сервісів (наприклад, Generative AI, Vision, Language, Speech) можливий лише у рамках платного тарифного плану, оскільки ці сервіси потребують значних обчислювальних ресурсів та підтримки API-запитів на комерційному рівні. Таким чином, практична реалізація повноцінних ШІ-рішень передбачає перехід до платного рівня підписки після ознайомчого періоду або використання прямих API ключів від Cohere або Meta [10].

Для зручності розуміння нижче наведено короткий огляд ключових сервісів розділу AI Services, доступних у середовищі OCI. Таблиця включає стислий опис призначення кожного з них для оцінки можливостей Oracle (табл. 1.1).

Таблиця 1.1

Огляд основних AI-сервісів Oracle Cloud Infrastructure

Сервіс	Призначення
Generative AI	Сервіс дозволяє розробникам створювати, розгортати та тестувати великі мовні моделі, що здатні генерувати текстовий контент на основі заданого запиту. Він включає попередньо навчені моделі та підтримує налаштування під специфічні бізнес-завдання, наприклад, створення чат-ботів, резюмування текстів або автоматизацію відповіді на запити.
Language	Дозволяє виконувати глибокий аналіз неструктурованого тексту, включаючи визначення настроїв, виявлення ключових фраз, розпізнавання іменованих сутностей (NER) та автоматичну класифікацію тексту. Підтримуються як вбудовані моделі, так і навчання власних моделей за допомогою власних датасетів, що особливо корисно для спеціалізованих предметних галузей. Також передбачена можливість машинного перекладу на 21 мову.
Speech	Використовується для транскрипції дзвінків служби підтримки клієнтів, автоматичного субтитрування та генерації метаданих для медіа-активів для створення архіву з можливістю повного пошуку.

Продовження табл. 1.1

Сервіс	Призначення
Vision	Надає інструменти комп'ютерного зору для розпізнавання та класифікації об'єктів на зображеннях, виявлення облич, а також зчитування тексту з графічних документів. Сервіс підходить як для використання вбудованих моделей, так і для навчання власних.
Document Understanding	Призначений для автоматизованої обробки документів. Дозволяє витягувати текст, структури (наприклад, таблиці) та інші важливі дані з PDF, сканованих файлів або зображень документів.
Digital Assistant	Платформа для створення інтелектуальних чат-ботів, які можуть взаємодіяти з користувачами за допомогою природної мови. Вона підтримує інтеграцію з службами Oracle і сторонніми API.
Generative AI Agents	Сучасний сервіс для побудови інтелектуальних агентів, що працюють на основі генеративного ШІ. Ці агенти можуть підключатись до баз даних, здійснювати пошук (RAG-підхід), генерувати SQL-запити, опрацьовувати введення користувача й взаємодіяти з веб-сервісами. Вони здатні автоматизувати аналітику, технічну підтримку й взаємодію з корпоративними знаннями.

Сформовано автором на основі [16,26,27].

Отже, Oracle Cloud Infrastructure пропонує широкий спектр інструментів для реалізації систем на базі генеративного штучного інтелекту, які охоплюють обробку тексту, зображень, мови та документів. Інтеграція цих сервісів уніфікована в межах екосистеми OCI, що значно спрощує побудову, розгортання та масштабування додатків. Особливу увагу варто приділити перевагам Oracle, серед яких – висока надійність хмарної інфраструктури, гнучкість налаштувань, високий рівень безпеки за допомогою двоетапної перевірки, а також повна сумісність з корпоративними продуктами Oracle. Окремою перевагою є можливість тісної інтеграції AI-сервісів з платформою Oracle APEX, яка дозволяє швидко створювати веб-застосунки з мінімальним програмуванням та управліннями базами даних. Завдяки цьому навіть користувачі без глибоких технічних знань можуть реалізувати інтерфейси для взаємодії з генеративними

моделями, наприклад, створити власного чат-бота або панель аналітики з доповненою генерацією відповіді. Така гнучкість і доступність робить Oracle одним із лідерів серед платформ для побудови генеративних систем на рівні підприємств.

1.3. Порівняльний аналіз інструментів генерації: OpenAI, Cohere, Anthropic, Google

Ринок штучного інтелекту активно розвивається, і на передовій цього руху перебувають кілька ключових компаній, які визначають напрям і темпи інновацій у сфері. OpenAI, Cohere, Anthropic та Google пропонують різні підходи до побудови та використання генеративних моделей, орієнтуючись на широке коло сценаріїв: від створення тексту й аналізу природної мови до генерації зображень і коду. Хоча всі ці системи базуються на архітектурах трансформерів, вони відрізняються рівнем доступності, функціональністю, швидкістю, масштабованістю та підходами до етики. Порівняння їхніх можливостей дозволить краще оцінити, яка з платформ найбільше підходить для дослідницьких цілей, інтеграції в корпоративні рішення або реалізації прикладних проєктів на зразок чат-ботів, інтелектуальних агентів чи сервісів підтримки.

OpenAI − це компанія з досліджень штучного інтелекту, заснована у 2015 році з метою розвитку ШІ, який приносить користь всьому людству. Спочатку створена як некомерційна організація, вона була профінансована інвесторами, які зібрали $1 мільярд для підтримки відкритих досліджень і ресурсів. З часом OpenAI трансформувалася у комерційну структуру, зберігаючи при цьому фокус на етичному розвитку та доступності ШІ. Ключовим продуктом компанії є ChatGPT − популярний чат-бот загального спрямування, здатний генерувати текстові, графічні відповіді та код на запити користувачів. Проте найбільш ефективно його використовувати як розмовний чат-бот, тому що при генерації зображень і коду виникають труднощі з розпізнаванням та створенням зображень,

запам’ятовуванням та генерацією коду. Ці інструменти демонструють широкий спектр можливостей генеративного ШІ, від обробки природної мови до генерації візуального контенту. Для оптимального вирішення специфічних завдань доцільно використовувати спеціалізовані продукти компанії. Як приклад можна навести DALL-E – генеративну модель від OpenAI, яка спеціалізується на створенні високоякісних зображень на основі текстових запитів [40].

Компанія Cohere є одним із розробників великих мовних моделей, що спеціалізується на створенні високоякісних рішень у сфері обробки природної мови для корпоративного використання. Заснована у 2019 році, вона зосереджує свою діяльність на розробці генеративних моделей, моделей семантичного пошуку та систем покращення релевантності результатів. Модельні серії: Command, Embed та Rerank охоплюють генерацію тексту, побудову векторних представлень та повторну оцінку релевантності запитів, забезпечуючи багатомовну підтримку й гнучкість налаштування під специфіку окремих бізнес-сценаріїв. Важливу роль у стратегії Cohere відіграє безпека – компанія активно підтримує приватне розгортання своїх моделей та інтеграцію в локальні середовища клієнтів. Однією з ключових ініціатив є платформа North, яка поєднує можливості персоналізованих агентів, інтелектуального пошуку та безпечного керування даними для масштабування роботи з AI у контрольованому середовищі. Важливим стратегічним кроком компанії стала співпраця з Oracle, яка забезпечила глибоку інтеграцію моделей Cohere у хмарні сервіси Oracle Cloud Infrastructure, розширивши можливості використання Generative AI у корпоративному секторі [9].

Anthropic є новою компанією у галузі штучного інтелекту яка зосереджена на розробці безпечних і передбачуваних великих мовних моделей для генеративного ШІ. Заснована у 2021 році колишніми співробітниками OpenAI, компанія позиціонує себе як прихильник етичного й безпечного розвитку штучного інтелекту. Основна увага Anthropic зосереджена на розробці сімейства

моделей Claude, які створені з урахуванням принципів інтерпретованості, прозорості та керованості поведінки AI-систем. Ці моделі призначені для виконання широкого спектру задач, включаючи генерацію текстів, обробку запитів природною мовою, аналіз документів та допомогу в написанні коду, з особливим акцентом на мінімізацію токсичності та некоректних відповідей. Anthropic активно впроваджує метод під назвою Constitutional AI, що дозволяє моделі дотримуватись заданих правил поведінки, наближаючи її до принципів відповідального ШІ. Компанія співпрацює з такими технологічними гігантами, як Google Cloud та Amazon, що свідчить про її стратегічне прагнення до інтеграції у масштабні хмарні екосистеми та впровадження генеративного ШІ у корпоративне середовище [21].

Компанія Google є одним із провідних інноваторів у сфері штучного інтелекту, яка ще з 2011 року активно інвестує в дослідження та впровадження передових моделей машинного навчання. Підрозділи Google Research і DeepMind відіграють ключову роль у розвитку генеративного ШІ, зокрема через розробку архітектури трансформерів, яка стала основою для сучасних LLM. У 2023 році Google представила власну генеративну модель Gemini (раніше Bard), яка поєднує можливості обробки тексту, коду, зображень та іншого мультимодального контенту. Модель Gemini інтегрована з екосистемою Google Workspace і хмарними рішеннями Google Cloud, що забезпечує її доступність для широкого кола користувачів і розробників. Особливістю моделей від Google є висока продуктивність, інтеграція з пошуковими системами та орієнтація на безшовну роботу в корпоративному середовищі. Розвиток таких інструментів, як PaLM 2 і Gemini, свідчить про прагнення Google створити конкурентоспроможні моделі, здатні адаптуватися до різноманітних сценаріїв застосування [8].

Для об’єктивного порівняльного аналізу моделей генеративного ШІ, розроблених компаніями OpenAI, Cohere, Anthropic та Google, було обрано найновіші доступні текстові моделі з безкоштовним доступом. Для актуалізації

даних було використано аналітичні платформи з оцінювання LLM, тобто LMArena та ArtificialAnalysis.ai, які надають ранжування моделей за результатами тестування у відкритих і незалежних сценаріях. У таблиці подано стислу характеристику вибраних моделей (табл. 1.2).

Таблиця 1.2

Порівняльний аналіз продуктивності сучасних LLM за метриками

Назва моделі	GPT-4o (березень 2025)	Command A	Claude 3.7 Sonnet	Gemini 2.0 Flash
Context window	128 тисяч токенів	256 тисяч токенів	200 тисяч токенів	1 мільйон токенів
Artificial analysis intelligence index	50%	40%	48%	48%
MMLU	80%	71%	80%	78%
GPQA Diamond	66%	53%	66%	62%
Livecode Bench	43%	29%	39%	33%
Scicode	37%	28%	38%	31%
HumanEval	96%	82%	95%	90%
Math-500	89%	82%	85%	93%
AIME 2024	33%	10%	22%	33%
Arena score	1408	1305	1301	1354

Сформовано автором на основі [13,23].

У таблиці наведено основні метрики з згаданих вище сайтів, проте варто зазначити що вони не можуть оцінити спеціалізацію моделі, наприклад модель Command A буде краще справлятися з корпоративними рішеннями та задачами ніж інші. Context window визначає обсяг інформації (у токенах), яку модель здатна обробляти одночасно. Artificial Analysis Index – агрегований показник продуктивності моделі за різними тестами. MMLU оцінює здатність до міркувань на основі запитань із загальної освіти. GPQA Diamond перевіряє глибокі знання з природничих наук. Livecode Bench і Scicode тестують навички програмування. HumanEval оцінює точність виконання завдань на Python. Math-500 – складні математичні задачі. AIME 2024 моделює олімпіадні завдання з математики. Arena

Score – загальний рейтинговий бал, що враховує результати моделі у відкритому багатокористувацькому порівняльному середовищі [23].

Оцінка представлених моделей на основі даних таблиці свідчить про високий рівень продуктивності всіх чотирьох LLM-систем. Модель GPT-4o, яка нещодавно оновилась, вирізняється найбільш збалансованим набором характеристик, зокрема максимальною точністю в тестах HumanEval (96%), Math-500 (89%) та високим Arena score (1408). Це підтверджує її ефективність у складних задачах кодування та логіки. Водночас Claude 3.7 Sonnet демонструє співставні показники в задачах природної мови та програмування, зокрема 95% у HumanEval і 85% у Math-500, але поступається в загальному індексі продуктивності.

Gemini 2.0 Flash, попри рекордне контекстне вікно у 1 мільйон токенів, дещо відстає в загальному Arena score (1354), проте стабільно тримається на високому рівні в більшості метрик. Command A демонструє нижчі показники за більшістю метрик порівняно з конкурентами. Проте вона має важливу нішову перевагу. Модель спеціально оптимізована для корпоративного ринку. Її головна сильна сторона - забезпечення максимальної продуктивності при мінімальних витратах на апаратне забезпечення. Це робить Command A привабливим рішенням для різноманітних бізнес-застосувань.

Загалом, порівняння підкреслює домінування GPT-4o як найсильнішої з досліджуваних моделей у 2025 році, при цьому інші моделі демонструють свої унікальні переваги, як-от масштаб контексту чи оптимізація під конкретні задачі та сфери застосування.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ЧАТУ

2.1. Побудова RAG-архітектури: pipeline з embedding, retrieval і LLM

Загальна структура розробленого чат-бота базується на сучасній RAG-архітектурі, яка дозволяє поєднувати можливості генеративного штучного інтелекту з пошуком релевантної інформації у базі знань за допомогою векторизації даних. Дана архітектура забезпечує точнішу та обґрунтовану генерацію відповідей на основі власних, спеціалізованих документів. Чат-бот має модульну побудову, яка включає етапи попередньої обробки запиту, формування векторного простору знань, пошуку релевантного контенту та генерації остаточної відповіді. На рисунку нижче зображено схему архітектури системи, яка ілюструє зв'язки між користувацьким інтерфейсом, RAG-механізмом та векторною базою даних (рис. 2.1).

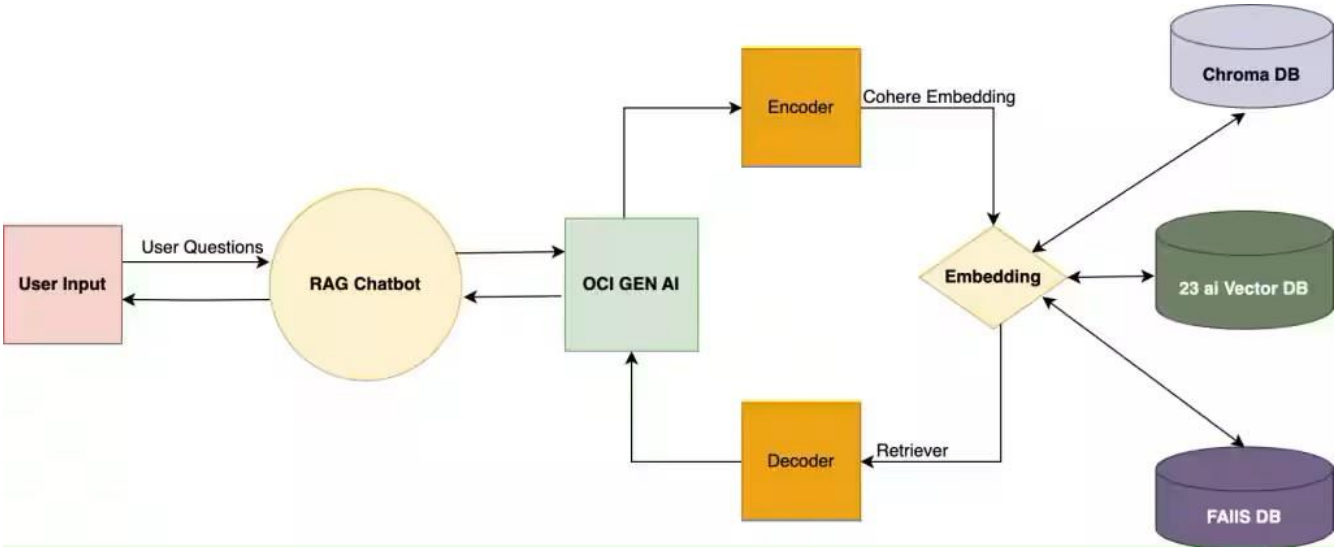


Рис. 2.1. Загальна архітектура ключових елементів чат-бота

RAG (Retrieval-Augmented Generation) – це сучасний підхід до генеративного штучного інтелекту, який поєднує дві ключові технології: пошук релевантної інформації (retrieval) і генерацію тексту (generation). На відміну від класичних LLM, які опираються виключно на попередньо навчені параметри, RAG-моделі мають доступ до зовнішнього векторного сховища, з якого під час

виконання запиту витягується контекстуальна інформація. Такий підхід вирішує проблеми з якими стикнулися LLM моделі, наприклад галюцинації, обмежені база знань моделі для зміни якої потрібне перенавчання для оновлення інформації та інші. Також дана архітектура значно підвищує релевантність відповідей умовах, коли модель повинна працювати з актуальними або специфічними знаннями, не врахованими під час початкового навчання. За словами експерта Євгена Моспана (EPAM), RAG стає ключовим елементом у створенні корпоративних рішень, оскільки дозволяє організаціям використовувати власні дані без необхідності повторного навчання великих мовних моделей. Це робить технологію ідеальною для побудови інтелектуальних асистентів, чат-ботів і пошукових систем, орієнтованих на конкретну предметну область [5].

Один із трьох головних етапів реалізації системи на основі RAG – це побудова конвеєра обробки даних (embedding pipeline) за яким виконуємо підготовку, поділ, векторизацію та індексацію текстової інформації для подальшого пошуку та генерації відповідей. На етапі підготовки (data ingestion) система приймає вхідні документи у форматах доступних форматах, виконує їхню структуризацію, очищення та розбиття на логічні фрагменти, зазвичай у вигляді блоків тексту фіксованого розміру. Кожен фрагмент отримує метадані, які дозволяють ідентифікувати його джерело. Далі за допомогою попередньо обраної моделі embedding виконується перетворення кожного текстового фрагмента у векторну форму. Отримані вектори разом із метаданими зберігаються у векторному сховищі, яке пізніше буде використано на 2 етапі пошуку релевантних відповідей. Саме цей pipeline дозволяє забезпечити швидкий та точний пошук на основі змістовної близькості між запитом користувача і документами. Нижче наведений рисунок алгоритму архітектури Embedding Pipeline (рис. 2.2) [29].

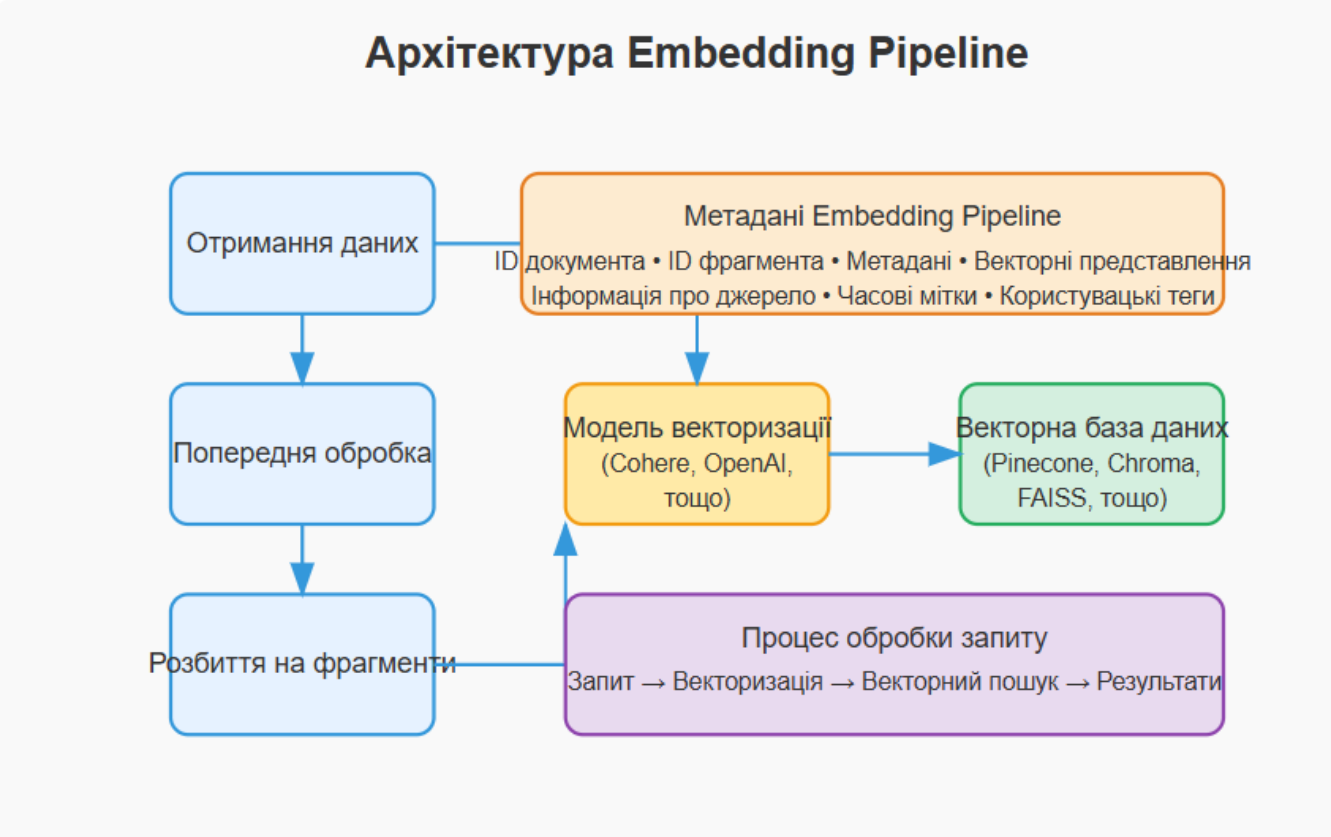


Рис. 2.2. Алгоритм архітектури Embedding Pipeline

Retrieval – це етап у RAG-архітектурі зосереджений на вилучення релевантної інформації з векторного сховища на основі запиту користувача. У системах, реалізованих на базі Oracle Cloud Infrastructure, цей етап забезпечується через порівняння векторного представлення запиту з попередньо збереженими embedding-векторами текстових документів. Завдяки використанню технології AI Vector Search в Oracle Database, пошук відбувається з високою швидкістю та точністю. Результатом цього етапу є контекстуально близькі документи, які передаються генеративній моделі для формування обґрунтованої відповіді [41].

Генерація тексту є завершальним етапом у системах згаданій раніше системі, під час якого велика мовна модель формує відповідь, спираючись на отримані з джерел фрагменти інформації. Цей процес забезпечує перетворення релевантного контексту в природномовний текст, орієнтований на конкретний запит користувача. Генерація здійснюється через застосування моделей типу encoder-decoder або decoder-only, які працюють з інструкцією (prompt), який

включає запит і витягнуті документи. Якість відповіді суттєво залежить від способу форматування контексту, обраного шаблону запиту та параметрів моделі. У практичній реалізації часто використовуються підходи zero-shot або few-shot, що дозволяє налаштовувати поведінку моделі без потреби її перенавчання. Крім того, застосовується так звана інструкційна генерація (instruction-following), коли модель навчається реагувати на запити у відповідності до сформованих інструкцій або завдань [15].

Для реалізації функцій генерації відповіді використано модель cohere.command-r-08-2024, а для поділу файлів на шматки для векторного сховища модель cohere.embed-multilingual-v3.0. Нижче у таблиці наведені характеристики даних моделей з сайту Oracle (табл. 2.1).

Таблиця 2.1

Характеристики моделей за документацією Oracle

Назва моделі	cohere.command-r-08-2024	cohere.embed-multilingual-v3.0
Тип моделі	Chat	Embedding
Особливості моделі	Модель підтримує до 128 000 токенів на один запуск (включаючи запит і відповідь), при цьому максимальна довжина відповіді становить 4 000 токенів. Доступна можливість тонкого налаштування з використанням власного датасету, де запит може містити до 16 000 токенів. Вона демонструє підвищену точність у розв'язанні математичних задач, програмуванні та логічному міркуванні. Підтримується багатомовна генерація з функцією пошуку (RAG) та налаштованими посиланнями на джерела. Для виділеного висновування (inference) модель може бути розгорнута на спеціалізованому AI-кластері.	Модель підтримує англійську та приблизно 109 мов. Створює 1024-вимірний embedding-вектор для кожного запиту. Проте обмежена до 96 речень за один запуск; максимум 512 токенів на одне embedding-представлення.

Сформовано автором на основі [28].

На завершальному етапі реалізації архітектури RAG було проведено тестовий запуск системи на віртуальній машині Oracle Cloud за допомогою хмарного терміналу Oracle Cloud Shell. Даний термінал створений для керування ресурсами OCI. У ході цього запуску відбулася завантаження PDF-файлів із тематикою криптовалют, блокчейну і децентралізованих фінансів, які було розділено на 530 змістових фрагментів (chunks) для подальшої обробки. Для побудови векторних представлень фрагментів застосовано embedding-модель від Cohere, а отримані вектори збережено у векторне сховище на базі Oracle Database 23c. Після цього було ініціалізовано мовну модель та сформовано повний RAG-ланцюг для обробки запитів користувача. Як свідчить результат на терміналі, усі компоненти системи були успішно активовані та готові до подальшого тестування. Результат тестового запуску представлено у вигляді лістингу (рис. 2.3).

```
Loading book: pdf_doc/Віртуальна валюта. Біткоїн та технологія блокчейн.pdf...
Loaded 23 pages from pdf_doc/Віртуальна валюта. Біткоїн та технологія блокчейн.pdf
Loading book: pdf_doc/Впровадження блокчейн-технологій в економіці України переваги та виклики.pdf...
Loaded 10 pages from pdf_doc/Впровадження блокчейн-технологій в економіці України переваги та виклики.pdf
Loading book: pdf_doc/ЕКОНОМІЧНА ПРИРОДА ВІРТУАЛЬНИХ ВАЛЮТ.pdf...
Loaded 39 pages from pdf_doc/ЕКОНОМІЧНА ПРИРОДА ВІРТУАЛЬНИХ ВАЛЮТ.pdf
Loading book: pdf_doc/Криптовалюта як інвестиційний інструмент .pdf...
Loaded 20 pages from pdf_doc/Криптовалюта як інвестиційний інструмент .pdf
Loading book: pdf_doc/МОДЕЛЮВАННЯ КОМПЛЕКСНОЇ ОЦІНКИ РИЗИКУ КІБЕРШАХРАЙСТВ ТА ЛЕГАЛІЗАЦІЇ КРИМІНАЛЬНИХ ДОХОДІВ .pdf...
Loaded 136 pages from pdf_doc/МОДЕЛЮВАННЯ КОМПЛЕКСНОЇ ОЦІНКИ РИЗИКУ КІБЕРШАХРАЙСТВ ТА ЛЕГАЛІЗАЦІЇ КРИМІНАЛЬНИХ ДОХОДІВ .pdf
Splitted the documents in 530 chunks...
Size of splits: 24bytes
Initializing Embeddings model...
Loading Cohere Embeddings Model...
Size of splits---: 4776bytes
Indexing: using ORACLEDB as Vector Store...
Index created.
No reranking...
Using COHERE llm...
/home/opc/AI_documents/init_rag_streamlit.py:205: LangChainDeprecationWarning: The class `Cohere` was deprecated in LangCha
package and should be used instead. To use it run `pip install -U :class:`~langchain-cohere` and import as `from :class:`~l
  llm = Cohere(
Building rag_chain...
Init RAG complete...
```

Рис. 2.3. Лістинг тестового запуску чат-бота

2.2. Взаємодія елементів через API Cohere та Oracle DB за допомогою фреймворку LangChain

Одним із головних етапів побудови системи генеративного чат-боту є забезпечення ефективної взаємодії між моделями, інфраструктурою зберігання векторів та компонентами логіки обробки запитів. У реалізованому чат-боті взаємодія досягається через використання API сервісу Cohere для генерації embedding-векторів і виведенні згенерованих відповідей, а також через підключення до векторного сховища, реалізованого на базі Oracle Database 23c. OCI API не використовується через неможливість використання моделей безкоштовно, тому що закінчився Free Tier. Координацію всіх компонентів забезпечує Python-фреймворк LangChain, який дозволяє легко будувати послідовності операцій, включаючи індексацію, пошук релевантних фрагментів і генерацію відповіді.

API (Application Programming Interface) – це програмний інтерфейс, який забезпечує стандартизований спосіб взаємодії сторонніх додатків із зовнішніми сервісами. В OCI API реалізовані на основі REST-протоколу і дозволяють виконувати всі основні операції з ресурсами: створення, оновлення, видалення, отримання даних. Однією з ключових особливостей API є високий рівень безпеки, який забезпечується за допомогою криптографічної автентифікації. Для доступу до API користувач має надати конфігураційний файл із зазначенням таких елементів, як user ocid, tenancy ocid, compartment ocid, ідентифікатор ключа, кінцеву точку регіону, де зареєстрований аккаунт та шлях до приватного ключа. Ступінь захищеності залежить від налаштування безпеки відділення, наприклад якщо взагалі його не налаштовувати, то стандартні вимоги до API будуть тільки compartment ocid та кінцева точка регіону. Усі запити підписуються приватним ключем, що гарантує автентичність та захист даних. Завдяки цим механізмам якщо у вас вкрадуть приватний ключ, то крадій все одно не отримаю доступ до інформації та ваших моделей [32].

Cohere API – це RESTful інтерфейс, який дозволяє розробникам інтегрувати генеративний ШІ у власні застосунки. API охоплює сервіси для генерації тексту, побудови ембеддінгів, класифікації, резюмування, переформулювання запитів та інші операції з природною мовою. Доступ здійснюється через прості HTTP-запити з автентифікацією за допомогою приватного API-ключа. Cohere API побудований на принципах надійної масштабованості та стандартизованої структури запитів і відповідей. Кожен endpoint має чітко задокументовані параметри, приклади запитів та структуру відповіді. Наприклад, для endpoint у generate можна налаштувати мову, кількість токенів, кількість варіантів відповіді, тональність тексту (ввічлива, неформальна) або вказати вхідні підказки. Особливу увагу приділено семантичному пошуку, де ембеддінги можна створювати та порівнювати з високою точністю. Крім того, на сайті Cohere можна побачити статистику обробки запитів та їх моніторинг, що дає змогу ефективно контролювати використання моделей і оптимізувати роботу з великими мовними моделями у реальному часі. Варто зазначити, що кількість запитів та токенів до моделі обмежена і залежить від типу приватного API ключа, проте кожного місяця оновлюється. Для кращого розуміння взаємодії елементів чат-боту з API нижче наведено схему послідовного обміну даних для генерування відповіді (рис. 2.4) [19].

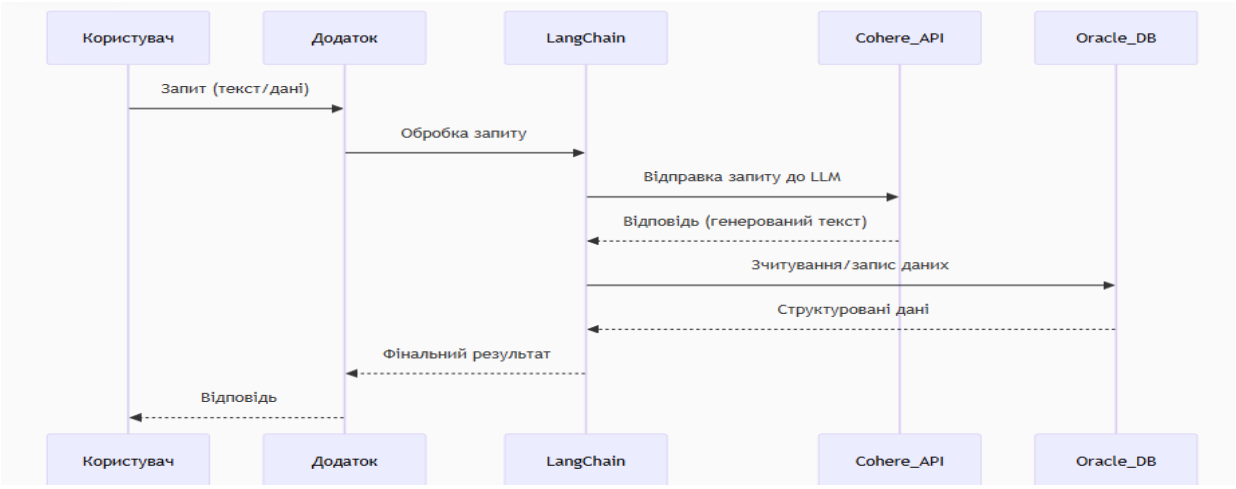


Рис. 2.4. Послідовність обміну даними між компонентами системи.

Програмна реалізація чат-боту поєднує в собі можливості генеративного штучного інтелекту, векторного пошуку та взаємодії з зовнішніми API, була побудована з використанням мови програмування Python і сучасного фреймворку LangChain. Python є загальновизнаним стандартом у галузі машинного навчання та обробки природної мови завдяки своїй гнучкості, широкому спектру бібліотек, активній спільноті та простоті інтеграції з іншими системами. У цій системі Python використовується для організації повного циклу обробки запиту, тобто введення даних користувачем, взаємодії з векторним сховищем, формування запитів до моделей генерації та отримання зворотного зв'язку. До основних бібліотек системи відносяться LangChain, Cohere, Oracledb, streamlit, OCI, RE та OS. Бібліотека Re використовується для обробки тексту та видалення зайвих для моделі символів, а OS для роботи з шляхами до файлів, оскільки вони розміщенні на віртуальній машині.

LangChain – це сучасний фреймворк, створений для побудови застосунків на основі великих мовних моделей, який дозволяє ефективно поєднувати різні компоненти в одному логічному ланцюзі. Основна ідея фреймворку полягає у тому, щоб забезпечити розробникам можливість інтегрувати LLM із зовнішніми джерелами даних, інструментами, API та базами знань, забезпечуючи динамічну генерацію відповідей на основі актуальної інформації. LangChain надає високий рівень абстракції для керування послідовністю обробки запитів, що значно полегшує реалізацію систем зі складною логікою роботи. Нещодавно популярність фреймворку стрімко зросла на тлі розмов про ШІ-агентів та агентські робочі процеси [2].

Серед ключових особливостей вирізняється модульна структура фреймворку, яка дозволяє будувати застосунки з повторно використовуваних та взаємозамінних компонентів. У межах цієї архітектури можна налаштовувати окремі елементи, зокрема сховища пам'яті для збереження історії діалогу, інструменти для інтеграції із зовнішніми сервісами, векторні бази даних для

пошуку інформації та агенти. LangChain підтримує різні режими роботи: від простих послідовних ланцюгів до більш складних агентних систем, які самостійно обирають наступні дії, адаптуючись до контексту задачі. Крім того, важливою перевагою є можливість інтеграції з великою кількістю різних моделей, провайдерів LLM та векторних баз даних [22].

Також LangChain надає засоби для трасування та моніторингу роботи компонентів через інструменти observability, що особливо корисно в умовах розробки та масштабування промислових рішень. Інтерфейси для налаштування векторного пошуку, кешування результатів та управління пам'яттю користувача дозволяють створювати персоналізовані системи зі збереженням контексту між запитами. Завдяки підтримці таких форматів, як LangChain Expression Language (LCEL), розробники можуть описувати логіку взаємодії між компонентами у декларативному вигляді. Дана особливість підвищує прозорість і читабельність коду [22].

У процесі розробки особливу увагу було приділено розмежуванню логіки на рівні окремих модулів. Наприклад компонент відповіді користувачеві відокремлений від модулів, що відповідають за пошук релевантної інформації у векторному сховищі. Завдяки такому підходу стало можливим незалежно тестувати генерацію відповідей, змінювати джерела знань без впливу на основну логіку або ж підключати додаткові джерела контексту. LangChain у цьому випадку виконує роль «об'єднуючої шини», яка дозволяє поєднувати логіку, пам'ять, інструменти та запити в єдиному ланцюгу, забезпечуючи прозоре керування всіма кроками виконання.

Отже, реалізована взаємодія між API сервісів Cohere та Oracle Database 23c за допомогою фреймворку LangChain стала основою функціонування генеративного чат-бота з підтримкою векторного пошуку. Дана архітектура має потенціал до подальшого розширення в напрямку реалізації агентних підходів та персоналізованих рішень. Також вона може використовуватись як шаблон для

створення нових чат-ботів та помічників, але для інших сфер та роботи з власними базами даних.

2.3. Підготовка та векторизація бази знань для чат-боту

Підготовка та векторизація бази знань є одним із початкових етапів у побудові інтелектуального чат-боту на основі архітектури Retrieval-Augmented Generation. Якість відповідей, які генерує модель, безпосередньо залежить від того, наскільки релевантно та структуровано представлено інформаційне середовище, з якого здійснюється пошук. Саме тому процес попередньої обробки, нормалізації та перетворення тексту у векторні представлення є визначальним у досягненні високої точності, повноти та контекстної відповідності під час діалогу з користувачем.

Процес попередньої обробки тексту передбачає низку послідовних операцій, кожна з яких має на меті зменшити шум у даних та стандартизувати вхідну інформацію. Однією з перших процедур є приведення тексту до нижнього регістру для уникнення варіативності слів залежно від їх написання. Далі здійснюється видалення пунктуаційних знаків, чисел і зайвих пробілів, що не несуть семантичного навантаження. Важливими етапами також є токенізація – поділ тексту на окремі слова або фрази, і видалення стоп-слів, тобто загальновживаних лексем, які не мають значущості для контексту запиту. Лематизація та стемінг дозволяють звести слова до базової чи кореневої форми, зменшуючи розмірність простору ознак. Для реалізації цих процедур активно використовуються бібліотеки Natural Language Toolkit (NLTK), spaCy. На більш складному рівні застосовуються вже методи машинного навчання, зокрема класифікація тексту, яка дозволяє відсіювати нерелевантну інформацію або розділяти дані за тематичними групами, а також методи виявлення дублікатів або аномалій. Застосування цих методів у підготовці бази знань дає змогу створити чисту й однорідну текстову сукупність, придатну для подальшої векторизації та ефективного інформаційного пошуку [34].

У межах реалізації чат-бота було прийнято рішення використовувати текстові дані, що стосуються тематики блокчейну, криптовалют і децентралізованих фінансів, оскільки ці джерела є відкритими, структурованими та мають значну аналітичну цінність. Для формування бази знань обрано документи у форматах, які забезпечують зручність попередньої обробки, розділення на одновимірні шматки та подальшу векторизацію. Основна увага приділялася форматам, які найчастіше використовуються в AI системах на основі RAG, зокрема PDF, DOCX, TXT. У таблиці нижче представлено підтримувані формати даних, які можуть бути використані для побудови RAG-системи.

Таблиця 2.2

Підтримувані формати файлів RAG

Категорія	Розширення
Текст	.txt
Портативні	.pdf, .md, .msg, .org, .rst, .csv, .tsv
Загальні	.eml, .html, .xml
Microsoft Office	doc, .docx, .ppt, .pptx, .xls, .xlsx, .rtf
Open Document	.odt, .odp, .ods
Електронні книги	epub
JSON	.json, .json
Зображення	.png, .jpeg/.jpg, .tiff, .bmp, .gif
Аудіо	.mp3, .ogg, .wav, .flac, .aac, .m4a
Відео	.mp4, .avi, .mov, .mkv
Спеціальні	.custom, .web

Джерело: [30].

Згідно таблиці RAG підтримує значну кількість форматів і поєднання деяких з них дозволяє залучати ширший спектр джерел інформації, від текстових документів до структурованих даних, що підвищує якість та об’єктивність генерації відповідей. Однак робота з багатоформатними даними створює низку проблем: неоднорідність структури, обмежену підтримка бібліотек, значні витрати на попередню обробку. Для стабільної роботи RAG потрібно буде привести дані в один формат, проте з даним процесом завжди виникають складності особливо потрібно враховувати в якому форматі отримуємо підказки користувачів та чи

надаємо можливість їм прикріплювати файли. Наприклад, помилки в розбитті PDF на шматки можуть призвести до втрати контексту, а некоректна конвертація JSON до спотворення семантики даних. Тому ключовим етапом є вибір оптимальних інструментів та налаштування конвертації з урахуванням специфіки кожного типу файлів.

У розробці чат-боту важливою складовою є векторне подання текстових даних, що дозволяє ефективно здійснювати пошук за змістом, а не лише за ключовими словами. Вектор − це просто набір чисел, що представляє ознаки об'єкта, незалежно від того, чи є цей об'єкт словом, реченням, документом, зображенням, відео- чи аудіофайлом. Вектори потрібні, оскільки порівняння або пошук такого типу неструктурованого контенту є складним для комп'ютерів. З іншого боку, порівняння або пошук векторів набагато простіше та базується на добре зрозумілій математиці. Вектори зберігаються в базі даних разом з об'єктами даних, на яких вони базуються, або з посиланням на них. Вектори, математично близькі один до одного, як правило, описують об'єкти зі схожими ознаками, тому можна швидко порівнювати або шукати їх і повертати схожі об'єкти. Векторна база даних – це будь-яка база даних, яка може безпосередньо зберігати та керувати векторними вбудовуваннями, а також обробляти неструктуровані дані, які вони описують. Зважаючи на важливість векторного пошуку для генеративного штучного інтелекту, технологічна галузь породила багато спеціалізованих, автономних векторних баз даних. Тим часом, версії визнаних фаворитів, таких як Oracle Database та база даних MySQL з відкритим кодом , включили вектори як рідний тип даних поряд з багатьма іншими типами даних. Це дозволяє здійснювати пошук комбінації бізнес- та семантичних даних швидше та точніше, оскільки обидва типи даних доступні в одній базі даних [39].

Векторний пошук у базі даних Oracle AI є сучасною технологією, яка дозволяє ефективно знаходити інформацію на основі семантичної близькості, а не лише збігів за ключовими словами. Oracle Database 23ai інтегрує векторний пошук

безпосередньо у свою інфраструктуру, дозволяючи зберігати вектори у спеціалізованому типі даних vector, індексувати їх та здійснювати пошук із використанням подібних метрик, таких як косинусна або евклідова відстань. Завдяки підтримці SQL-операторів, таких як vector_search, користувачі можуть поєднувати традиційні структуровані запити з векторними, що забезпечує інтеграції з аналітичними системами, процесами класифікації, рекомендацій та генерації контенту. Як зазначено на офіційному сайті Oracle, ця технологія дозволяє безперервно отримувати релевантну інформацію з великих наборів даних у режимі реального часу, забезпечуючи масштабованість, продуктивність і точність, необхідні для корпоративного впровадження AI-рішень, таких як чат-боти та цифрові помічники [11].

Підсумовуючи, процес підготовки та векторизації бази знань становить фундаментальну основу функціонування чат-бота на архітектурі RAG. Реалізація етапів нормалізації, токенізації та перетворення текстових даних у векторні представлення забезпечує точність і релевантність інформаційного пошуку. Обрана комбінація форматів даних та технологій векторизації, зокрема використання Oracle Database 23ai, дозволяє ефективно обробляти запити та генерувати контекстуально відповідні відповіді у сфері блокчейну, криптовалют і децентралізованих фінансів.

РОЗДІЛ 3. РОЗРОБКА ВЕБ-ІНТЕРФЕЙСУ СИСТЕМИ

3.1. Фреймворк Streamlit для швидкої реалізації користувацького інтерфейсу в браузері

У процесі розробки чат-бота завершальним етапом є вибір ефективного інструменту для створення користувацького інтерфейсу, який би забезпечив зручну взаємодію користувача з системою та зберігав оцінку користувачів щодо якості відповідей через браузер. Такий інтерфейс повинен бути не лише функціональним і адаптованим до потреб кінцевого користувача, а й швидким у реалізації, простим в інтеграції з моделями штучного інтелекту та здатним до масштабування зі зростанням складності проєкту. З огляду на ці вимоги, для розробки вебінтерфейсу чат-бота було розглянуто три популярні Python фреймворки: Streamlit, Gradio та Evidence.

Gradio, Streamlit та Evidence – три популярні інструменти для створення інтерактивних дата-додатків. Хоча всі вони дозволяють будувати веб-інтерфейси, їхні основні сфери застосування відрізняються: Gradio здебільшого застосовується для демонстрації моделей машинного навчання завдяки простоті підключення до Python-функцій та миттєвому запуску у браузері, проте існує багато практичних реалізацій демонстраційних версій чат-ботів та агентів. Streamlit вирізняється широкими можливостями для створення інтерактивних панелей та додатків із гнучким компонентним дизайном, дана особливість робить його зручним для інтеграції з RAG-системами та AI-агентами. Evidence орієнтований переважно на SQL-аналітиків та бізнес-звітність. Кожен із них має свої переваги та обмеження, тому доцільним є порівняння їх ключових функціональних характеристик, що дозволить обґрунтовано обрати найкращий варіант для реалізації інтерфейсу чат-бота (табл. 3.1) [18].

Таблиця 3.1

Порівняння фреймворків для створення інтерфейсу

Характеристика	Gradio	Streamlit	Evidence
Створення інформаційних панелей	Обмежена	Спеціалізований для даної задачі	Оптимізований для звітності
UI-компоненти	Базові	Розширені	Розширені
Джерело даних	Функції Python	Python DataFrame	SQL-бази даних
Мова розробки	Python	Python	SQL + Markdown
Розгортання	Python-середовище	Python-середовище	Статичний сайт
Продуктивність	Висока	Висока	Миттєва
Гнучкість у налаштуванні	Обмежена	Гнучка	Підтримує повну персоналізацію
Інтеграція зі ШІ/ML	Оптимізований для ML та демо версій AI	Потребує додаткового налаштування	Слабко орієнтований на AI
Простота інтеграції з RAG	Висока	Дуже висока	Низька
Можливість зберігати відгук	Через Python-функції	Вбудовані компоненти	Потрібне окреме рішення
Підтримка API	Частково (через backend-логіку)	Вбудована підтримка API	Обмежена
Вимоги до знань програмування	Базові знання Python	Середній рівень Python	SQL + базові знання Markdown
Підходить для прототипу	Так	Так	Частково
Розширення функціоналу в майбутньому	Складно	Легко	Складно

Сформовано автором на основі [18].

Після детального аналізу можливостей інструментів було прийнято рішення на користь Streamlit, оскільки він забезпечує оптимальне поєднання гнучкості, інтеграції з моделями штучного інтелекту, зручності створення користувацького інтерфейсу та вбудованої підтримки API. На відміну від Evidence, який орієнтований переважно на звітність, або Gradio обмежений у можливості кастомізації. Streamlit дозволяє легко реалізувати вебінтерфейс, який підтримує

інтерактивну взаємодію, збір зворотного зв’язку користувачів та подальше розширення функціональності системи.

Також важливу роль у виборі Streamlit для реалізації вебінтерфейсу відіграв аналіз практичного досвіду його використання, викладений у фаховій літературі. У книзі [36] детально розглянуто особливості розробки інтерактивних застосунків для обробки та візуалізації даних за допомогою даного фреймворку. Автор зосереджується на тому, як за допомогою лише мови Python можна створювати повноцінні вебінтерфейси без необхідності залучення додаткових вебтехнологій. У межах книги розглянуто інтеграцію Streamlit з популярними бібліотеками для обробки даних, побудови моделей машинного навчання та використання LLM моделей. Книга дозволила побачити практичну реалізацію взаємодії моделі GPT та вебінтерфейсу, що значно прискорило розробку власно інтерфейсу з моделями Cohere, логіка реалізації зберіглась. Ці практичні приклади з книги стали основою для розробки в рамках кваліфікаційної роботи, оскільки засвідчили ефективність Streamlit у побудові адаптивного інтерфейсу, здатного інтерактивно взаємодіяти з AI-компонентами системи [36 с. 98-103].

Згідно з офіційною документацією Streamlit, архітектура фреймворку побудована на клієнт-серверній моделі, яка дозволяє досягати високої інтерактивності та простоти у створенні вебдодатків на Python. Серверна частина запускає скрипт Python, який відповідає за логіку обробки даних, завантаження моделей штучного інтелекту та генерацію елементів інтерфейсу. Клієнтська частина, реалізована за допомогою звичайних вебтехнологій, відображає ці елементи у браузері користувача. Ключовою особливістю Streamlit є механізм реактивного рендерингу, завдяки якому кожна взаємодія з елементами інтерфейсу (введення тексту, натискання кнопки чи вибір зі списку) призводить до перезапуску всього Python-скрипта з урахуванням нового стану інтерфейсу. Це дозволяє уникнути складної обробки подій та створення власних циклів оновлення. Зміна вхідних даних автоматично оновлює всі залежні елементи,

забезпечуючи синхронізацію між логікою та інтерфейсом без потреби у збереженні проміжного стану вручну. Проте така архітектура створює значні проблеми з продуктивністю та управлінням станом для дуже складних додатків, які будуть орієнтуватись на великий трафік та повний контроль процесів застосунку, оскільки при кожній найменшій дії користувача відбувається повторне завантаження та обробка всіх даних, повторна ініціалізація моделей та повторне виконання всіх обчислень [37].

На основі обраної архітектури та особливостей фреймворку Streamlit було розроблено веб-інтерфейс чат-боту, оптимізований для взаємодії з користувачами з тестовими даними у галузі криптовалют та блокчейну. Дизайн інтерфейсу виконано у мінімалістичному стилі з акцентом на функціональність, інтуїтивність використання та подальшу персоналізацію для клієнтів. Головна сторінка додатку містить область вводу запиту, панель історії повідомлень. Загальний вигляд розробленого вебінтерфейсу представлено на рисунку нижче (рис. 3.1).



Рис. 3.1. Прототип вебінтерфейсу чат-боту

Для реалізації програмної частини чат-бота було використано інтегроване середовище розробки JupyterLab. Дане середовище дозволило ефективно організувати структуру проєкту, забезпечити запуск компонентів у Docker-контейнері на віртуальній машині, а також взаємодіяти з файлами конфігурації,

кодом Streamlit-інтерфейсу та модулями взаємодії з базою даних і API. На рисунку представлено частину коду та основне робоче вікно середовища в якому здійснюється налаштування і запуск інтерфейсу чат-бота (рис. 3.2).

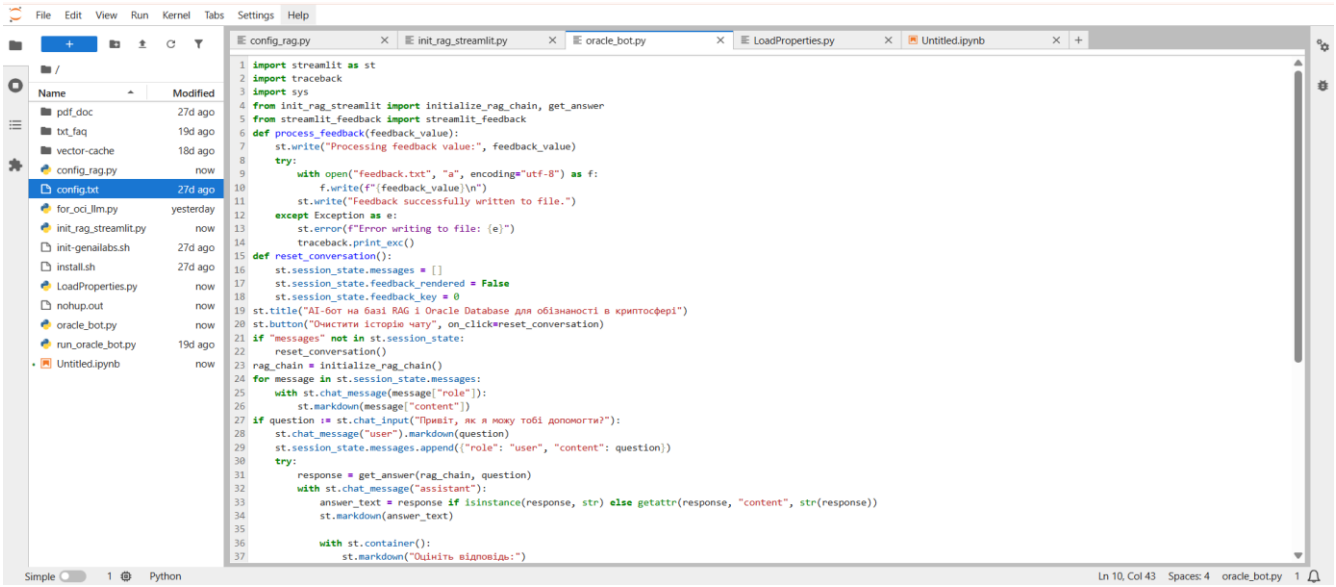


Рис. 3.2. Програмне робоче середовище проєкту та частина коду інтерфейсу

Розроблений інтерфейс наразі слугує прототипом, який демонструє базову функціональність чат-бота та його здатність до інтеграції з новими джерелами знань, збереженням історії взаємодії та збору зворотного зв'язку. Він створений як шаблон, який у майбутньому може бути персоналізований та оновлений відповідно до специфіки запитів потенційних користувачів, зокрема представників малого та середнього бізнесу.

3.2. Тестування відповідей чат-бота та оцінка ефективності впровадження на безкоштовному рівні OCI

Один із ключових аспектів ефективності роботи генеративного штучного інтелекту полягає у правильному проєктуванні вхідних запитів, що отримали назву prompt engineering. Prompt engineering – це процес створення, оптимізації та вдосконалення підказок (prompts) для моделей штучного інтелекту з метою отримання найбільш релевантних, точних і корисних результатів. У контексті генеративного ШІ підказка – це текстовий запит, який формулює завдання або

контекст, який дозволяє моделі правильно інтерпретувати очікування користувача. Інженерія підказок розглядається як критичний етап у тестуванні та перевірці поведінки AI-моделей, оскільки саме від формулювання запиту залежить точність відповіді, контекстуальна релевантність і надійність виводу [12].

Ефективне створення підказок є ключовим чинником якості відповідей генеративних моделей у системах типу RAG. Якість підказки безпосередньо впливає на здатність моделі правильно інтерпретувати завдання та генерувати точні, логічно зв'язані відповіді. Згідно з рекомендаціями компанії Cohere, оптимальні підказки мають бути чіткими, контекстуально багатими та орієнтованими на конкретний результат. Важливо не лише формулювати сам запит, а й надавати приклади очікуваного формату відповіді, структури тексту або навіть бажаного стилю. Наприклад, замість загального «Поясни тему X», ефективніше буде: «Поясни тему X так, як для новачка, використовуючи прості терміни і приклади». Також варто уникати надмірно загальних або двозначних формулювань, які можуть призвести до невизначених відповідей. Таким чином, інженерія підказок є не лише технічним процесом, а й важливим дизайнерським елементом взаємодії з генеративним ШІ, що визначає точність, релевантність і надійність результатів [7].

Крім формулювання чіткого запиту, особливу увагу слід приділяти структурі підказки. Важливою частиною підказки є контекстне підсилення, тобто включення релевантної інформації до самого запиту або використання префіксів, що задають роль моделі (наприклад: «Ти – експерт з фінансів, поясни...»). Це дозволяє моделі краще орієнтуватися у сфері завдання і генерувати відповіді, які відповідають очікуваному рівню складності та стилю. Ще однією ефективною технікою є few-shot prompting – включення кількох прикладів запитів і бажаних відповідей до основної підказки. Це особливо корисно у випадках, коли потрібна стандартизація результатів або генерація тексту у визначеній формі. Таким чином, майстерність у prompt engineering не лише покращує результати генерації, а й

дозволяє адаптувати поведінку моделей до конкретних бізнес-завдань або навчального середовища [33].

Тестування чат-бота здійснювалося в межах розробленого веб-інтерфейсу на основі Streamlit із підключенням до Oracle Database та використанням моделей від Cohere. Для прикладу, на рисунку нижче зображено інтерфейс взаємодії з ботом, де продемонстровано відповідь на запит користувача (рис 3.3).

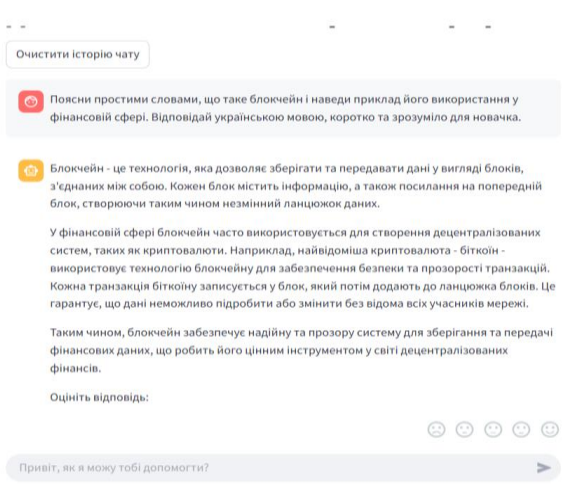


Рис. 3.3. Тестування чат-бота на генерацію відповіді

На наступному рисунку зображено згенеровану відповідь на основі бази знань та конкретними файлами звідки проводився пошук інформації (рис 3.4).

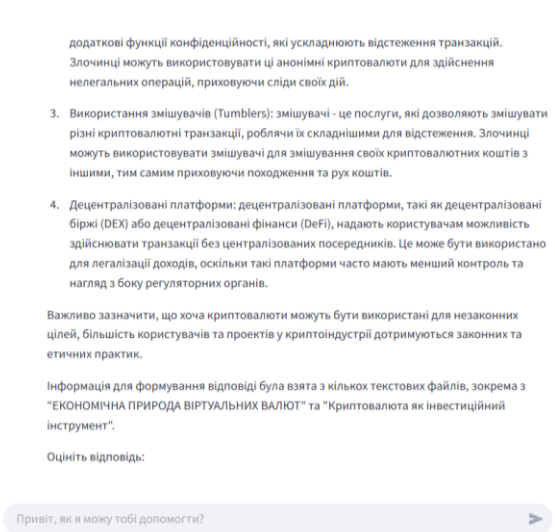


Рис. 3.4. Тестування згенерованої відповіді з відображенням джерел

За допомогою написання некоректного запиту перевірив відповідь чат-бота на галюцинації моделі, як ми бачимо з рисунку нижче модель відповіла заготовлену відповідь з відмовою, а не почала вигадувати дані (рис 3.5).

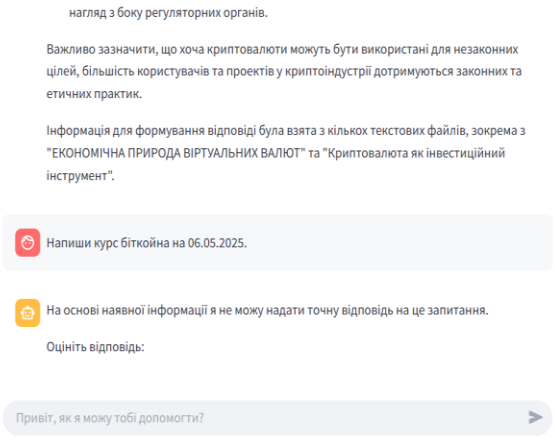


Рис. 3.5. Тестування відповіді чат-боту на некоректний запит

Після тестування чат-бот демонструє стабільну роботу, швидко обробляючи запити і надав точні відповіді. Інтеграція Cohere API, Oracle DB та LangChain дозволяє ефективно поєднувати генерацію текстів з векторним пошуком, а інтуїтивний Streamlit-інтерфейс спрощує взаємодію.

Реалізація чат-бота на пробному рівні OCI Free Tier виявилася цілком життєздатною: вдалося розгорнути всю інфраструктуру: віртуальну машину, налаштування сервера, базу даних, векторне сховище та доступ до потрібних моделей. На рівні Always Free вдалося теж реалізувати всю інфраструктуру, проте виникли проблеми з доступом до великих мовних моделей через платний функціонал нового розділу OCI. Для реалізації повної безкоштовної розробки чат-боту було прийнято рішення отримати тестові ключі до моделей напряму через офіційний сайт Cohere. Тому можна вважати, що будь-який розробник з певною базою знань може реалізувати безкоштовно власний генеративний чат-бот і підтримувати його безкоштовно, проте потрібно враховувати обмеження в доступної кількості пам’яті у базі даних та місячний ліміт у кількості 1000 запитів до різних типів моделей.

3.3. Перспективи розширення функціональності чат-бота та розрахунок економічного ефекту розробки

Останнім часом впровадження штучного інтелекту в бізнес-процеси набуває дедалі більшої популярності, особливо серед компаній малого та середнього бізнесу, які прагнуть автоматизувати рутинні завдання, покращити обслуговування клієнтів і зменшити витрати. Одним із найперспективніших напрямів такої автоматизації є використання інтелектуальних чат-ботів, здатних не лише відповідати на типові запитання, а й надавати детальні консультації, орієнтуючись на специфічні знання компанії. Завдяки поєднанню сучасних генеративних моделей та механізмів доступу до локальних джерел даних, чат-боти нового покоління відкривають широкі можливості для створення персоналізованих інформаційних систем із низьким порогом входу в експлуатацію та адаптацію.

Для оцінки комерційної доцільності реалізованої архітектури чат-бота потрібно змоделювати потенційні сценарії її комерційного використання в умовах ринку IT-рішень для малого та середнього бізнесу. Зокрема, інтелектуальний чат-бот, розроблений на базі гнучкої модульної структури з використанням генеративних моделей, може бути запропонований як шаблон або основа для швидкого розгортання персоналізованих систем під потреби клієнтів з різних галузей: торгівлі, обслуговування, консалтингу, освітніх послуг тощо. За рахунок мінімізації витрат на початкову розробку, цей підхід дозволяє реалізувати ефективну модель продажу готових рішень або створення сервісів за замовленням із суттєвим скороченням часу виходу на ринок.

У рамках цієї бізнес-моделі основним джерелом доходу можуть стати як прямі замовлення на кастомізацію чат-бота під специфіку конкретного клієнта, так і пропозиція шаблонів на умовах одноразової ліцензії або щомісячної підписки. Наприклад базова версія чат-бота з інтеграцією в сайт або систему може надаватись за передплатною моделлю з оплатою х доларів на місяць, тоді як повна

персоналізація під конкретну нішу може передбачати одноразовий платіж у визначеному розмірі у доларів.

Для оцінки комерційної ефективності розробленої архітектури чат-бота доцільно скористатися методикою порівняння витрат на створення рішення із прогнозованими доходами від його продажу. У цьому контексті використаємо класичний підхід до визначення потенційного прибутку П, який базується на формулі (3.1):

$$П_{загальний} = Д_{загальний} - В_{загальний}, \quad (3.1)$$

Де $Д_{загальний}$ – загальний дохід від реалізації проєкту протягом року.

$В_{загальний}$ – повні витрати на розробку, підтримку та просування продукту на рік.

Припустимо, що протягом першого року планується реалізувати до 20 індивідуальних впроваджень, з середньою ціною одного проєкту 500 умовних одиниць та подальшою підтримкою половини проєктів за 6000 ум. од. за рік. Таким чином, орієнтовний дохід становитиме (3.2):

$$Д_{загальний} = Д_{проєкт} \times К_{кількість} + Д_{підтримка} = 500 \times 20 + 6000 = 16000, \quad (3.2)$$

де $Д_{загальний}$ – загальний дохід від реалізації проєкту протягом року.

$Д_{проєкт}$ – дохід за розробку 1 проєкту.

$К_{кількість}$ – кількість проєктів за рік.

$Д_{підтримка}$ – дохід за підтримку проєктів за рік.

Загальні витрати розрахуємо за формулою (3.3):

$$В_{загальний} = В_{розробників} + В_{хостинг} + В_{маркетинг} + В_{операційні}, \quad (3.3)$$

де $В_{розробників}$ – витрати на розробників.

$В_{хостинг}$ – витрати на хостинг, домен та API.

$В_{маркетинг}$ – витрати на маркетинг.

$В_{операційні}$ – витрати на вирішення проблем та підтримки діяльності.

Витрати на розробників становлять 9000 ум. од., витрати на хостинг 2000 ум. од., на маркетинг 2000 ум. од. та операційні витрати у розмірі 1500 ум. од.

Таким чином:

$$В_{загальний} = 9000 + 2000 + 2000 + 1500 = 14500$$

Тепер можемо обрахувати прибуток за формулою «(3.1)»:

$$П_{загальний} = 16000 - 14500 = 1500$$

Отже, уже в перший рік реалізації потенційний прибуток може скласти 1500 ум. од, проте слід зосередитись більше на підтримці, тому що вона приносить стабільний дохід та потребує менших затрат.

Щоб глибше проаналізувати доцільність проєкту, варто розрахувати показники рентабельності та терміну окупності. Рентабельність R визначається як відношення прибутку до загальних витрат і виражається у відсотках (3.4):

$$R = \frac{П_{загальний}}{В_{загальний}} \times 100\% = \frac{16000}{14500} \times 100\% = 10{,}34\%, (3.4)$$

Далі обрахуємо ще потенційний термін окупності. Термін окупності (Т) – це період, протягом якого доходи від продажів повністю покривають витрати (3.5):

$$Т = \frac{В_{загальний}}{Д_{середній}} = \frac{14500}{(16000:12)} = 10{,}88 \text{ місяців}, (3.5)$$

де Т – термін окупності.

$В_{загальний}$ – загальні витрати на проєкт.

$Д_{середній}$ – середній дохід у місяць.

Також варто визначити точку беззбитковості – критичний обсяг продажів, за якого дохід дорівнює витратам. Цей показник визначається за формулою (3.6):

$$ТБЗ = \frac{В_{загальний}}{Д_{впровадження}} = \frac{14500}{500} = 29 \text{ впроваджень}, (3.6)$$

де ТБЗ – точка беззбитковості проєкту.

$В_{загальний}$ – загальні витрати на проєкт.

$Д_{впровадження}$ – дохід за розробку 1 чат-боту.

Отже, розроблений прототип чат-бота продемонструє задовільні фінансові показники згідно тестових показників. При прогнозованому доході в 16 000 ум. од. та загальних витратах на рівні 14 500 ум. од, розрахункова рентабельність становить приблизно 10,34 %. Такий рівень прибутковості дозволяє очікувати повернення вкладених коштів уже через 10,88 місяців від початку продажів. Точка беззбитковості становить 29 впроваджень за ціною 500 ум. од. кожне. Проте варто зазначити, що комерційна вигода залежить від масштабів проникнення на ринок, кількості підтримуючих додатків та ефективності маркетингу. На старті розроблені чат-боти матиму високі витрати та низькі доходи. Однак, при досягненні широкої користувацької бази та створенні додаткових платних функцій проект може стати прибутковим. Для реальної оцінки витрат на хостинг можна звернутися на сайт Oracle [25].

Подальший розвиток функціональності чат-бота передбачає поступове впровадження нових можливостей, спрямованих на підвищення цінності продукту для кінцевого користувача. До таких нових функціональних можливостей можна віднести процес переранжування документів за допомогою моделей Cohere Rerank.

Cohere Rerank – це один із компонентів платформи обробки природної мови від Cohere, створений для поліпшення результатів пошуку за допомогою повторного ранжування знайдених документів на основі їх релевантності до запиту. На відміну від традиційного пошуку, який ґрунтується виключно на векторній близькості чи ключових словах, Rerank аналізує запит і вміст документів глибше, використовуючи LLM-модель для врахування семантичного контексту. Це дозволяє розміщувати найбільш змістовні та корисні відповіді на початку списку результатів, що підвищує точність систем з відкритим пошуком RAG. Rerank підтримує до 100 документів одночасно та демонструє високу ефективність у багатомовному середовищі, що робить його універсальним рішенням для широкого спектру застосувань від пошукових систем до чат-ботів, навчання співробітників і систем підтримки клієнтів [31].

Особливістю використання переранжування в RAG архітектури є те, що після початкового пошуку релевантних фрагментів знань з векторного сховища, Rerank застосовується як другий етап, який перерозподіляє порядок результатів відповідно до того, наскільки добре кожен документ відповідає на конкретний запит користувача. Згідно з аналітикою AWS, така двоетапна стратегія дозволяє зменшити кількість нерелевантних або поверхневих відповідей та забезпечити кращу узгодженість з контекстом діалогу. Особливо важливо те, що Rerank не потребує повного перегенерування embedding-векторів чи перебудови індексації, а може бути легко інтегрований у вже існуючі RAG-системи. Це робить його зручним та економічно ефективним інструментом для адаптивного покращення якості відповідей без радикального переосмислення всієї архітектури [20].

Також можна створити функції розпізнавання голосу та обробки користувацьких зображень для покращення взаємодії чат-бота з користувачем. Розширити можливість обробки інших типів файлів у залежності від конкретного спрямування за потреби, вивантаження та обробка PDF-документів прямо з вебінтерфейсу. Ще одною важливою майбутньою функцією є інтеграція механізму авторизації та персоналізації запитів.

ВИСНОВКИ

У кваліфікаційній роботі досягнуто поставленої мети, а саме розроблено чат-бот із використанням Generative AI Service на платформі Oracle Cloud Infrastructure, який дозволяє здійснювати векторний пошук по власних даних користувача. Актуальність дослідження зумовлена стрімким розвитком генеративного штучного інтелекту, потребою в автоматизації обробки великої кількості текстових джерел та персоналізації інформаційної взаємодії з користувачами у різних галузях. Розроблена система пропонує сучасне рішення на основі Retrieval-Augmented Generation, поєднуючи точність бази знань із мовною гнучкістю моделей ШІ.

Було проведено теоретико-методологічний аналіз підходів до створення RAG-систем, вивчено архітектури інтеграції моделей generative AI із векторними базами даних, досліджено особливості платформенних рішень Oracle Cloud Infrastructure. Проаналізовано сучасні інструменти: OCI Generative AI, Oracle Database 23ai, LangChain, LLM Cohere, Streamlit. Обґрунтовано вибір Oracle як платформи для експериментів через її розвинуту AI-інфраструктуру, підтримку масштабування можливість безкоштовного використання. Виявлено актуальну потребу в універсальному рішенні, яке дозволяє зберігати, швидко оновлювати й використовувати корпоративні чи відкриті знання для генерації точних відповідей у чат-інтерфейсі.

На основі сформульованих вимог було спроєктовано архітектуру, яка забезпечує збереження embedding-векторів у Oracle Database, швидке оновлення бази знань, підтримку моделей генерації, та вебінтерфейс для користувача з можливістю залишати зворотний зв'язок. Реалізовано інтерфейс на Streamlit, вбудовано модуль збереження ембеддингів локально або в базі, систему пошуку релевантних відповідей та обробку повідомлень користувача з контекстом історії. Проведено тестування функціоналу на прикладі відкритих даних у сфері про

криптовалюти, блокчейн та децентралізованих фінансів. Тестування підтвердило стабільність, точність та коректність роботи на різних етапах pipeline.

Практичне значення одержаних результатів полягає в створенні універсального чат-бота, який може бути адаптований під будь-які текстові дані організації: від внутрішньої документації до освітніх матеріалів. Завдяки можливості швидкого оновлення бази знань та векторного пошуку бот ефективно застосовується для підтримки користувачів, автоматизованого навчання персоналу чи самообслуговування клієнтів.

Наукова новизна полягає в практичній реалізації генеративного підходу з використанням Generative AI Service у вигляді повноцінної RAG-системи з модульною архітектурою, яка може бути масштабована й модифікована під специфіку конкретного бізнесу.

На основі дослідження та апробації сформульовано рекомендації щодо подальшого розвитку системи: інтеграція механізму авторизації та персоналізації запитів, розширення підтримки різних розширень файлів за потреби, вивантаження та обробка PDF-документів прямо з вебінтерфейсу, використання переранжування даних та векторів, а також підключення голосового інтерфейсу для інтерактивного спілкування.

Отже, у ході виконання дипломної роботи було повністю реалізовано поставлену мету. Здійснено всебічний аналіз технологічних інструментів, спроєктовано архітектуру та створено діючий прототип чат-бота з підтримкою RAG, перевірено його працездатність і підтверджено ефективність підходу. Отримані результати засвідчують доцільність використання генеративного ШІ в інфраструктурі Oracle для задач доступу до персоналізованих знань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Буєвич А.О. Історія штучного інтелекту від 1950-х до сьогодні. freeCodeCamp.org.
URL: https://www.freecodecamp.org/ukrainian/news/istoriya-shtuchnoho-intelektu-vid-1950-kh-do-sohodni/ (дата звернення: 31.03.2025).

2. Вступ до LangChain: Найкращі приклади використання на підприємствах + найкращі інструменти та фреймворки - AI&YOU #56 - Skim AI. SkimAI.com. URL: https://surl.lu/xeblws (дата звернення: 31.03.2025).

3. Історія штучного інтелекту | КРВ. KRV. URL: https://krv.media/istoriya-shtuchnogo-intelektu-vid-ideyi-do-suchasnyh-tehnologij/ (дата звернення: 31.03.2025).

4. Повний посібник для початківців з генеративного ШІ. DreamHost Blog. URL: https://www.dreamhost.com/blog/uk/posibnik-z-generativnogo-shtuchnogo-intelektu/ (дата звернення: 31.03.2025).

5. Сучасний інструментарій фахівця у сфері штучного інтелекту: перспективи та можливості. EPAM.ua. URL: https://careers.epam.ua/blog/modern-toolkit-for-ai-professionals-yevgen-mospan (дата звернення: 31.03.2025).

6. Як працює генеративний ШІ: переваги та основні сфери застосування | Smodin. Посібник з написання статей - SmodinBlog. URL: https://smodin.io/blog/uk/how-does-generative-ai-work/
(дата звернення: 31.03.2025).

7. A guide to crafting effective prompts – Cohere. URL: https://docs.cohere.com/docs/crafting-effective-prompts
(дата звернення: 31.03.2025).

8. About Google DeepMind. URL: https://deepmind.google/about/ (дата звернення: 31.03.2025).

9. About our company | Cohere. Cohere. URL: https://cohere.com/about (дата звернення: 31.03.2025).

10. Access cloud services for free. Oracle.com. URL: https://www.oracle.com/cloud/free (дата звернення: 31.03.2025).

11. Advanced AI Vector Search for Business Data Insights. Oracle.com. URL: https://www.oracle.com/ua/database/ai-vector-search/ (дата звернення: 31.03.2025).

12. AI testing & prompt engineering: boosting automation & accuracy. URL: https://keploy.io/blog/community/ai-testing-prompt-engineering (дата звернення: 31.03.2025).

13. Chatbot Arena (formerly LMSYS): Free AI Chat to Compare & Test Best AI Chatbots. URL: https://lmarena.ai/ (дата звернення: 31.03.2025).

14. Devlin J. Chang M.-W. Lee K. Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. ArXiv.org. URL: https://arxiv.org/abs/1810.04805 (дата звернення: 31.03.2025).

15. Gao Y., Xiong Y., Gao X., Jia K., Pan J., Bi Y., Dai Y., Sun J., Wang M., Wang H. Retrieval-Augmented Generation for Large Language Models: A Survey. URL: https://arxiv.org/pdf/2312.10997 (дата звернення: 31.03.2025).

16. Generative AI. Oracle.com. URL: https://docs.oracle.com/en-us/iaas/Content/generative-ai/home.htm (дата звернення: 31.03.2025).

17. Generative artificial intelligence: evolving technology, growing societal impact, and opportunities for information systems research - information systems frontiers. SpringerLink. URL: https://link.springer.com/article/10.1007/s10796-025-10581-7 (дата звернення: 31.03.2025).

18. Gradio vs. Streamlit: choosing a tool for your data app | evidence learn. URL: https://evidence.dev/learn/gradio-vs-streamlit (дата звернення: 31.03.2025).

19. How to work with Cohere API. Apidog.com. URL: https://apidog.com/blog/cohere-api/ (дата звернення: 31.03.2025).

20. Improve RAG performance using Cohere Rerank. Amazon Web Services. URL: https://aws.amazon.com/ru/blogs/machine-learning/improve-rag-performance-using-cohere-rerank/ (дата звернення: 31.03.2025).

21. Introducing Claude. Anthropic.com. URL: https://surl.gd/fotxok (дата звернення: 20.04.2025).

22. Introduction|LangChain. URL: https://python.langchain.com/docs/introduction/ (дата звернення: 20.04.2025).

23. LLM Leaderboard - Compare GPT-4o, Llama 3, Mistral, Gemini & other models | Artificial Analysis. AI Model & API Providers Analysis | Artificial Analysis. URL: https://artificialanalysis.ai/leaderboards/models (дата звернення: 20.04.2025).

24. Oleksandr Redych and Ruslan Boichuk Comparative Analysis of Holt-Winters Algorithms on the Oracle Machine Learning Platform // Proceedings of the 12th International Conference on Applied Innovations in IT : volume 12, issue 1, pp.71-77 : Koethen, Germany, 19 March 2024 (DOI:10.25673/115644; PPN 1884680062) (Scopus) (дата звернення: 20.04.2025).

25. OCI price list. Oracle.com. URL: https://www.oracle.com/cloud/price-list (дата звернення: 20.04.2025).

26. Oracle Cloud Infrastructure Documentation. Oracle.com. URL: https://docs.oracle.com/en-us/iaas/Content/home.htm (дата звернення: 20.04.2025).

27. Oracle Cloud. Oracle.com. URL: https://cloud.oracle.com/ai-service/generative-ai/overview?region=uk-london-1 (дата звернення: 20.04.2025).

28. Pretrained Foundational Models in Generative AI. Oracle.com. URL: https://docs.oracle.com/en-us/iaas/Content/generative-ai/pretrained-models.htm (дата звернення: 20.04.2025).

29. RAG Assistants Introduction | Article. Wiki.genexus.com. URL: https://wiki.genexus.com/enterprise-ai/wiki?44,RAG+Assistants+Introduction (дата звернення: 20.04.2025).

30. RAG file formats | Article. Wiki.genexus.com. URL: https://wiki.genexus.com/enterprise-ai/wiki?137,RAG+File+Formats (дата звернення: 20.04.2025).

31. Reranking. Cohere. URL: https://cohere.com/llmu/reranking (дата звернення: 20.04.2025).

32. REST APIs. Oracle.com. URL: https://docs.oracle.com/en-us/iaas/Content/API/Concepts/usingapi.htm (дата звернення: 29.04.2025).

33. Shot-Based Prompting: Zero-Shot, One-Shot, and Few-Shot Prompting. URL: https://learnprompting.org/docs/basics/few_shot?srsltid=AfmBOop68C9hgBBDiZHnCDnoAgtTgaT6SB0GmBCioODXERGAFT--wOwq (дата звернення: 29.04.2025).

34. Text Preprocessing in NLP - GeeksforGeeks. URL: https://www.geeksforgeeks.org/text-preprocessing-for-nlp-tasks/ (дата звернення: 29.04.2025).

35. The history of AI: a timeline of artificial intelligence. Coursera. URL: https://www.coursera.org/articles/history-of-ai (дата звернення: 29.04.2025).

36. Treuille A. Streamlit for Data Science: Create Interactive Data Apps in Python. 2nd ed. de Gruyter GmbH, Walter, 2023. 301 p.

37. Understanding Streamlit's client-server architecture - Streamlit Docs. URL: https://docs.streamlit.io/develop/concepts/architecture/architecture (дата звернення: 29.04.2025).

38. Vaswani A. Shazeer N. Parmar N. Uszkoreit J. Jones L. Gomez A.N. Kaiser L. Polosukhin I. Attention Is All You Need. ArXiv.org. URL: https://arxiv.org/abs/1706.03762v7 (дата звернення: 29.04.2025).

39. What Is a Vector Database? Oracle.com. URL: https://www.oracle.com/database/vector-database/ (дата звернення: 29.04.2025).

40. What is OpenAI? Everything you need to know. Coursera. URL: https://www.coursera.org/articles/what-is-openai (дата звернення: 29.04.2025).

41. What Is Retrieval-Augmented Generation (RAG)? Oracle.com. URL: https://www.oracle.com/ua/artificial-intelligence/generative-ai/retrieval-augmented-generation-rag/ (дата звернення: 29.04.2025).

ДОДАТКИ

Додаток А

Особливості формування бази знань для RAG архітектури, роль попередньої обробки різних типів розширень файлів.

Формування бази знань для RAG архітектури передбачає обов’язкову попередню обробку вхідних документів з урахуванням їх типу, формату та структури. На початковому етапі реалізації системи, ключовим завданням є збір та визначення типів змістовних матеріалів. Вони можуть бути представлені у вигляді PDF, DOCX, TXT, HTML, CSV, фото та аудіо форматів. Також потрібно врахувати чи надавати користувач можливість використовувати власні файли для отримання кращої відповіді. Кожен з цих форматів має свої особливості при зчитуванні: PDF-файли можуть містити багатоколонковий текст, виноски чи графічні блоки, які потребують додаткової обробки для коректного вилучення текстового вмісту. У той же час, DOCX-файли часто містять стилістичні або структурні елементи (заголовки, списки, таблиці), які можуть слугувати орієнтирами для логічного розбиття на фрагменти. Простий текст у форматі TXT найменш складний для обробки, проте часто потребує ручної перевірки форматування та структури, особливо якщо джерело було конвертовано з інших типів документів.

Окремої уваги потребує обробка CSV-файлів, які використовуються для зберігання структурованих даних у табличному форматі. На відміну від текстових документів, CSV не містить розгорнутих текстів, а представляє інформацію у вигляді рядків і стовпців, де кожен запис може відповідати окремому факту, події або об’єкту. Важливим аспектом при роботі з цим типом даних є правильне трактування заголовків стовпців та логічна інтерпретація значень у кожному рядку. Також потрібно відокремити функцію обробки даного формату через необхідність працювати безпосередньо через SQL запити наприклад, через бібліотеку sqlite або створити AI агента для виконання функціоналу для оптимізації роботи чат-бота.

Продовження додатка А

Під час попередньої обробки CSV-файлів важливо здійснити очищення від пропущених або дубльованих значень, нормалізацію назв колонок, а також переведення кожного рядка в окремий структурований фрагмент тексту, який може бути перетворений у векторне представлення. Нижче наведено рисунок алгоритму формування бази знань та процесу векторизації (рис. А. 1.1).



Рис. А. 1.1 Алгоритм підготовки бази знань для RAG архітектури

1. Як згадувалось раніше дані бувають різних форм. Щоб правильно налаштувати конвеєр попередньої обробки, потрібно зрозуміти: які формати файлів є у наборі даних? Чи всі ваші дані в одному форматі, чи маємо справу з різними типами файлів або навіть типами даних? Виходячи з відповідей на ці запитання, можна вибрати правильні інструменти для вилучення даних, які містяться у файлах, та їх об'єднання для подальшої обробки.

Продовження додатка А

2. Мета попередньої обробки − зробити дані «готовими до RAG», зберігаючи при цьому інформацію, яку вони містять. Для цього корисно видаляти з даних специфічні для формату символи, які не містять інформації, такі як зайві пробіли, порожні рядки, вказані підрядки, регулярні вирази, заголовки та нижні колонтитули сторінок. Однак деякі з цих даних також можна перетворити на метадані, щоб зберегти важливу інформацію про структуру документа, яка допоможе пізніше.

3. Алгоритми та моделі обробки мови мають неявні налаштування щодо довжини текстових фрагментів, які вони можуть обробити. На цьому кроці розділяємо текстові дані на менші частини за допомогою налаштувань моделі Chunk_size, Chunk_Overlap, тобто врахування розміру фрагментів та зберігання контексту, щоб досягти оптимальної довжини. Існує багато стратегій фрагментації даних, які значною мірою визначаються методом індексації. Хоча раніше багато моделей вбудовування вимагали текстових фрагментів від 200 до 300 слів, зараз часто розглядаються моделі, які можуть обробляти більші частини тексту, такі як цілі сторінки PDF або навіть цілі багатосторінкові документи в налаштуваннях.

4. Термін «метадані» стосується міток, які розповідають більше про обрані дані, наприклад, коли було створено точку даних, хто її створив тощо. Під час процесу фрагментації можна витягти метадані з вихідних даних і додати їх до очищених і фрагментованих фрагментів. Наприклад, можна зберегти інформацію про те, з якої сторінки походить фрагмент, під яким заголовком він був опублікований або який фрагмент йде одразу після нього. Це додає контекст, який може бути дуже корисним для пошуку потрібної інформації пізніше.

Продовження додатка А

5. Індексування − це процес додавання даних до бази даних для їх легкого пошуку. Тому воно доповнює пошук даних. Для індексації даних можна використовувати розріджені або щільні методи. Найпопулярнішим розрідженим методом є BM25, алгоритм ранжування, про який згадав в перспективах розвитку чат-боту. Щільні методи використовують мовні моделі вбудовування.

Розріджені та щільні методи доповнюють один одного та часто використовуються разом у гібридній системі . Важливо використовувати той самий метод, який буде використано для пошуку даних пізніше. Що стосується метаданих, вони будуть індексовані разом із самими текстовими даними, тому можуть слугувати майбутнім фільтром або іншим засобом покращення контексту.

Також потрібно врахувати, що проіндексовати кілька документів на одній машині відносно легко, але виробничим системам часто потрібно швидко обробляти велику кількість файлів. У виробничому середовищі вам потрібно враховувати такі фактори, як пропускна здатність (скільки файлів можна обробити за певний час) та затримка (як швидко нова інформація стає доступною для пошуку). Системи індексації у виробничому середовищі часто використовують розподілені архітектури для вирішення цих проблем. Вони можуть ставити запити на індексацію в чергу та використовувати кілька машин для паралельної обробки файлів.

Додаток Б

Код підготовки та налаштування RAG архітектури

```
import os
from dotenv import load_dotenv
DEBUG = False
load_dotenv()
# Завантажені книги для використання в процесі доповнення (augmentation)
BOOK1 = "pdf_doc/Blockchain and Crypto Currency.pdf"
BOOK2 = "pdf_doc/Blockchain and Decentralized Finance (DeFi) Challenges and Future Potential.pdf"
BOOK3 = "pdf_doc/English_decentralized-finance.pdf"
BOOK4 = "pdf_doc/Finansovi tehnologii.pdf"
BOOK5 = "pdf_doc/Віртуальна валюта. Біткоїн та технологія блокчейн.pdf"
BOOK6 = "pdf_doc/Впровадження блокчейн-технологій в економіці України переваги та виклики.pdf"
BOOK7 = "pdf_doc/ВСТУП ДО ТЕХНОЛОГІЇ БЛОКЧЕЙН ТА КРИПТОВАЛЮТ.pdf"
BOOK8 = "pdf_doc/ЕКОНОМІЧНА ПРИРОДА ВІРТУАЛЬНИХ ВАЛЮТ.pdf"
BOOK9 = "pdf_doc/Криптовалюта як інвестиційний інструмент .pdf"
BOOK10 = "pdf_doc/МОДЕЛЮВАННЯ КОМПЛЕКСНОЇ ОЦІНКИ РИЗИКУ КІБЕРШАХРАЙСТВ ТА ЛЕГАЛІЗАЦІЇ КРИМІНАЛЬНИХ ДОХОДІВ .pdf"
BOOK11 = "pdf_doc/Прогнозування_криптовалют.pdf"
FAQ = "txt_faq/FAQ.txt"
BOOK_LIST = [BOOK1, BOOK2, BOOK3, BOOK4, BOOK5, BOOK6, BOOK7, BOOK8, BOOK9, BOOK10, BOOK11, FAQ]
# Параметри для поділу документів на частини та тип векторної моделі.
CHUNK_SIZE = 1000
CHUNK_OVERLAP = 50
MAX_DOCS_RETRIEVED = 7
VECTOR_STORE_NAME = "ORACLEDB"
EMBED_TYPE = "COHERE"
EMBED_COHERE_MODEL_NAME = "embed-multilingual-v3.0"
COHERE_EMBED_KEY = os.getenv("COHERE_EMBED_KEY")
#Додавання можливості повторного ранжування для майбутнього оновлення.
ADD_RERANKER = False
# конфігурація LLM
LLM_TYPE = "COHERE"
LLM_COHERE_MODEL_NAME = "command-r-08-2024"
COHERE_CHAT_KEY = os.getenv("COHERE_CHAT_KEY")
# максимальна, задана кількість токенів та поведінка системи (креативність).
MAX_TOKENS = 1000
TEMPERATURE = 0.3
```

Додаток В

Код основних елементів та функцій системи

```
#Завантаження бібліотек та фреймворків.
import streamlit as st
import oracledb
import re
import os
import cohere
from langchain_community.document_loaders import TextLoader, PyPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain.storage import LocalFileStore
from langchain_community.vectorstores import OracleVS
from langchain_core.runnables import RunnablePassthrough
from langchain_cohere import CohereEmbeddings
from langchain.embeddings import CacheBackedEmbeddings
from langchain.retrievers import ContextualCompressionRetriever
from langchain.retrievers.document_compressors import CohereRerank
from langchain_core.prompts import ChatPromptTemplate
from langchain.llms import Cohere
import oci
from langchain_community.llms.oci_generative_ai import OCIGenAI
from langchain_community.vectorstores.utils import DistanceStrategy
from langchain import hub
import time
# Обробка тексту після поділу на частини (очищення від спецсимволів, надлишкових пробілів)
def post_process(splits):
    for split in splits:
        split.page_content = split.page_content.replace("\n", " ")
        split.page_content = re.sub("[^a-zA-Z0-9 \n\.]", " ", split.page_content)
        split.page_content = " ".join(split.page_content.split())
    return splits
# Завантаження всіх сторінок з PDF або TXT документів
def load_all_pages(book_list):
    all_pages = []
    for book in book_list:
        print(f"Loading book: {book}...")
        if book.endswith(".pdf"):
            loader = PyPDFLoader(book)
        elif book.endswith(".txt"):
            loader = TextLoader(book, encoding="utf-8")
```

Продовження додатка В

```
            else:
                print(f"Unsupported file format for: {book}")
                continue
            try:
                pages = loader.load()
                all_pages.extend(pages)
                print(f"Loaded {len(pages)} pages from {book}")
            except Exception as e:
                print(f"✗ Failed to load {book}: {e}")
        return all_pages
    # Поділ сторінок на частини для обробки в RAG
    def split_in_chunks(all_pages):
        text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=CHUNK_SIZE, chunk_overlap=CHUNK_OVERLAP
        )
        splits = text_splitter.split_documents(all_pages)
        splits = post_process(splits)
        for split in splits:
            source = split.metadata.get("source_name", "Unknown")
            split.metadata["link"] = os.path.basename(source)
        print(f"Splitted the documents in {len(splits)} chunks...")
        print("Size of splits: " + str(text_splitter.__sizeof__()) + "bytes")
        return splits
    # Ініціалізація моделі створення ембеддінгів з кешем
    def create_cached_embedder():
        print("Initializing Embeddings model...")
        fs = LocalFileStore("./vector-cache/")
        if EMBED_TYPE == "COHERE":
            print("Loading Cohere Embeddings Model...")
            embed_model = CohereEmbeddings(
                model=EMBED_COHERE_MODEL_NAME,
cohere_api_key=COHERE_EMBED_KEY
            )
        embed_model.request_timeout = 5.0
        time.sleep(10)
        cached_embedder = CacheBackedEmbeddings.from_bytes_store(
            embed_model, fs, namespace=EMBED_COHERE_MODEL_NAME
        )
        return cached_embedder
```

Продовження додатка В

```
        # Створення механізму пошуку з опціональни, можливим повторним ранжуванням.
        def create_retriever(vectorstore):
            if ADD_RERANKER == False:
                print("No reranking...")
retriever = vectorstore.as_retriever(search_kwargs={"k": MAX_DOCS_RETRIEVED})
            else:
                print("Adding reranking to QA chain...")
                compressor = CohereRerank(cohere_api_key=COHERE_EMBED_KEY)
                base_retriever = vectorstore.as_retriever(
                    search_kwargs={"k": MAX_DOCS_RETRIEVED}
                )
                retriever = ContextualCompressionRetriever(
                    base_compressor=compressor, base_retriever=base_retriever
                )
            return retriever
        # Створення векторного сховища на основі Oracle Database
        def create_vector_store(store_type, document_splits, embedder):
            print(f"Indexing: using {store_type} as Vector Store...")
            # Перетворюємо Document об'єкти на рядки
            texts = [doc.page_content for doc in document_splits]
            metadatas = [doc.metadata for doc in document_splits]
            if store_type == "ORACLEDB":
                connection = oracledb.connect(
                    user="vector",
                    password="vector",
                    dsn="localhost/FREEPDB1"
                )
                vectorstore = OracleVS.from_texts(
                    texts=texts,
                    embedding=embedder,
                    client=connection,
                    table_name="oravs",
                    distance_strategy=DistanceStrategy.DOT_PRODUCT,
                    metadatas=metadatas
                )
            print("Index created.")
            return vectorstore
#Побудова LLM  на основі типу моделі (даний код адаптовано для COHERE)
        def build_llm(llm_type):
            print(f"Using {llm_type} llm...")
```

Продовження додатка В

```
        signer = make_security_token_signer(oci_config=oci_config)
        if llm_type == "OCI":
            print("Using OCI experimental integration with LangChain...")
            llm = OCIGenAI(
                model_id="cohere.command",
service_endpoint="https://inference.generativeai.uk-london 1.oci.oraclecloud.com",
compartment_id="ocid1.compartment.oc1..aaaaaaaaqdj2ou3o2e66laaddzsecdbrwc5jxqa
zdmdgpfp5ssvs7jymxyxq",
                model_kwargs={"max_tokens": 200},
                auth_type='SECURITY_TOKEN',
            )
        elif llm_type == "COHERE":
            llm = Cohere(
                model=LLM_COHERE_MODEL_NAME,
                cohere_api_key= COHERE_CHAT_KEY,
                max_tokens= MAX_TOKENS,
                temperature= TEMPERATURE,
            )
        return llm
    # Ініціалізація RAG-ланцюга — головної логіки системи
    @st.cache_resource
    def initialize_rag_chain():
        all_pages = load_all_pages(BOOK_LIST)
        document_splits = split_in_chunks(all_pages)
        embedder = create_cached_embedder()
        print("Size of splits---: " + str(document_splits.__sizeof__()) + "bytes")
        vectorstore = create_vector_store(VECTOR_STORE_NAME,
document_splits, embedder)
        retriever = create_retriever(vectorstore)
        llm = build_llm(LLM_TYPE)
        template = """Ви — експертний помічник з питань криптовалют,
блокчейну та децентралізованих фінансів (DeFi).
        {context}
        Зважаючи на наведений вище контекст, надайте відповідь на запитання
користувача. Якщо ви не можете знайти відповідь у наданому контексті, чесно
визнайте це, написавши: "На основі наявної інформації я не можу надати точну
відповідь на це запитання." НЕ ВИГАДУЙТЕ інформацію і не надавайте
фінансових порад.
        Question: {question}
        """
        rag_prompt = ChatPromptTemplate.from_template(template)
```

Продовження додатка В

```
    print("Building rag_chain...")
    rag_chain = (
        {"context": retriever, "question": RunnablePassthrough()} | rag_prompt |
llm
    )
    print("Init RAG complete...")
    return rag_chain
# Функція отримання відповіді від моделі
def get_answer(rag_chain, question):
    response = rag_chain.invoke(question)
    if DEBUG:
        print(f"Question: {question}")
        print("The response:")
        print(response)
        print()
        print("Sources:")
        for doc in response.get("source_documents", []):
            print(f" - {doc.metadata}")
    return str(response)
```

Додаток Г

Код вебінтерфейсу системи

```
#Завантаження необхідних бібліотек
import streamlit as st
import traceback
import sys
from init_rag_streamlit import initialize_rag_chain, get_answer
from streamlit_feedback import streamlit_feedback
# Функція для обробки зворотного зв'язку від користувача
def process_feedback(feedback_value):
    st.write("Processing feedback value:", feedback_value)
    try:
        with open("feedback.txt", "a", encoding="utf-8") as f:
            f.write(f"{feedback_value}\n")
        st.write("Feedback successfully written to file.")
    except Exception as e:
        st.error(f"Error writing to file: {e}")
        traceback.print_exc()
# Функція для скидання історії чату
def reset_conversation():
    st.session_state.messages = []
    st.session_state.feedback_rendered = False
    st.session_state.feedback_key = 0
st.title("AI-бот на базі RAG і Oracle Database для обізнаності в криптосфері")
st.button("Очистити історію чату", on_click=reset_conversation)
# Ініціалізація стану сесії
if "messages" not in st.session_state:
    reset_conversation()
# Ініціалізація RAG-ланцюга
rag_chain = initialize_rag_chain()
# Відображення попередніх повідомлень з історії чату
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])
# Введення питання користувачем і обробка відповіді
if question := st.chat_input("Привіт, як я можу тобі допомогти?"):
    st.chat_message("user").markdown(question)
    st.session_state.messages.append({"role": "user", "content": question})
    try:
        response = get_answer(rag_chain, question)
        with st.chat_message("assistant"):
```

```
                answer_text = response if isinstance(response, str) else getattr(response, "content", str(response))
                st.markdown(answer_text)
                 # Компонент для збору зворотного зв'язку
                with st.container():
                   st.markdown("Оцініть відповідь:")

                   if not st.session_state.feedback_rendered:
                      def _submit_feedback(feedback_value, *args, **kwargs):
                         print("Відгук надіслано:", feedback_value, file=sys.stderr)
                         st.write("Отримано відгук:", feedback_value)
                         process_feedback(feedback_value)
                         st.session_state.feedback_rendered = False

                      feedback_component = streamlit_feedback(
                         feedback_type="faces",
                         on_submit=_submit_feedback,
                         key=f"feedback_{st.session_state.feedback_key}",
                         optional_text_label="Будь ласка, надайте додаткову інформацію",
                         args=["✓"]
                      )
                      st.session_state.feedback_key += 1
                      st.session_state.feedback_rendered = True

             st.session_state.messages.append({"role": "assistant", "content": answer_text})

          except Exception as e:
             st.error(f"An error occurred: {e}")
             traceback.print_exc()
```