Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

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

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Метод генерації відповідей на основі RAG із використанням графа походження та оцінки довіри
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-22-2

Олександра ДРОГІНА
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки

(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект

(повна назва освітньої програми)

Керівник проф. Олександр ШЕВЧЕНКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Лариса ЧАЛА
(підпис) (власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

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

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Дрогіній Олександрі Леонідівні
(прізвище, ім'я, по батькові)

1. Тема роботи Метод генерації відповідей на основі RAG із використанням графа походження та оцінки довіри

затверджена наказом університету від 5 червня 20 26 р. № 488Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 7 липня 20 26 р.

3. Вихідні дані до роботи документація Python, документація Streamlit, документація LangChain, документація FAISS, документація NetworkX, наукові публікації з тематики Retrieval-Augmented Generation, графів походження та оцінювання довіри до відповідей великих мовних моделей, підручники зі штучного інтелекту та обробки природної мови, відкриті API веб-провайдерів (Wikipedia, OpenAlex, Google), тестовий корпус україномовних документів

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі
2) Постановка задачі та вимоги до системи
3) Метод генерації відповідей на основі RAG із графом походження та оцінкою довіри
4) Програмна реалізація та експериментальні дослідження

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.06.2026	виконано
2	Аналіз предметної галузі	09.06.2026	виконано
3	Постановка задачі та формування вимог	11.06.2026	виконано
4	Проєктування архітектури методу	13.06.2026	виконано
5	Реалізація модуля індексації	13.06.2026	виконано
6	Реалізація модуля векторного пошуку	14.06.2026	виконано
7	Реалізація модуля побудови графа походження	16.06.2026	виконано
8	Реалізація модуля генерації відповіді	17.06.2026	виконано
9	Реалізація модуля оцінки довіри	17.06.2026	виконано
10	Розробка інтерфейсу користувача	18.06.2026	виконано
11	Експериментальне дослідження системи	18.06.2026	виконано
12	Написання пояснювальної записки	20.06.2026	виконано
13	Перевірка на академічний плагіат	23.06.2026	виконано
14	Проходження нормоконтролю	24.06.2026	виконано
15	Підготовка презентації	25.06.2026	виконано
16	Попередній захист	28.06.2026	виконано
17	Рецензування	29.06.2026	виконано
18	Захист перед ЕК	07.07.2026	виконано

Дата видачі завдання 8 червня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Олександр ШЕВЧЕНКО
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 117 с., 14 рис., 8 табл., 5 дод., 27 джерел.

ВБУДОВУВАННЯ, ВЕБ-РЕТРІВАЛ, ВЕКТОРНИЙ ПОШУК, ВЕЛИКІ МОВНІ МОДЕЛІ, ГЕНЕРАЦІЯ ВІДПОВІДЕЙ, ГРАФ ПОХОДЖЕННЯ, ОЦІНКА ДОВІРИ, ПОЯСНЮВАНІСТЬ, RAG, STREAMLIT.

Об'єкт дослідження – процес генерації відповідей на запити користувача на основі великих мовних моделей із залученням зовнішніх джерел знань.

Предмет дослідження – метод генерації відповідей на основі підходу Retrieval-Augmented Generation з використанням графа походження документів та багатофакторної оцінки довіри.

Мета роботи – підвищення прозорості та надійності систем генерації відповідей шляхом розробки методу, що інтегрує RAG, граф походження документів і багатофакторну оцінку довіри, та його програмної реалізації.

Методи дослідження – аналіз наукової літератури, методи векторного представлення тексту, методи теорії графів, методи інженерії програмного забезпечення, експериментальний порівняльний аналіз.

Сучасні RAG-системи не надають користувачу структурованої інформації про джерела відповіді і не дозволяють оцінити її надійність. У роботі запропоновано метод, що поєднує Retrieval-Augmented Generation із графом походження документів і багатофакторною оцінкою довіри. Виконано аналіз предметної галузі, сформульовано постановку задачі та вимоги до системи. Експериментальне дослідження підтвердило доцільність використання графа походження та багатофакторної оцінки довіри для підвищення прозорості й інтерпретованості системи.

ABSTRACT

Bachelor’s thesis contains: 117 pp., 14 fig., 8 tabl., 5 ann., 27 references.

EMBEDDINGS, EXPLAINABILITY, LARGE LANGUAGE MODELS, PROVENANCE GRAPH, RAG, RESPONSE GENERATION, STREAMLIT, TRUST ESTIMATION, VECTOR SEARCH, WEB RETRIEVAL.

Object of research – the process of generating responses to user queries based on large language models with the involvement of external knowledge sources.

Subject of research – a response generation method based on the Retrieval-Augmented Generation approach using a document provenance graph and multi-factor trust estimation.

Purpose of the work – improving the transparency and reliability of response generation systems by developing a method that integrates RAG, a document provenance graph and multi-factor trust estimation, along with its software implementation.

Research methods – scientific literature analysis, vector text representation methods, graph theory methods, software engineering methods, experimental comparative analysis.

Modern RAG systems do not provide users with structured information about the sources of a response and do not allow its reliability to be assessed. This work proposes a method that combines Retrieval-Augmented Generation with a document provenance graph and multi-factor trust estimation. An analysis of the subject domain was carried out and the problem statement and system requirements were formulated. The experimental study confirmed the effectiveness of integrating a provenance graph and multi-factor trust estimation for improving the transparency and interpretability of the system.

ЗМІСТ

Вступ 10
1 Аналіз предметної галузі 14
1.1 Огляд підходів до генерації відповідей 14
1.2 Системи на основі великих мовних моделей 15
1.3 Метод Retrieval-Augmented Generation 16
1.3.1 Векторний пошук та моделі вбудовувань 18
1.4 Графи походження у системах генерації відповідей 19
1.4.1 Графи походження та семантичних зв’язків у RAG-системах . 20
1.5 Оцінювання довіри до відповідей мовних моделей 21
1.5.1 Основні підходи до оцінки довіри 22
1.6 Висновки за розділом 23
2 Постановка задачі та вимоги до системи 25
2.1 Формалізована постановка задачі 25
2.1.1 Вхідні дані системи 25
2.1.2 Вихідні дані системи 26
2.1.3 Умови якості результату 26
2.2 Функціональні вимоги 28
2.3 Нефункціональні вимоги 30
2.4 Обмеження та припущення 33
2.4.1 Припущення щодо якості вхідних даних 34
2.4.2 Припущення щодо середовища виконання 35
2.5 Огляд та обґрунтування вибору технологічного стека 35
2.6 Висновки за розділом 38
3 Метод генерації відповідей на основі RAG із графом походження та оцінкою довіри 39
3.1 Загальна архітектура методу 39
3.2 Індексація документів і векторний пошук 41
3.3 Ранжування та відбір кандидатів 45

3.4 Побудова графа походження та графове розширення контексту 51
3.5 Інтеграція веб-ретрівалу 54
3.5.1 Архітектура WebSearchClient 54
3.5.2 Підтримувані провайдери та стратегія вибору запиту 55
3.5.3 Ранжування та фільтрація веб-результатів 56
3.6 Генерація відповіді на основі контексту 58
3.7 Багатофакторна оцінка довіри 61
3.7.1 Перелік факторів довіри та їх ваги 61
3.7.2 Обчислення значень факторів 63
3.7.3 Уточнення оцінки за якістю відповіді 65
3.7.4 Якісні рівні довіри та інтерпретація 67
3.8 Висновки за розділом 68
4 Програмна реалізація та експериментальні дослідження 70
4.1 Структура програмного забезпечення 70
4.2 Опис модулів програмної реалізації 74
4.2.1 Модуль rag_engine.py 74
4.2.2 Модуль web_retrieval.py 77
4.2.3 Модуль nlp_utils.py 79
4.2.4 Модуль app.py 80
4.3 Інтерфейс користувача 82
4.4 Тестове середовище та набір запитів 92
4.4.1 Апаратне та програмне середовище 92
4.4.2 Склад тестового корпусу 94
4.4.3 Набір тестових запитів 96
4.4.4 Методика проведення випробувань 97
4.5 Порівняння режимів «з графом походження» та «без графа» 98
4.6 Аналіз поведінки оцінки довіри 100
4.7 Виявлені обмеження та напрями вдосконалення 102
4.8 Висновки за розділом 104
Висновки 106

Перелік джерел посилання 108
Додаток А Схема прийняття рішень 112
Додаток Б Загальна архітектура методу генерації відповідей 113
Додаток В Програмна реалізація індексації документів 114
Додаток Г Тестові запити 115
Додаток Д Відомість кваліфікаційної роботи 117

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – Artificial Intelligence – штучний інтелект;

API – Application Programming Interface – інтерфейс програмування застосунків;

BERT – Bidirectional Encoder Representations from Transformers – двонаправлена трансформерна модель для задач обробки природної мови;

Chunk – фрагмент тексту, отриманий під час розбиття документа для індексації у векторному сховищі;

Embedding – векторне представлення тексту, що кодує його семантику;

FAISS – Facebook AI Similarity Search – бібліотека для пошуку найближчих сусідів у векторних просторах;

GPT – Generative Pre-trained Transformer – генеративна попередньо навчена трансформерна модель;

IR – Information Retrieval – інформаційний пошук;

LLM – Large Language Model – велика мовна модель;

NLP – Natural Language Processing – обробка природної мови;

Provenance graph – граф походження та семантичних зв’язків – граф, що відображає ієрархічні та семантичні зв’язки між джерелами, документами й фрагментами тексту;

QA – Question Answering – система запитань і відповідей;

RAG – Retrieval-Augmented Generation – генерація відповідей із залученням зовнішніх джерел;

Retriever – компонент системи, що виконує пошук релевантних фрагментів документів за запитом користувача;

UI – User Interface – інтерфейс користувача;

XAI – Explainable Artificial Intelligence – пояснюваний штучний інтелект.

ВСТУП

Стрімкий розвиток великих мовних моделей (Large Language Models, LLM) суттєво змінив підходи до побудови інтелектуальних систем запитань і відповідей. Сучасні моделі здатні формувати зв’язні відповіді природною мовою на широке коло запитів, проте їх практичне застосування у відповідальних задачах ускладнене двома фундаментальними проблемами: схильністю до галюцинацій, тобто формування правдоподібних, але фактично невірних тверджень, та відсутністю в самій відповіді прозорого зв’язку з джерелами знань.

Одним з найбільш поширених способів подолання цих обмежень є метод Retrieval-Augmented Generation (RAG) [1], який поєднує мовну модель із зовнішнім модулем пошуку. У такій схемі модель формує відповідь не лише на основі власних параметрів, а й на основі релевантних фрагментів документів, отриманих за запитом користувача. Це підвищує фактичну точність відповідей і робить можливою прив’язку результату до конкретних джерел. Метод RAG активно застосовується у задачах побудови корпоративних асистентів, освітніх систем, інструментів наукового пошуку та діалогових систем, орієнтованих на роботу з актуальною й перевіреною інформацією.

Водночас базова реалізація RAG не вирішує задачі пояснюваності та оцінки довіри до відповіді. Користувач отримує лише текст, без структурованої інформації про те, які саме джерела й яким чином вплинули на результат, та без кількісної міри впевненості системи у згенерованій відповіді. Для практичних застосувань, зокрема у наукових та освітніх системах, де ціна помилки висока, ці аспекти є критичними. Перспективним напрямом розвитку є поєднання RAG з графом походження документів, у якому фіксуються джерела інформації, проміжні фрагменти контексту та зв’язки між ними, а також з багатофакторною оцінкою довіри, що враховує характеристики як етапу пошуку, так і самої відповіді. Такий підхід

дозволяє не лише підвищити надійність відповідей, а й забезпечити їх інтерпретованість для кінцевого користувача.

Незважаючи на широке поширення RAG-систем, питання прозорості та обґрунтованості відповідей залишається відкритим. Більшість наявних рішень обмежуються поверненням переліку документів, не пропонуючи структурованої моделі їх взаємозв'язків і не надаючи кількісної оцінки довіри до сформованої відповіді. Розробка методу, який інтегрує RAG із графом походження та багатофакторною оцінкою довіри, є актуальним науково-прикладним завданням, що відповідає сучасним напрямам досліджень у сферах інформаційного пошуку, обробки природної мови та пояснюваного штучного інтелекту.

Об'єктом дослідження є процес генерації відповідей на запити користувача на основі великих мовних моделей із залученням зовнішніх джерел знань.

Предметом дослідження є метод генерації відповідей на основі підходу Retrieval-Augmented Generation з використанням графа походження документів та багатофакторної оцінки довіри до згенерованої відповіді

Мета роботи – підвищення прозорості та надійності систем генерації відповідей шляхом розробки методу, що інтегрує RAG, граф походження документів і багатофакторну оцінку довіри, та програмної реалізації цього методу у вигляді інтерактивного застосунку.

Для досягнення поставленої мети у роботі сформульовано та розв'язано такі задачі:

– проаналізувати підходи до побудови систем генерації відповідей, дослідити архітектуру методу RAG, моделі векторного пошуку та вбудовувань;

– розглянути поняття графа походження документів і способи його застосування у RAG-системах, а також підходи до оцінювання довіри у системах на основі великих мовних моделей;

– сформулювати постановку задачі побудови методу генерації відповідей із графом походження та багатофакторною оцінкою довіри й визначити вимоги до програмної системи;

– розробити архітектуру методу, що поєднує локальний і веб-пошук, побудову графа походження над знайденим контекстом та обчислення кількісної оцінки довіри з можливістю її пояснення користувачу;

– реалізувати метод у вигляді Python-застосунку;

– провести експериментальне дослідження розробленої системи на наборі тестових запитів, оцінити вплив графового розширення контексту на якість відповіді та поведінку оцінки довіри;

– сформулювати висновки щодо ефективності запропонованого методу та визначити напрями його подальшого розвитку.

Було використано методи аналізу та синтезу наукової літератури з проблематики систем запитань і відповідей, методів інформаційного пошуку та обробки природної мови; методи векторного представлення тексту на основі трансформерних моделей вбудовувань; методи теорії графів для побудови та аналізу графа походження документів; методи інженерії програмного забезпечення для проєктування та реалізації застосунку; методи експериментального порівняльного аналізу для оцінки впливу графового розширення контексту на характеристики згенерованих відповідей.

У роботі запропоновано метод генерації відповідей, який, на відміну від типових реалізацій RAG, поєднує граф походження документів і багатофакторну оцінку довіри до відповіді. Зокрема:

– набули подальшого розвитку підходи до пояснення відповідей мовних моделей за рахунок використання графа походження, у якому одночасно фіксуються джерела, документи, фрагменти та їх семантичні зв’язки;

– запропоновано схему багатофакторної оцінки довіри, що інтегрує характеристики ретрівалу, структурні метрики графа походження й

властивості згенерованої відповіді у вигляді зваженої суми нормалізованих факторів, що дозволяє інтерпретувати оцінку як з кількісного, так і з якісного боку;

– реалізовано режим порівняльного оцінювання роботи системи «з графом походження» та «без графа», що дає змогу експериментально оцінити внесок графового розширення у фінальну якість відповіді;

– розроблено інтерфейс користувача, орієнтований як на дослідника, що аналізує внутрішні метрики системи, так і на кінцевого користувача, якому потрібні лише відповідь і зрозумілий індикатор довіри;

– продемонстровано можливість поєднання локальних та хмарних мовних моделей у межах єдиного пайплайна, що забезпечує гнучкий баланс між вимогами до якості, продуктивності та конфіденційності.

Розроблений застосунок може бути використаний як інструмент для роботи з локальними та веб джерелами знань у наукових, освітніх і прикладних задачах, де важлива не лише сама відповідь, а й можливість її верифікації. Реалізована архітектура є модульною і може бути адаптована до різних предметних областей шляхом заміни корпусу документів та переліку веб провайдерів, а також до різних обчислювальних умов – від персонального комп’ютера до віддаленого сервера. Підхід до оцінки довіри як зваженої суми факторів може бути використаний як самостійний компонент в інших RAG системах, де потрібен прозорий індикатор надійності відповіді. У перспективі це створює передумови для побудови цілого класу пояснюваних систем генерації, у яких походження інформації та рівень довіри стають невід’ємною частиною взаємодії з користувачем.

Запропоновані рішення зберігають сумісність із типовими компонентами RAG пайплайнів, тому окремі модулі – зокрема граф походження та оцінка довіри – можуть інтегруватися в уже існуючі системи з мінімальними змінами. Отриманий застосунок може також слугувати навчальним та експериментальним стендом для дослідження поведінки RAG систем за зміни моделей, джерел даних чи гіперпараметрів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

У межах цього розділу розглянуто сучасний стан проблеми генерації відповідей на запити користувача за допомогою інтелектуальних систем. Основну увагу приділено еволюції підходів від класичних систем запитань і відповідей до моделей на основі великих мовних моделей, архітектурі Retrieval-Augmented Generation, поняттю графа походження документів та методам оцінювання довіри до відповідей, що генеруються мовними моделями.

1.1 Огляд підходів до генерації відповідей

Класичні системи запитань і відповідей (Question Answering, QA) розвивалися як інструмент для отримання короткої точної відповіді на запит, сформульований природною мовою. Історично виділяють два основних типи таких систем.

Перший тип – інформаційно-пошукові (IR-based) системи. У них пошук відповіді зводиться до знаходження у корпусі документів того фрагмента, який з найвищою імовірністю містить шукану інформацію. Користувачу повертається або весь фрагмент, або речення, виокремлене за допомогою правил чи статистичних евристик.

Другий тип – екстрактивні моделі на основі нейронних мереж. Такі моделі приймають на вхід пару «запит – документ» і повертають діапазон символів у документі, що відповідає шуканій відповіді. Найбільш відомі реалізації базуються на трансформерних архітектурах, попередньо навчених на великих корпусах текстів.

Перевагою класичних QA-систем є висока керованість і можливість прямого посилання на джерело: будь-яку відповідь можна співвіднести з конкретним документом і фрагментом тексту. Проте такі системи мають низку суттєвих обмежень. Відповідь обмежена тим, що дослівно присутнє у

тексті, складно агрегувати інформацію з кількох документів одночасно, неможливо переформулювати чи узагальнити вилучені фрагменти, а робота з багатомовними або параметрично складними запитами потребує окремих, часто спеціалізованих рішень.

1.2 Системи на основі великих мовних моделей

Поява великих мовних моделей на основі архітектури Transformer [2], наприклад, BERT [3], GPT[4], LLaMA[5] та інших, суттєво змінила підходи до побудови QA-систем. На відміну від класичних екстрактивних моделей, такі моделі здатні генерувати відповідь природною мовою, агрегувати інформацію з кількох фрагментів, переформулювати її та враховувати контекст попереднього діалогу.

Водночас практичне використання великих мовних моделей у відповідальних задачах ускладнюється кількома характерними обмеженнями. Першим є проблема галюцинацій [6]: модель може формулювати правдоподібні, але фактично невірні твердження, особливо у випадках, коли запит виходить за межі її тренувального розподілу. Другим обмеженням є відсутність явних джерел у згенерованому тексті, що ускладнює його верифікацію та використання у наукових і навчальних контекстах. Третім є застарілість знань, оскільки модель навчається на фіксованому зрізі даних і не має прямого доступу до актуальної інформації. Четвертим – обмежена прозорість, тобто неможливість безпосередньо визначити, на основі яких саме фрагментів навчальних даних сформовано конкретну відповідь. Крім того, більшість LLM-систем не надає користувачу кількісної оцінки довіри до відповіді та явного відстеження використаних джерел. Це ускладнює їх застосування в задачах, де необхідні прозорість і верифіковність результатів. Узагальнене порівняння класичних QA-систем, підходу на основі LLM та архітектури RAG наведено на рисунку 1.1.

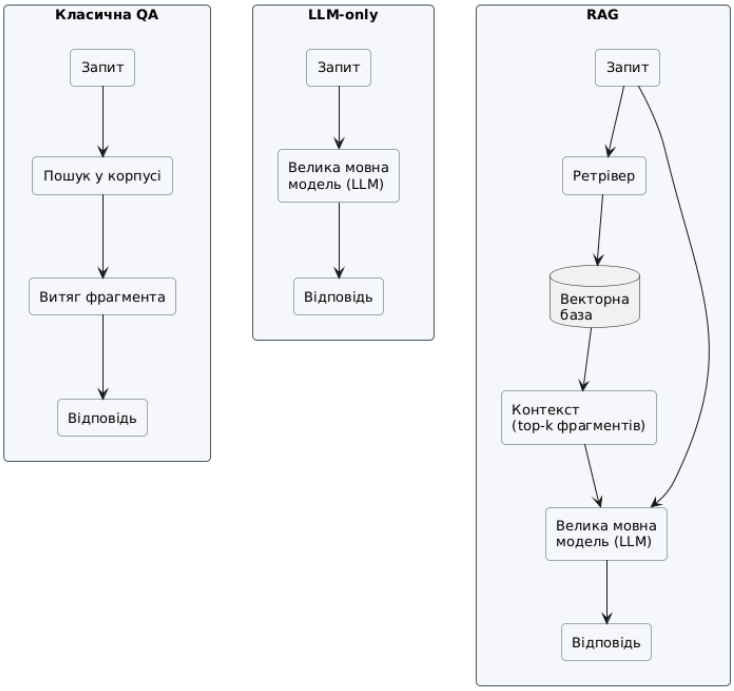


Рисунок 1.1 – Узагальнені архітектури систем генерації відповідей: класична QA, підхід на основі LLM та RAG

Зазначені обмеження стали підставою для розвитку гібридних архітектур, у яких велика мовна модель використовується разом із зовнішнім модулем пошуку, що надає актуальний і верифіковний контекст. Найпоширенішою такою архітектурою є Retrieval-Augmented Generation.

1.3 Метод Retrieval-Augmented Generation

Метод Retrieval-Augmented Generation, запропонований у роботах П. Левіса та співавторів [1], передбачає поєднання двох основних компонентів: ретрівера, що виконує пошук релевантних фрагментів у зовнішньому корпусі, та генератора, яким зазвичай є попередньо навчена велика мовна модель.

Узагальнено робота RAG-системи може бути описана такою послідовністю операцій. Користувач формулює запит природною мовою. Ретрівер перетворює запит у векторне представлення і знаходить у

векторній базі найрелевантніші фрагменти документів. Знайдені фрагменти разом із вихідним запитом передаються до мовної моделі як контекст. Модель формує відповідь, що ґрунтується одночасно на запиті та переданому контексті, після чого результат повертається користувачу. Узагальнену архітектуру методу Retrieval-Augmented Generation наведено на рисунку 1.2.

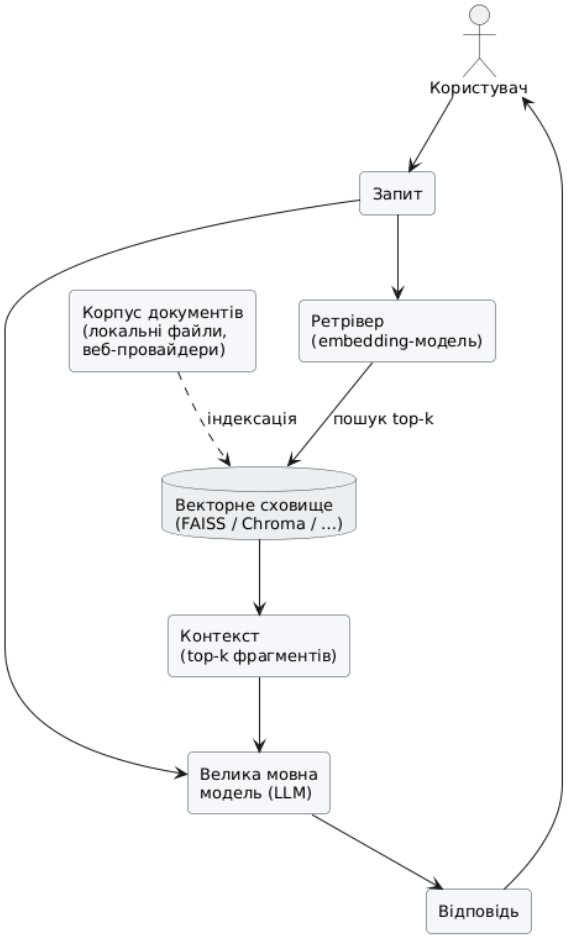


Рисунок 1.2 – Узагальнена архітектура методу RAG

Перевагами такого підходу є можливість оновлення знань без перенавчання моделі (для цього достатньо змінити корпус документів), потенційна можливість посилання на джерела та зменшення кількості галюцинацій за рахунок прив’язки відповіді до конкретних фрагментів. Крім того, RAG-архітектура природно поєднується з різними типами

джерел: локальними документами користувача, корпоративними базами знань, а також веб-ресурсами.

1.3.1 Векторний пошук та моделі вбудовувань

Ключовим елементом RAG є векторний пошук. Документи попередньо розбиваються на фрагменти (chunks) певної довжини, після чого кожен фрагмент перетворюється у щільне векторне представлення за допомогою моделі вбудовувань (embedding model). Найчастіше використовуються трансформерні моделі сімейств sentence-transformers[7], bge, e5 та інші, орієнтовані саме на побудову якісних семантичних представлень коротких текстових фрагментів.

Отримані вектори зберігаються у спеціалізованих векторних сховищах, таких як FAISS[8], Chroma, Qdrant, Pinecone тощо. Під час обробки запиту виконується аналогічне векторне перетворення запиту, після чого у сховищі здійснюється пошук k найближчих сусідів за обраною метрикою подібності, зазвичай косинусною або скалярним добутком, цей підхід відомий у літературі як dense retrieval [9]. Косинусна подібність між запитом *q* і фрагментом *d* обчислюється за формулою (1.1):

$$\mathrm{sim}(q, d) = \cos(e_q, e_d) = \frac{e_q \cdot e_d}{\|e_q\| \cdot \|e_d\|}, \qquad (1.1)$$

де e_q та e_d – вектори вбудовувань (embeddings) запиту і фрагмента відповідно;

$\|\cdot\|$ – евклідова норма вектора.

На основі обчислених значень подібності, отриманих за (1.1), формується множина top-*k* фрагментів, що передаються до мовної моделі як контекст (1.2):

$$D_k(q) = Top\,K_{d \in D}\, sim(q, d)\,, \quad (1.2)$$

де D – повна множина проіндексованих фрагментів корпусу;

TopK повертає k елементів d з найбільшими значеннями sim(q, d).

Якість ретрівалу безпосередньо впливає на якість фінальної відповіді. До основних проблем векторного пошуку у RAG належать чутливість до якості розбиття документів на фрагменти, залежність від якості embedding-моделі та її здатності працювати з відповідною мовою і предметною областю, а також необхідність балансу між кількістю переданих фрагментів і обмеженнями контекстного вікна великої мовної моделі. Окремим викликом є робота з багатомовними запитами, коли запит формулюється однією мовою, а доступні джерела – іншою.

1.4 Графи походження у системах генерації відповідей

Походження даних (data provenance) у широкому значенні являє собою інформацію про те, звідки походить певна одиниця даних, як вона була отримана, які проміжні перетворення над нею виконувалися та які інші дані вплинули на її формування[10]. Поняття активно використовується у системах управління базами даних, наукових обчисленнях, обробці потоків даних та системах прийняття рішень.

У контексті інтелектуальних систем походження дозволяє відповісти на питання щодо того, на основі яких саме документів сформовано конкретну відповідь, які з джерел є найбільш впливовими та які зв’язки існують між використаними фрагментами. Усе це є основою для побудови систем пояснюваного штучного інтелекту (Explainable AI), у яких прозорість процесу формування результату має не менше значення, ніж сам результат [11].

1.4.1 Графи походження та семантичних зв’язків у RAG-системах

У задачах генерації відповідей походження зручно представити у вигляді графа походження. Вузлами такого графа є джерела, документи та їх фрагменти, а ребрами – зв’язки між ними: належність фрагмента до документа, належність документа до провайдера або корпусу, тематична близькість між фрагментами тощо (рисунок 1.3). У подальшому під терміном «граф походження» в цій роботі мається на увазі саме «граф походження та семантичних зв’язків», тобто структура, яка одночасно відображає шлях проходження інформації від джерел до відповіді та семантичну спорідненість між фрагментами тексту.

Граф походження у RAG може використовуватися щонайменше у двох аспектах. Перший аспект – пояснення: візуалізація графа дозволяє показати, які саме фрагменти та джерела вплинули на формування відповіді, які з них були використані безпосередньо, а які додані за рахунок розширення контексту. Це створює основу для побудови інтерпретованого інтерфейсу системи. Другий аспект – розширення контексту: за допомогою графових зв’язків можна додавати до базової вибірки ретрівера фрагменти, що структурно або семантично пов’язані з найрелевантнішими, що особливо корисно у випадку коротких або неоднозначних запитів.

Окремою перевагою графового представлення є можливість використання структурних метрик, таких як центральність вузлів, для оцінки того, наскільки контекст є узгодженим і збалансованим. Ці метрики можуть бути безпосередньо враховані у формулі оцінки довіри до відповіді. а відміну від простого списку фрагментів, така структура фіксує не лише «що було знайдено», а й «як саме ці фрагменти пов’язані між собою та з джерелами». Завдяки цьому граф походження та семантичних зв’язків виступає не лише засобом візуалізації, а й інструментом для керування контекстом і довірою в Retrieval-Augmented Generation системі.

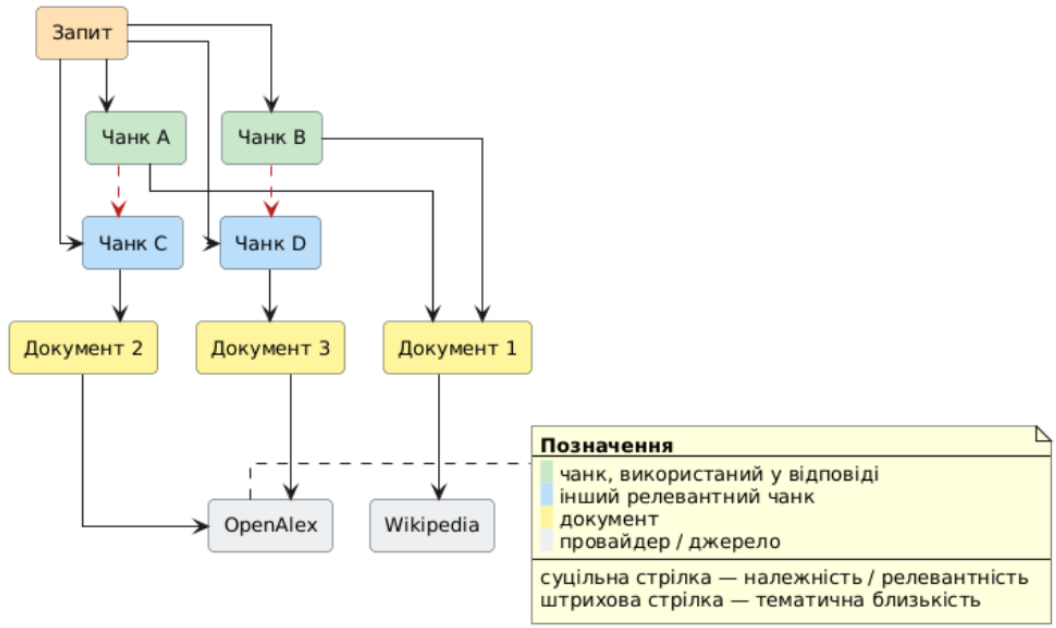


Рисунок 1.3 – Приклад структури графа походження документів у RAG-системі

1.5 Оцінювання довіри до відповідей мовних моделей

Однією з ключових проблем у застосуванні великих мовних моделей є їх здатність формувати відповіді з високою лінгвістичною впевненістю незалежно від того, наскільки фактично коректним є їх зміст. Для користувача такі відповіді виглядають однаково переконливими, що ускладнює критичну оцінку інформації, особливо у випадках, коли користувач не має достатньої експертизи у відповідній предметній галузі.

З огляду на це, у системах RAG особливого значення набуває оцінка довіри до згенерованої відповіді. Її основними задачами є: надати користувачу кількісний орієнтир щодо надійності інформації; дозволити системі самостійно ухвалювати рішення про звернення до додаткових джерел або відмову у відповіді у випадку слабкого контексту; забезпечити можливість порівняння різних режимів роботи системи, зокрема – із застосуванням графа походження та без нього.

1.5.1 Основні підходи до оцінки довіри

У літературі можна виділити кілька основних груп підходів до оцінки довіри у системах на основі великих мовних моделей. До імовірнісних методів належать ті, що ґрунтуються на ймовірностях токенів, які видає модель, та її логітах. Такі методи природно інтегруються з самою моделлю, проте вимагають доступу до її внутрішніх характеристик, що не завжди є можливим у випадку використання моделей через зовнішні API.

Методи самосумісності (self-consistency) базуються на порівнянні кількох незалежно згенерованих відповідей на той самий запит: висока узгодженість відповідей трактується як ознака впевненості моделі [12]. Цей підхід є відносно простим у реалізації, проте суттєво підвищує обчислювальні витрати.

Методи калібрування (calibration) спрямовані на зіставлення внутрішньої впевненості моделі з фактичною точністю на еталонних наборах [13]. Вони дозволяють отримати більш достовірні оцінки, проте вимагають наявності розмічених даних і регулярного перенавчання.

Окремою групою є багатофакторні евристичні оцінки [14], що поєднують характеристики ретрівалу (середня релевантність, стабільність score, покриття контексту), графові метрики (центральність вузлів, обережність графового розширення) та властивості самої відповіді(узгодженість із контекстом, наявність явних посилань, лексична підкріпленість контекстом).

У загальному вигляді багатофакторну оцінку довіри C до відповіді можна записати як зважену суму нормалізованих факторів (1.3):

$$C = \sum_{i=1^n} w_i \cdot f_i\,, \qquad \sum_{i=1^n} w_i = 1, \qquad f_i \in [0,1], \tag{1.3}$$

де f_i – нормалізоване значення і-го фактора (наприклад, середня релевантність ретрівалу, стабільність score, покриття контексту, центральність вузлів у графі походження, узгодженість відповіді з контекстом тощо);

w_i – його ваговий коефіцієнт;

n – загальна кількість факторів;

Підсумкове значення C інтерпретується як скалярна оцінка довіри у діапазоні [0, 1] і додатково розбивається на якісні рівні (низька / обережна / помірна / висока).

Для прикладних RAG-систем саме багатофакторний підхід згідно з формулою 1.3 є найбільш практично виправданим, оскільки він не вимагає додаткового навчання, легко інтерпретується і добре поєднується з графом походження: окремі фактори можуть напряму спиратися на властивості графа й використовуватися як для розрахунку числового значення довіри, так і для формування пояснення для користувача.

1.6 Висновки за розділом

У результаті аналізу предметної галузі встановлено сучасний стан та основні тенденції розвитку інтелектуальних систем генерації відповідей на запити користувача.

Розглянуто еволюцію підходів від класичних інформаційно-пошукових та екстрактивних QA-систем до рішень на основі великих мовних моделей. Визначено, що класичні підходи забезпечують високу керованість і можливість прямого посилання на джерела, проте мають суттєві обмеження: відповідь обмежена дослівно присутнім у тексті, агрегація інформації з кількох документів і переформулювання фрагментів є практично недосяжними. Великі мовні моделі усувають ці обмеження, проте породжують нові: схильність до галюцинацій, відсутність явних

джерел у відповіді, застарілість знань та обмежена прозорість формування результату.

Проаналізовано архітектуру методу Retrieval-Augmented Generation як перспективного способу поєднання переваг мовних моделей з контролем за джерелами. Досліджено математичний апарат векторного пошуку — формалізовано обчислення косинусної подібності між запитом і фрагментом та операцію формування множини top-k кандидатів. Встановлено основні проблеми ретрівалу: чутливість до якості розбиття документів на фрагменти, залежність від характеристик моделі вбудовувань, обмеження контекстного вікна мовної моделі та робота з багатомовними запитами.

Розглянуто поняття походження даних і концепцію графа походження у RAG-системах. Показано, що граф походження виконує дві ключові функції: забезпечує пояснення сформованої відповіді через візуалізацію задіяних джерел і зв'язків між ними та дозволяє структурно розширювати контекст за рахунок семантично або послідовно пов'язаних фрагментів. Окремо встановлено, що структурні метрики графа, зокрема центральність вузлів, можуть бути безпосередньо використані в оцінці довіри.

Систематизовано підходи до оцінювання довіри до відповідей мовних моделей: імовірнісні методи, методи самосумісності, методи калібрування та багатофакторні евристичні оцінки. Обґрунтовано, що для прикладних RAG-систем найбільш доцільним є багатофакторний підхід у вигляді зваженої суми нормалізованих факторів, що не потребує додаткового навчання, легко інтерпретується і природно інтегрується з графом походження.

Отримані результати формують теоретичне підґрунтя для постановки задачі та розробки методу генерації відповідей на основі RAG із використанням графа походження документів і багатофакторної оцінки довіри, що є предметом наступного розділу.

2 ПОСТАНОВКА ЗАДАЧІ ТА ВИМОГИ ДО СИСТЕМИ

У межах цього розділу сформульовано формалізовану постановку задачі побудови методу генерації відповідей на основі Retrieval Augmented Generation із використанням графа походження документів та багатофакторної оцінки довіри. Визначено функціональні та нефункціональні вимоги до програмної системи, описано обмеження та припущення, прийняті під час проєктування, а також обґрунтовано вибір технологічного стека реалізації.

2.1 Формалізована постановка задачі

Задача, що розв'язується у цій роботі, відноситься до класу задач генерації природномовних відповідей на запити користувача з можливістю верифікації результату за джерелами та кількісної оцінки надійності отриманої інформації. На відміну від класичного підходу до побудови RAG-систем, де виходом є лише текстова відповідь, у даній роботі система повертає розширений результат, що складається з трьох взаємопов'язаних компонентів: самої відповіді, структурованої інформації про джерела у вигляді графа походження та скалярної оцінки довіри.

2.1.1 Вхідні дані системи

Вхідними даними є запит користувача q, сформульований природною мовою. Запит може бути будь-якого типу: означальним («що таке...»), фактологічним («скільки...», «хто...»), переліковим («які є...»), процесним («як...») або загальним. Окрім запиту, система приймає корпус локальних документів D, кожен з яких є текстовим файлом у форматі .txt, .md, .pdf або .docx. Корпус може бути порожнім, якщо система працює виключно у веб-режимі. Додатково задається множина активних

веб-провайдерів P (Wikipedia, OpenAlex, arXiv, Semantic Scholar, Bing, Google), обраних користувачем. Нарешті, користувач задає два числових параметри пошуку: $k \in \{1,\ldots,10\}$ – кількість базових фрагментів-кандидатів, що повертає векторний ретрівер на першому кроці, та $hops \in \{0,1,2,3\}$ – глибина обходу графа походження при розширенні контексту.

2.1.2 Вихідні дані системи

Результатом роботи системи є три взаємопов'язані об'єкти. По-перше, відповідь a – природномовний текст, сформований на основі знайденого контексту або вилучений з нього екстрактивним способом. По-друге, оцінка довіри $C \in [0, 1]$ – скалярна величина, обчислена як зважена сума восьми нормалізованих факторів відповідно до формули (1.3); вона додатково перетворюється на якісний рівень (низька, обережна, помірна, висока). По-третє, граф походження $G = (V, E)$, де вузлами V є джерела, документи та їх фрагменти, а ребрами E – зв'язки між ними: послідовні, тобто, фрагменти одного документа, що йдуть один за одним, та семантичні, тобто, фрагменти різних документів, між якими виявлено тематичну схожість.

2.1.3 Умови якості результату

Коректність роботи системи визначається одночасним виконанням трьох умов. Перша умова – відповідь a ґрунтується виключно на фрагментах, присутніх у контексті, переданому до моделі генератора: система не «вигадує» інформацію поза наданими джерелами. Друга умова – оцінка довіри C коректно відображає ступінь підкріпленості відповіді знайденими фрагментами: чим більше прямих текстових підтверджень знайдено і чим вищою є їх релевантність, тим вища C. Третя умова – граф

походження G містить записи для всіх фрагментів, що увійшли до фінального контексту, із зазначенням повного шляху їх отримання – чи то прямим векторним пошуком, чи то графовим розширенням із зазначенням seed-фрагмента, типу ребра та відстані в хопах.

Окремим технічним обмеженням є максимально допустима сумарна довжина фрагментів, що передаються до мовної моделі генератора. Це обмеження обумовлено розміром контекстного вікна локальних моделей і задається параметром max_context_chars класу RAGConfig. У поточній реалізації значення цього параметра становить 2200 символів і є налаштовуваним: у разі використання хмарної моделі OpenAI обмеження може бути збільшено відповідно до її контекстного вікна.

Залежно від того, які джерела знань задіяні, система підтримує три режими роботи. У локальному режимі ($D \neq \emptyset$, $P = \emptyset$) пошук виконується лише у векторному індексі локальних документів із застосуванням графового розширення контексту на основі попередньо побудованого графа походження. У веб-режимі ($D = \emptyset$, $P \neq \emptyset$) пошук виконується виключно через зовнішні веб-провайдери із множини P; граф походження будується динамічно над результатами пошуку безпосередньо під час обробки запиту. У комбінованому режимі ($D \neq \emptyset$, $P \neq \emptyset$) виконуються обидва типи пошуку, результати об'єднуються, дедублікуються та ранжуються за релевантністю; фінальна оцінка довіри обчислюється як зважена комбінація оцінок локального та веб ретрівалу, де вага кожної складової визначається часткою її внеску у підсумковий контекст.

Загалом, постановку задачі можна сформулювати так: побудувати програмну систему, що реалізує функцію перетворення вхідного запиту і доступних джерел знань на відповідь, граф походження та оцінку довіри, відповідно до умов якості та технічних обмежень, описаних вище. Архітектура та алгоритми реалізації цієї функції детально розглядаються у розділі 3.

2.2 Функціональні вимоги

Функціональні вимоги визначають перелік операцій та поведінку системи, які вона зобов'язана забезпечити для досягнення поставленої мети. Вимоги сформульовано на основі аналізу функціональності застосунку та структури його модулів; їх узагальнення наведено в таблиці 2.1.

Таблиця 2.1 – Функціональні вимоги до системи

№	Вимога	Параметри/обмеження
1	Завантаження локальних документів	.txt, .md, .pdf, .docx
2	Побудова графа походження	Послідовні та семантичні ребра
3	Вибір режиму пошуку	Локальний / Веб / Комбінований
4	Вибір моделі-генератора	TinyLlama, Qwen, OpenAI gpt-4o-mini
5	Параметри запиту k та hops	$k \in \{1…10\}$, $hops \in \{0…3\}$
6	Веб-ретрівал	7 провайдерів, авто-переклад запиту
7	Ранжування та фільтрація	Комбінований score, обрізка контексту
8	Генерація відповіді	Екстрактивна або генеративна
9	Оцінка довіри	8 факторів, $C \in [0,1]$, 4 якісних рівні
10	Візуалізація графа походження	Інтерактивна PyVis-мережа [15]
11	Режим порівняння	З графом vs без графа
12	Історія запитів у сесії	Streamlit session state [16]

Вимога 1. Завантаження та обробка локальних документів. Система повинна підтримувати завантаження документів у форматах .txt, .md, .pdf та .docx. Для кожного завантаженого документа виконується витяг текстового вмісту, розбиття на фрагменти фіксованої довжини з перекриттям, векторизація фрагментів та збереження їх у векторному індексі FAISS. Кожному фрагменту присвоюється унікальний ідентифікатор у

форматі {джерело}_{порядковий_номер}, що використовується для побудови графа походження та формування записів провенансу.

Вимога 2. Побудова графа походження над локальним корпусом. Після індексації документів система повинна автоматично будувати граф походження, у якому вузлами є проіндексовані фрагменти, а ребрами – послідовні зв'язки між сусідніми фрагментами одного документа та семантичні зв'язки між фрагментами різних документів, косинусна подібність між якими перевищує визначений поріг. Граф зберігається у пам'яті та доступний для подальшого використання під час обробки запитів.

Вимога 3. Вибір режиму пошуку. Система повинна підтримувати три режими пошуку: локальний (пошук виключно у векторному індексі локального корпусу), веб (пошук через зовнішні провайдери) та комбінований (одночасне використання обох джерел). Режим обирається користувачем у сайдбарі інтерфейсу і впливає на весь подальший процес обробки запиту.

Вимога 4. Вибір моделі для генерації відповіді. Система повинна підтримувати щонайменше три варіанти моделі генератора: локальну модель TinyLlama, локальну модель Qwen та хмарну модель OpenAI. Вибір моделі здійснюється користувачем до ініціалізації бази знань. Для хмарної моделі система зчитує API ключ зі змінної середовища, за відсутності ключа виконується повернення відповідної помилки з описом необхідних дій.

Вимога 5. Виконання запиту з підтримкою параметрів k та hops. Під час обробки запиту користувач повинен мати можливість задати кількість базових фрагментів-кандидатів k (від 1 до 10) та глибину обходу графа hops. При hops = 0 графове розширення не виконується і система працює як базовий RAG без графа. При hops > 0 до контексту додаються сусідні вузли графа у межах заданої глибини.

Вимога 6. Веб-ретрівал із підтримкою кількох провайдерів. У веб-режимі та комбінованому режимі система повинна підтримувати одночасний пошук через декілька обраних провайдерів: Wikipedia,

OpenAlex, arXiv, Semantic Scholar, Bing та Google. Для кожного провайдера застосовується відповідний метод отримання результатів. Для академічних провайдерів та комерційних пошукових систем (Bing, Google) запит автоматично перекладається на англійську мову за допомогою моделі OpenAI або передається у вихідному вигляді. Провайдери Bing та Google потребують наявності відповідних API-ключів у змінних середовища.

Вимога 7. Ранжування та фільтрація результатів**.** Результати, отримані від векторного ретрівера та веб-провайдерів, повинні бути переранжовані з урахуванням комбінованого score, що поєднує векторну подібність, лексичне перекриття з запитом та евідентний score відповідності фрагмента запиту. Результати з низьким score повинні відфільтровуватися до передачі у модель-генератор. Контекст обрізається до max_context_chars символів із збереженням найрелевантніших фрагментів.

Вимога 8. Генерація відповіді. Система повинна формувати відповідь одним з двох способів залежно від типу запиту: екстрактивним (вилучення найрелевантнішого речення або переліку безпосередньо з контексту для запитів типу definition, list, factoid, yes_no) або генеративним (передача контексту і запиту до мовної моделі з відповідним промптом для запитів типу process та general). Для означальних запитів пріоритет надається фрагментам із джерел Wikipedia, якщо такі присутні у контексті.

Узагальнений перелік функціональних вимог наведено у таблиці 2.1.

2.3 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи, що не пов'язані безпосередньо з конкретними функціями, але суттєво впливають на її практичну придатність, надійність та можливість подальшого розвитку. У цьому підрозділі розглянуто вимоги до продуктивності, надійності, розширюваності, зручності використання та портативності системи.

Вимога до продуктивності. Система повинна забезпечувати прийнятний час відповіді в умовах типового тестового середовища. У локальному режимі з використанням локальної мовної моделі час обробки одного запиту з урахуванням векторного пошуку, графового розширення, також обчислення оцінки довіри та генерації відповіді не повинен перевищувати 30 секунд. Час відповіді у веб-режимі додатково залежить від затримок зовнішніх API і може сягати 15–20 секунд лише на отримання результатів від усіх обраних провайдерів. При використанні хмарної моделі OpenAI час генерації відповіді суттєво скорочується порівняно з локальними моделями завдяки апаратній інфраструктурі хмарного провайдера. Первинне завантаження локальної моделі та побудова векторного індексу є одноразовими операціями і не враховуються при оцінці часу обробки запитів.

Вимога до надійності. Система повинна коректно обробляти відмови зовнішніх джерел. У разі недоступності окремого веб-провайдера (тайм-аут, помилка мережі або відсутність API-ключа) виконання запиту не переривається: система продовжує роботу з результатами решти провайдерів. Кожен метод звернення до зовнішнього API огорнутий у блок обробки виключень, що повертає порожній список результатів у разі будь-якої помилки. Таким чином, деградація функціональності є частковою і керованою: у найгіршому випадку система повертає відповідь на основі лише тих джерел, що залишились доступними. Окремо передбачена перевірка наявності API-ключів перед зверненням до Bing та Google: за відсутності ключа відповідні провайдери пропускаються із виведенням інформаційного повідомлення у консоль.

Вимога до розширюваності. Архітектура системи повинна допускати додавання нових компонентів без переписування існуючих. Додавання нового веб провайдера зводиться до реалізації одного методу та реєстрації його назви. Додавання нового фактора оцінки довіри зводиться до оголошення нового об'єкта та реалізації відповідної логіки обчислення.

Заміна моделі вбудовувань або векторного сховища потребує зміни лише у методі ініціалізації, оскільки решта коду взаємодіє з ними через уніфікований інтерфейс. Така модульна організація коду знижує вартість внесення змін і мінімізує ризик порушення існуючої функціональності.

Вимога до зручності використання. Система повинна бути доступною для використання без спеціальних технічних знань у галузі машинного навчання. Всі параметри налаштування виносяться у бокову панель інтерфейсу з підписами та значеннями за замовчуванням, що відповідають типовому сценарію використання. Система повинна надавати користувачу зрозумілі повідомлення у разі некоректних дій: спроби надіслати запит до незапущеної бази знань, вибір веб-режиму без обраних провайдерів або завантаження документа непідтримуваного формату. Оцінка довіри виводиться одночасно у числовому форматі (C = 0.732) та якісному (помірна), що робить її зрозумілою незалежно від рівня підготовки користувача. Додатково інтерфейс надає користувачу доступ до проміжних артефактів роботи системи – списку знайдених фрагментів, графа походження та розкладу оцінки довіри на фактори, що підсилює прозорість і керованість взаємодії.

Вимога до портативності. Система повинна запускатися на будь-якій платформі, де доступний інтерпретатор Python версії 3.10 і вище, без прив'язки до конкретної операційної системи чи апаратної конфігурації. Усі залежності зафіксовані у файлі requirements.txt, що забезпечує відтворюваність середовища виконання. Локальні мовні моделі завантажуються автоматично з репозиторію HuggingFace під час першого запуску і кешуються локально для подальшого використання без доступу до мережі. Єдиною апаратною вимогою для роботи з локальними моделями є наявність щонайменше 8 ГБ оперативної пам'яті, що є характеристикою більшості сучасних персональних комп'ютерів.

2.4 Обмеження та припущення

При проєктуванні та реалізації системи прийнято ряд обмежень і припущень, що визначають межі застосовності розробленого методу та умови його коректної роботи. Знання цих обмежень є важливим як для правильного розгортання системи, так і для інтерпретації отриманих результатів.

Обмеження, пов'язані з локальними мовними моделями. Локальні моделі TinyLlama та Qwen є компактними моделями з відносно невеликою кількістю параметрів, що обумовлює їх обмежені генеративні можливості порівняно з повнорозмірними моделями. Зокрема, якість генерації складних аналітичних відповідей для цих моделей може поступатися хмарній моделі OpenAI. Вибір саме цих моделей продиктований їх здатністю працювати на споживчому обладнанні без необхідності у спеціалізованих GPU-прискорювачах. Завантаження локальної моделі з репозиторію виконується одноразово під час першого запуску і потребує стабільного інтернет-з'єднання; при подальших запусках модель використовується з локального кешу.

Обмеження контекстного вікна. Розмір контексту, що передається до мовної моделі-генератора, обмежений параметром у 2200 символів. Це обмеження є прямим наслідком розміру контекстного вікна локальних моделей і не може бути збільшене без заміни моделі. На практиці це означає, що навіть якщо ретрівер знайшов велику кількість релевантних фрагментів, до генератора передаються лише найрелевантніші з них, обрані за результатами ранжування. Таким чином, якість відповіді залежить не лише від повноти ретрівалу, а й від точності ранжування кандидатів.

Обмеження, пов'язані з веб-провайдерами. Доступність і повнота результатів веб ретрівалу залежать від зовнішніх факторів, що не перебувають під контролем системи. Провайдери Bing та Google потребують наявності відповідних API ключів у змінних середовища, за їх

відсутності ці провайдери автоматично пропускаються. Академічні провайдери (OpenAlex, arXiv, Semantic Scholar) орієнтовані на науковий контент і можуть бути менш ефективними для фактологічних або побутових запитів.

Обмеження мовної підтримки. Модель вбудовувань sentence-transformers орієнтована переважно на англомовні тексти. Для україномовних запитів і документів якість векторного представлення може бути дещо нижчою, ніж для англомовних, що безпосередньо впливає на точність векторного пошуку. Лексичні компоненти комбінованого підрахунку частково компенсують цей недолік за рахунок аналізу токенного перекриття між запитом і фрагментами без використання векторних представлень, проте повноцінна підтримка багатомовності потребує використання спеціалізованих мультилінгвальних моделей вбудовувань, що виходить за межі поточної реалізації.

Обмеження графового розширення. Графове розширення контексту виконується в межах параметра і дорівнює 5, що обмежує кількість додаткових вузлів, доданих через граф за один запит. Це обмеження введено навмисно, щоб уникнути надмірного розширення контексту нерелевантними фрагментами. Ребро між двома фрагментами додається до графа лише у тому випадку, якщо вага семантичного зв'язку між ними перевищує мінімальний поріг 0.6, що на практиці означає наявність сильного тематичного перекриття. Для коротких корпусів (менше 5–10 документів) граф може бути розрідженим, і графове розширення матиме незначний ефект.

2.4.1 Припущення щодо якості вхідних даних

Система припускає, що завантажені локальні документи містять осмислений текстовий контент, безпосередньо пов'язаний з тематикою очікуваних запитів. Завантаження документів, що містять переважно

таблиці, формули або зображення (наприклад, скановані PDF без розпізнаного тексту), не призведе до помилки, проте якість ретрівалу буде суттєво нижчою. Система також припускає, що запит формулюється однією мовою – українською або англійською; змішаномовні запити можуть оброблятися некоректно на етапі перекладу та лексичного аналізу.

2.4.2 Припущення щодо середовища виконання

Система розроблена і протестована в середовищі Python 3.12 на платформі Windows. Сумісність з іншими версіями Python (3.10, 3.11) та іншими операційними системами (Linux, macOS) забезпечується використанням крос-платформних бібліотек. Для нормальної роботи системи необхідне активне інтернет-з'єднання при першому запуску (для завантаження моделей) та у веб-режимі (для звернення до провайдерів).

Схема прийняття рішень при обробці запиту наведена у додатку А.

2.5 Огляд та обґрунтування вибору технологічного стека

Вибір технологічного стека є одним із ключових архітектурних рішень, що визначає можливості системи, складність її реалізації та перспективи подальшого розвитку. При виборі компонентів стека керувались такими критеріями: відкритість і доступність (перевага надається інструментам з відкритим кодом або безкоштовним рівнем доступу), зрілість і активна підтримка спільнотою, сумісність між компонентами, а також можливість роботи на споживчому обладнанні без спеціалізованих апаратних прискорювачів.

Основною мовою реалізації обрано Python 3.12. Python є де-факто стандартом у галузі машинного навчання та обробки природної мови: переважна більшість сучасних бібліотек для роботи з мовними моделями, векторного пошуку та побудови графів розроблені саме для Python і мають

найбільш повну та актуальну підтримку саме в цьому середовищі. Альтернативою могла б виступити мова Java, однак вона суттєво поступається Python за обсягом доступних ML бібліотек.

Для реалізації інтерфейсу обрано Streamlit – фреймворк для швидкої побудови інтерактивних веб застосунків на Python. Ключовою перевагою Streamlit є те, що весь інтерфейс описується у звичайному Python коді без необхідності писати HTML, CSS або JavaScript. Це дозволяє дослідницьким і прототипним застосункам досягати рівня зручності використання, порівнянного з повноцінними веб застосунками [17], при значно менших витратах на розробку.

Для інтеграції векторного сховища, моделей вбудовувань та мовних моделей у єдиний пайплайн використано бібліотеку LangChain[18]. Вона надає уніфікований інтерфейс для роботи з різними моделями та сховищами, що дозволяє замінювати компоненти пайплайну (наприклад, векторне сховище або мовну модель) без переписування решти коду. Альтернативою міг виступити LlamaIndex, однак обрана бібліотека має ширшу екосистему інтеграцій та більш детальну документацію для задач, пов'язаних із побудовою RAG систем.

Для перетворення текстових фрагментів і запитів у векторні представлення обрано модель sentence-transformers/all-MiniLM-L6-v2 [19] із бібліотеки sentence-transformers. Ця модель є компромісом між якістю семантичних представлень і обчислювальною ефективністю: маючи лише двадцять два мільйони параметрів, вона демонструє конкурентну якість ретрівалу на стандартних бенчмарках при часі векторизації, що дозволяє працювати без GPU. Серед розглянутих альтернатив – моделі сімейства bge та e5: перші орієнтовані переважно на англомовні тексти, другі мають кращу підтримку мультилінгвальності, проте потребують більших обчислювальних ресурсів.

Для зберігання та пошуку векторних представлень обрано FAISS – бібліотеку для ефективного пошуку найближчих сусідів у щільних

векторних просторах. Вона є одним з найшвидших рішень для векторного пошуку і не потребує окремого серверного процесу, що спрощує розгортання. Альтернативами є Chroma та Qdrant: Chroma зручна для прототипування і має простіший API, але поступається FAISS за продуктивністю на великих корпусах; Qdrant є потужним рішенням для комерційних систем, проте потребує розгортання окремого сервера, що надмірно ускладнює інфраструктуру для дослідницького застосунку.

Для побудови та обходу графа походження обрано бібліотеку NetworkX – стандарт де-факто для роботи з графами у Python. NetworkX надає повний набір алгоритмів обходу (BFS, DFS), обчислення центральності вузлів та аналізу структури графа, що безпосередньо використовуються у реалізованому методі. Для інтерактивної візуалізації графа в інтерфейсі використано бібліотеку PyVis, яка на основі об'єктів NetworkX генерує HTML файл з інтерактивною мережею на базі бібліотеки vis.js.

Для генерації відповідей у локальному режимі використано моделі TinyLlama та Qwen, завантажені через бібліотеку HuggingFace Transformers. Обидві моделі є інструкційно налаштованими, що означає їх здатність дотримуватися структурованих підказок і формувати зв'язні відповіді на запити. Вибір цих моделей зумовлений їх здатністю працювати на CPU без GPU прискорення при розумному часі генерації.

Як хмарна альтернатива локальним моделям підтримується OpenAI через офіційний API. Ця модель забезпечує суттєво вищу якість генерації та підтримує більший розмір контекстного вікна, що дозволяє передавати до генератора більше релевантних фрагментів. Інтеграція реалізована через пакет langchain openai і активується автоматично при наявності змінної середовища. Таким чином, користувач може обирати між локальними моделями з повним контролем над даними та хмарною моделлю з вищою якістю генерації залежно від вимог до конфіденційності й ресурсів.

2.6 Висновки за розділом

У другому розділі сформульовано постановку задачі та визначено повний набір вимог до розробленої системи.

Формалізована постановка задачі визначає систему як функцію перетворення запиту користувача і доступних джерел знань на три взаємопов'язані вихідні об'єкти: природномовну відповідь, скалярну оцінку довіри та граф походження документів. Визначено вхідні параметри системи – запит, локальний корпус, множину веб провайдерів, параметри k і hops – та умови якості результату, що регламентують вимоги до підкріпленості відповіді джерелами і повноти провенанс записів.

Функціональні вимоги охо плюють дванадцять ключових можливостей системи: від завантаження документів і побудови графа походження до режиму порівняльного оцінювання і збереження історії запитів у сесії. Нефункціональні вимоги визначають характеристики продуктивності, надійності, розширюваності, зручності використання та портативності, забезпечуючи практичну придатність системи для реального використання.

Зафіксовано ключові обмеження системи: розмір контекстного вікна локальних моделей, залежність від API ключів для окремих провайдерів, орієнтованість базової моделі вбудовувань переважно на англомовні тексти та обмеження глибини і масштабу графового розширення. Ці обмеження є усвідомленими проєктними рішеннями, що визначають межі застосовності поточної реалізації і вказують на напрями її вдосконалення. Окрему увагу приділено тому, щоб зазначені обмеження не суперечили поставленій меті роботи, а радше задавали реалістичний баланс між якістю результатів, ресурсними витратами та складністю впровадження.

Обґрунтовано вибір технологічного стека, а результати цього розділу є основою для розробки архітектури та алгоритмів методу, що детально описуються у розділі 3.

3 МЕТОД ГЕНЕРАЦІЇ ВІДПОВІДЕЙ НА ОСНОВІ RAG ІЗ ГРАФОМ ПОХОДЖЕННЯ ТА ОЦІНКОЮ ДОВІРИ

У цьому розділі описано архітектуру та алгоритми методу генерації відповідей. Метод поєднує підхід Retrieval-Augmented Generation із графом походження документів і багатофакторною оцінкою довіри, реалізованими у вигляді єдиного програмного пайплайну. Розділ побудовано таким чином, що кожен підрозділ відповідає одному компоненту методу – від індексації документів і векторного пошуку до генерації відповіді та обчислення оцінки довіри. Для кожного компонента наведено опис алгоритму, відповідні математичні вирази та фрагменти програмної реалізації.

3.1 Загальна архітектура методу

Данний метод є розширенням класичної архітектури Retrieval-Augmented Generation, описаної у розділі 1, і вирішує дві додаткові задачі, що не охоплюються базовим RAG: структуровану фіксацію походження кожного фрагмента, використаного для формування відповіді, та кількісну оцінку довіри до цієї відповіді на основі сукупності характеристик ретрівалу, графа та самої відповіді.

Метод складається з шести взаємопов'язаних компонентів, що виконуються послідовно при обробці кожного запиту.

Перший компонент – індексатор – відповідає за попередню обробку локального корпусу документів. Він розбиває завантажені документи на фрагменти фіксованої довжини, векторизує їх за допомогою моделі вбудовувань і зберігає у FAISS-індексі. Одночасно над проіндексованими фрагментами будується граф походження з послідовними та семантичними зв'язками. Ця операція є одноразовою і виконується до початку обробки запитів.

Другий компонент – ретрівер – отримує вхідний запит і знаходить у векторному індексі початковий набір з k найрелевантніших фрагментів. Для кожного seed-фрагмента обчислюється комбінований підрахунок, що враховує векторну подібність, лексичне перекриття з запитом та евідентну відповідність. Результат ранжується, після чого слабкі кандидати відфільтровуються.

Третій компонент – розширювач контексту на основі графа – для кожного seed-фрагмента виконує обхід графа походження в ширину на задану глибину (hops) і додає до контексту сусідні фрагменти, що задовольняють умовам мінімальної ваги ребра та мінімального score відповідності запиту. Для кожного доданого фрагмента фіксується повний шлях у графі – від seed-фрагмента до нього – з зазначенням типу ребра, його ваги та відстані в хопах. Якщо глибина = 0, цей компонент пропускається і метод вироджується у базовий RAG.

Четвертий компонент – генератор відповіді – отримує сформований контекст і запит та формує відповідь одним з двох способів: екстрактивним або генеративним. В екстрактивному режимі система вилучає найрелевантніше речення або структурований перелік безпосередньо з тексту фрагментів. У генеративному режимі контекст і запит передаються до мовної моделі у вигляді структурованого промпту.

П'ятий компонент – оцінювач довіри – обчислює скалярну оцінку довіри $C \in [0, 1]$ як зважену суму восьми нормалізованих факторів, кожен з яких відображає певну характеристику якості ретрівалу, структури графа або отриманої відповіді. Після генерації відповіді оцінка уточнюється шляхом калібрування за якістю самої відповіді та переводиться у якісний рівень.

Шостий компонент – реєстратор провенансу – протягом усього пайплайну формує граф походження і для кожного фрагмента, що потрапив до фінального контексту, зберігає структурований запис із зазначенням

джерела, режиму отримання, score та повного шляху від запиту до фрагмента у графі.

Повний маршрут обробки запиту наведено у додатку Б. Стрілки показують потік даних між компонентами: запит і параметри надходять на вхід ретрівера, seed-фрагменти передаються до розширювача, розширений контекст – до генератора та оцінювача довіри, а реєстратор провенансу отримує дані на всіх проміжних кроках паралельно з основним потоком обробки.

Залежно від обраного режиму пошуку архітектура методу дещо варіюється. У локальному режимі всі шість компонентів задіяні повністю. У веб-режимі індексатор і розширювач контексту замінюються на веб-ретрівер із клієнтом WebSearchClient, що виконує пошук через зовнішні провайдери; граф походження будується динамічно над отриманими веб-результатами на основі приналежності до провайдерів та тематичної схожості заголовків. У комбінованому режимі основний пошук за локальним корпусом та запити до веб-провайдерів виконуються паралельно, після чого користувачеві подаються дві окремі відповіді — на основі локальних джерел і на основі веб-джерел — кожна зі своїм переліком використаних фрагментів та власним значенням довіри. Додатково обчислюється агрегована оцінка довіри над об'єднаним набором фрагментів обох гілок, яка відображається у заголовку запису історії і слугує орієнтиром для загальної надійності відповіді у даному запиті.

3.2 Індексація документів і векторний пошук

Першим етапом обробки даних у запропонованому методі є індексація локального корпусу документів. Цей етап виконується одноразово перед обробкою запитів і складається з трьох послідовних кроків: розбиття документів на фрагменти, обчислення векторних представлень та побудова векторного індексу [20].

Завантажені документи мають змінну довжину – від декількох речень до десятків сторінок – що робить неможливим їх безпосередню векторизацію. Натомість кожен документ розбивається на невеликі фрагменти (chunks або чанки) фіксованої довжини. Розмір 500 символів є компромісом: достатньо великим, щоб фрагмент містив осмислений контекст (зазвичай 2–4 речення), і достатньо малим, щоб уникнути «розмиття» вектора при усередненні семантики надто довгого тексту. Перекриття у 50 символів між сусідніми фрагментами зменшує ймовірність втрати важливої інформації на межі поділу.

Після розбиття кожному фрагменту присвоюється унікальний ідентифікатор у форматі chunk_i, де i – порядковий номер фрагмента в загальному списку. Метадані кожного фрагмента містять посилання на джерело (source – назва вихідного файлу) та ідентифікатор (chunk_id).

Кожен фрагмент перетворюється на щільний вектор фіксованої розмірності за допомогою попередньо навченої моделі вбудовувань. Ця модель генерує вектори розмірності 384, що є компромісом між якістю семантичного представлення та обчислювальною ефективністю. У налаштуваннях моделі активовано опцію нормалізації векторів до одиничної довжини. Така попередня нормалізація дозволяє обчислювати косинусну подібність між запитом і фрагментом як простий скалярний добуток їх векторів, що значно прискорює пошук на великих обсягах даних без втрати точності результату.

Векторизовані фрагменти зберігаються у векторному сховищі FAISS. Сховище будує внутрішній індекс, що дозволяє виконувати пошук Top-k найближчих сусідів за час, значно менший за лінійний обхід усіх векторів. Обрана схема розбиття забезпечує відтворюваність індексації: повторне завантаження того самого корпусу дає однаковий набір фрагментів і векторів. Такий підхід добре масштабується від невеликих тестових корпусів до реальних масивів документів без зміни основних алгоритмів.

Загальний процес індексації описується на рисунку 3.1.

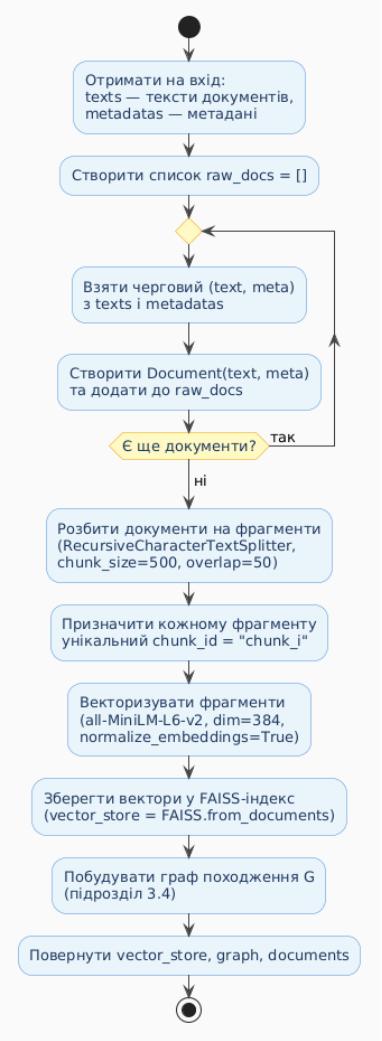


Рисунок 3.1 – Індексація локального корпусу документів

Програмна реалізація цього алгоритму наведена у додатку В.

Після завершення індексації система готова виконувати пошук релевантних фрагментів за вхідним запитом. Векторний пошук є першим кроком обробки кожного запиту і реалізується через виклик методу similarity_search_with_score бібліотеки LangChain. Цей метод інкапсулює в собі три послідовні операції, що виконуються автоматично: обчислення векторного представлення вхідного запиту за допомогою тієї самої моделі вбудовувань, що використовувалась на етапі індексації; пошук у FAISS-індексі найближчих сусідів до отриманого вектора; формування результату у вигляді списку пар «фрагмент – відстань», відсортованого за

зростанням відстані. Використання однієї моделі вбудовувань як для індексації, так і для векторизації запиту є принципово важливим: лише за цієї умови векторні представлення запиту і фрагментів належать одному простору і можуть бути коректно порівняні.

Слід враховувати особливість роботи FAISS: за замовчуванням ця бібліотека повертає не значення подібності, а евклідову (L2) відстань між векторами. Для нормалізованих векторів менше значення відстані відповідає більшій семантичній схожості, проте шкала L2-відстані обернена і не обмежена зверху, що ускладнює її пряме використання у подальших обчисленнях оцінки довіри. З цією метою застосовується монотонно спадне перетворення відстані на величину, обмежену інтервалом (0, 1]:

$$s = \frac{1}{1 + d}, \tag{3.1}$$

де $d \geq 0$ – L2-відстань;

$s \in (0, 1]$.

При нульовій відстані (повний збіг векторів) перетворення повертає максимальне значення $s = 1$; при зростанні відстані подібність плавно спадає до нуля, не досягаючи його. Така властивість зручна як для подальшого ранжування, так і для обчислення факторів довіри, де всі величини мають належати єдиній нормалізованій шкалі.

Кількість кандидатів, що запитується у FAISS на першому кроці, визначається параметром, що дорівню 12 – тобто це розмір пулу для подальшого переранжування. Збільшення пулу понад значення k, заданого користувачем, є усвідомленим архітектурним рішенням. Розширений пул дозволяє наступному компоненту – модулю переранжування, описаному у підрозділі 3.3, – обирати фінальні фрагменти з урахуванням не лише векторної подібності, а й лексичного перекриття із запитом та евідентної відповідності контенту фрагмента типу запиту. Таким чином, попарне

розділення обов'язків між векторним пошуком і переранжуванням [21] підвищує загальну якість ретрівалу.

3.3 Ранжування та відбір кандидатів

Векторний пошук, описаний у попередньому підрозділі, повертає пул із дванадцяти фрагментів кандидатів, упорядкованих за L2 відстанню. Безпосереднє використання цього упорядкування для формування фінального контексту, однак, має певні недоліки. Векторна подібність відображає лише семантичну близькість у просторі вбудовувань і не враховує лексичного перекриття між запитом і текстом фрагмента. Окрім того, для різних типів запитів важливою є різна структурна форма фрагмента: для означальних запитів – наявність дефінітивних маркерів (« – це», «називається»), для перелікових – наявність списків через кому, для фактологічних – наявність числових даних або власних назв. Для усунення цих незначних недоліків у системі реалізовано окремий етап переранжування, у межах якого для кожного кандидата обчислюється комбінований підрахунок [22] на основі трьох взаємодоповнюючих компонентів.

Перед обчисленням комбінованого score система визначає тип або режим запиту на основі його лексичної структури. Розрізняється шість режимів: означальний, фактологічний, переліковий, бінарний, процесний та загальний. Класифікація виконується методом за ключовими маркерами на початку запиту: «що таке», «визначення», «– це» вказують на означальний режим; «скільки», «хто», «коли» – на фактологічний; «які», «які є» – на переліковий; «чи» – на бінарний; «як» – на процесний; решта запитів класифікується як загальні. Визначений режим використовується на всіх подальших етапах обробки – при ранжуванні кандидатів, формуванні відповіді та оцінюванні довіри.

Для кожного кандидата з пулу обчислюються три незалежні компоненти оцінки [23]. Першим є базова семантична подібність, що обчислюється за формулою (3.1) на основі L2-відстані, повернутої FAISS, і відображає семантичну близькість фрагмента до запиту у просторі вбудовувань. Другим компонентом виступає лексичне перекриття – відношення кількості токенів запиту, присутніх у фрагменті, до загальної кількості значущих токенів запиту:

$$\mathrm{lex}(\mathrm{q},\mathrm{d}) = \frac{|\mathrm{T}(\mathrm{q}) \cap \mathrm{T}(\mathrm{d})|}{|\mathrm{T}(\mathrm{q})|}, \tag{3.2}$$

де lex ∈ [0, 1];

T(q) – множина значущих токенів запиту (після виключення стоп-слів і коротких токенів довжиною не більше двох символів);

T(d) – множина токенів фрагмента.

Цей компонент компенсує обмеження векторної моделі і безпосередньо вимірює наявність ключових слів запиту у тексті фрагмента. Третім компонентом є евідентний підрахунок, що становить максимальну оцінку відповідності серед усіх речень фрагмента. Для кожного речення обчислюється комбінація лексичного перекриття із запитом, наявності спеціальних маркерів, характерних для поточного режиму запиту, наприклад, для означального режиму – наявність патерну «– це», та бонусу за повний збіг ключової фрази запиту у реченні; серед усіх речень фрагмента обирається максимальне значення:

$$ev(q,d) = \max over\ s \in\ sentences(d) * \\ * \left[\, 0.65 \cdot lex(q,s) + \ bonus_{mode(q,s)} + \ phrase_{bonus(q,s)}\right], \tag{3.3}$$

де q – вхідний запит користувача (рядок природною мовою);

d – фрагмент-кандидат, отриманий з векторного індексу;

s – окреме речення з sentences(d), по якому виконується ітерація;

sentences(d) – множина речень, на які розбито текст фрагмента d. Розбиття виконується за пунктуаційними маркерами (. ! ?);

lex(q, s) – лексичне перекриття між запитом і реченням, обчислене за принципом, що й у формулі (3.2), але на рівні одного речення, а не всього фрагмента; величина безрозмірна, набуває значень з інтервалу [0, 1];

bonus_mode(q, s) – бонус, що залежить від режиму запиту (definition, factoid, list, yes_no, process, general) та наявності у реченні маркерів, характерних для цього режиму. Наприклад, для означального режиму нараховується бонус 0.30 за наявність патернів типу «– це», «називається», «являє собою»; для перелікового – за наявність переліку через кому або маркерів «типи», «види». Значення бонусу безрозмірне і обмежене зверху значенням 0.30;

phrase_bonus(q, s) – бонус за дослівний збіг повної ключової фрази запиту (з двох і більше токенів) у реченні s. Набуває одного з двох значень: 0 (немає збігу) або 0.20 (є збіг). Величина безрозмірна;

коефіцієнт 0.65 – вага лексичного перекриття у формуванні оцінки на рівні речення; обрана емпірично і відображає те, що базою евідентного score є саме лексична близькість, а бонуси лише підсилюють її.

Підсумкова величина ev(q, d) є безрозмірною і нормалізована до інтервалу [0, 1]. Семантичний зміст: чим вищий ev, тим більш «прицільно» хоча б одне речення фрагмента відповідає запиту з урахуванням як лексики, так і структурних особливостей очікуваної відповіді.

Евідентний підрахунок є найбільш «прицільним» з трьох компонентів, оскільки враховує не лише наявність ключових слів, а й їх розташування в межах одного речення та контекст вживання.

Підсумкова оцінка кандидата формується як зважена сума трьох описаних компонентів з додаванням бонусу за повний збіг ключової фрази запиту в тексті фрагмента:

$$\text{score}(q, d) = 0.35 \cdot \text{sim}(q, d) + 0.25 \cdot \text{lex}(q, d) + \\ + 0.25 \cdot \text{ev}(q, d) + \varphi(q, d), \quad (3.4)$$

де q – вхідний запит;

d – фрагмент-кандидат;

sim(q, d) – базова семантична подібність, обчислена за формулою (3.1) на основі L2-відстані між векторними представленнями запиту і фрагмента; величина безрозмірна, з інтервалу (0, 1];

lex(q, d) – лексичне перекриття між запитом і фрагментом згідно з формулою (3.2); безрозмірне, з інтервалу [0, 1];

ev(q, d) – евідентний score за формулою (3.3); з інтервалу [0, 1], безрозмірний;

φ(q, d) – бонус за збіг повної ключової фрази запиту (з двох і більше токенів) безпосередньо у тексті фрагмента. Набуває одного з двох значень: нуль (немає збігу) або 0.20 (є збіг); величина безрозмірна;

score(q, d), підсумкова величина, обрізається зверху значенням 1.0, отже фінальний інтервал – [0, 1].

Для запитів означального типу, які особливо чутливі до точного збігу терміна, передбачено додаткове коригування: якщо лексичне перекриття для кандидата є меншим за 0.5, його рахунок множиться на 0.55, що фактично виключає такі кандидати з топ-позицій і запобігає ситуаціям, коли семантично близький, але лексично некоректний фрагмент випереджає прямий збіг терміна.

Після обчислення комбінованого підрахунку для всіх кандидатів пулу вони впорядковуються за спаданням значення, і повертаються лідируючи результати. Для означальних запитів передбачено окремий шлях обробки: спочатку система намагається знайти кандидатів безпосередньо за лексичним і структурним критеріями означень, і лише за відсутності таких кандидатів повертається до загального алгоритму ранжування. Загальна послідовність дій етапу ранжування зображена на рисунку 3.2.

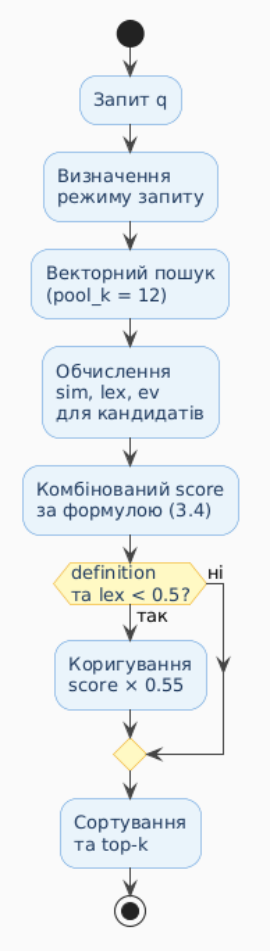


Рисунок 3.2 – Блок-схема алгоритму ранжування та відбору кандидатів

Програмна реалізація ключової формули (3.4) наведена у лістингу 3.1.

Лістинг 3.1 – Обчислення комбінованого score кандидата

```
    def _score_doc_against_query(self, query, doc, raw_score):
        base_similarity                                          =
self._distance_to_similarity(raw_score)
        lexical_score   =   self._lexical_overlap_score(query,
doc)
        phrase_bonus = self._phrase_match_bonus(query, doc)
        evidence_score = self._doc_evidence_score(query, doc)
        final_score = (
            0.35 * base_similarity
            + 0.25 * lexical_score
            + 0.25 * evidence_score
            + phrase_bonus
        )
```

Продовження лістингу 3.1

```
    if self._is_definition_query(query) and lexical_score < 
0.5:
            final_score *= 0.55
        return min(final_score, 1.0)
```

Узагальнення режимів запиту та відповідних маркерів класифікації наведено у таблиці 3.1. Представлена схема має евристичний характер: класифікація виконується на основі простих лексичних шаблонів і не претендує на повноту покриття всіх можливих формулювань. У разі неоднозначних запитів система переходить до загального режиму general, що гарантує коректну роботу методу навіть за неточної класифікації.

Таблиця 3.1 – Режими запиту та маркери класифікації

Режим	Лексичні маркери	Структурні особливості очікуваних фрагментів
definition	«що таке», «визначення», «це»	Дефінітивні патерни («– це», «називається»)
factoid	«скільки», «хто», «коли»	Числа, дати, власні назви
list	«які», «які є»	Перелік через кому, маркери «типи», «види», «класиф»
yes_no	«чи»	Стверджувальні/заперечні конструкції («може», «не може»)
process	«як»	Послідовність дій («спочатку», «далі», «потім»)
general	(всі інші)	Без специфічних обмежень

Сформований таким чином список основних фрагментів-кандидатів передається наступному компоненту методу – модулю розширення контексту на основі графа походження, описаному у підрозділі 3.4.

3.4 Побудова графа походження та графове розширення контексту

Граф походження є центральним компонентом запропонованого методу і відрізняє його від класичної архітектури RAG. У запропонованому підході граф виконує дві взаємодоповнюючі функції: він слугує структурованим записом про джерела й зв'язки між фрагментами знань, а також виступає інструментом розширення контексту шляхом залучення фрагментів, що пов'язані з seed-результатами векторного пошуку семантично або послідовно. Нижче послідовно описано структуру графа, алгоритм його побудови, процедуру обходу для розширення контексту та механізм фіксації провенанс-записів, що супроводжує всі ці операції.

Граф будується над множиною всіх фрагментів локального корпусу і реалізується як неорієнтований мультиграф на основі NetworkX [24]. Множина вузлів містить по одному вузлу на кожен фрагмент, причому кожен вузол зберігає такі атрибути як текст фрагмента та назву вихідного документа. Множина ребер поділяється на два типи, що відображають принципово різні види зв'язків між фрагментами. Ребра першого типу – послідовні – з'єднують фрагменти, що йдуть один за одним у межах одного вихідного документа, і відображають структурну неперервність тексту: сусідні фрагменти зазвичай разом утворюють єдиний логічний блок (абзац, розділ, опис явища) [25]. Послідовним ребрам присвоюється фіксована вага, оскільки структурний зв'язок є однозначним і не потребує оцінки сили. Ребра другого типу – семантичні – з'єднують фрагменти, що належать різним документам або не є послідовними у межах одного документа, але мають високу семантичну близькість у просторі вбудовувань [26]. Семантичні ребра дозволяють виявляти тематичні зв'язки між незалежно завантаженими документами, що особливо важливо при роботі з гетерогенним корпусом. Вага семантичного ребра обчислюється на основі тієї самої L2-відстані, що і у формулі (3.1):

$$w_{sem(d^1,d^2)} = \frac{1}{1 + L2(d^1, d^2)}, \quad (3.5)$$

де $L2(d_1, d_2)$ – L2-відстань між векторами фрагментів.

Ребро додається до графа лише за умови $w_{sem} > 0.45$, що відповідає достатньо сильному тематичному перекриттю; слабші зв'язки ігноруються, щоб уникнути «шумових» ребер, які не несуть змістового навантаження.

Побудова графа виконується одноразово після індексації локального корпусу. Для кожного фрагмента у списку створюється відповідний вузол із заповненими атрибутами. Далі виконується ітерація парами по сусідніх фрагментах: якщо два сусідні фрагменти належать одному джерелу, між їх вузлами додається ребро послідовного типу з вагою 1.0. На завершальному кроці для кожного фрагмента викликається пошук його чотирьох найближчих сусідів у векторному індексі, для кожного знайденого сусіда обчислюється вага семантичного ребра за формулою (3.5), і ребро додається до графа за умови, що вага перевищує поріг 0.45 та сусід не є тим самим фрагментом. Результатом виконання алгоритму є певний граф, у якому послідовні ребра утворюють «ланцюги» в межах кожного документа, а семантичні ребра з'єднують ці ланцюги між собою у точках тематичної подібності.

Після того, як ретрівер відібрав seed-фрагменти за допомогою комбінованого підрахунку (підрозділ 3.3), для кожного seed-фрагмента виконується обхід графа на задану користувачем глибину. Метою обходу є виявлення сусідніх фрагментів, що з високою імовірністю містять додаткову релевантну інформацію, але не були обрані векторним пошуком напряму через обмеження моделі вбудовувань або через те, що ключові слова запиту присутні у сусідніх фрагментах, а не у самому seed-фрагменті. Обхід реалізовано як модифікований пошук у ширину (BFS) [27]. На відміну від стандартного BFS, у моїй реалізації передбачено два важливі обмеження. По-перше, на кожному кроці серед усіх

непосіщених сусідів поточного вузла обираються лише два найкращі, упорядковані за пріоритетом ребра (послідовні мають пріоритет 2.0, семантичні – 1.0) та вагою. По-друге, загальна кількість вузлів, доданих до контексту через граф, обмежена параметром і дорівнює 5, що запобігає надмірному розростанню контексту нерелевантними фрагментами навіть у щільно зв'язаному графі.

Кожен кандидат на додавання, отриманий від обходу, проходить додаткову фільтрацію. Спочатку перевіряється, що вага ребра перевищує поріг, що дорівнює 0.6 – ще жорсткіший критерій, ніж той, що застосовується при побудові графа. Потім обчислюється комбінований рахунок кандидата відносно запиту за формулою (3.4) і коригується з урахуванням відстані та ваги ребра:

$$\mathrm{graph}_{\mathrm{score}(q,d_n)} = \mathrm{score}(q, d_n) \cdot 0.96^{\mathrm{dist}} \cdot \max(w_{\mathrm{edge}}, 0.5), \qquad (3.6)$$

де d_n – кандидат, виявлений через граф;

dist – відстань у хопах від seed-фрагмента до кандидата;

w_{edge} – вага ребра, через яке кандидат був досягнутий;

0.96^{dist} забезпечує плавне згасання значущості зі збільшенням дистанції від seed-фрагмента, а множник $\max(w_{\mathrm{edge}}, 0.5)$ пов'язує підсумкову оцінку з силою графового зв'язку. Для кожного seed-фрагмента до фінального контексту додається не більше двох найкращих упорядкованих графових сусідів.

Паралельно з формуванням контексту система веде структурований запис походження кожного фрагмента у словнику. Для фрагментів, отриманих безпосередньо векторним пошуком, створюється запис з полями retrieval_mode = "vector_seed", retrieval_score, reason та шляхом path, що містить один крок – від абстрактного вузла «query» безпосередньо до фрагмента. Для фрагментів, отриманих графовим розширенням, формується запис з retrieval_mode = "graph_expansion", що додатково містить

ідентифікатор seed-фрагмента, тип ребра, вагу ребра та відстань, а поле path містить повний ланцюжок: спочатку шлях від запиту до seed-фрагмента, успадкованого від його запису, а потім крок «seed → кандидат» з зазначенням типу графового переходу. Така структура провенанс записів дозволяє у подальшому пояснити користувачеві, чому саме той чи інший фрагмент потрапив до контексту, і відобразити повний шлях його отримання.

3.5 Інтеграція веб-ретрівалу

Окрім роботи з локальним корпусом документів, метод підтримує отримання релевантного контексту з зовнішніх веб джерел. Інтеграція веб-ретрівалу є необхідною з кількох причин: локальний корпус неминуче обмежений тематикою завантажених документів, тому для запитів за його межами система повинна мати можливість звернутися до зовнішніх джерел знань; і для запитів про події або довідкову інформацію (визначення, факти, дати) енциклопедичні веб-ресурси часто є надійнішим джерелом, ніж випадково завантажений локальний корпус. Веб ретрівер реалізований у вигляді окремого модуля і використовується класом WebSearchClient, що взаємодіє з основним RAG пайплайном через той самий інтерфейс, що і локальний ретрівер.

3.5.1 Архітектура WebSearchClient

Клас WebSearchClient побудований за принципом єдиного фасаду над набором незалежних методів, кожен з яких відповідає за звернення до одного конкретного веб провайдера. Така архітектура спрощує супровід коду і додавання нових джерел: реалізація методу для нового провайдера зводиться до написання функції, що звертається до відповідного API і повертає список об'єктів WebSearchResult – стандартизованої структури

даних з полями (провайдер, заголовок, сніпет, посилання) та попередньою оцінкою релевантності. Усі звернення до зовнішніх API виконуються через спільну HTTP сесію з обмеженням часу очікування у 12 секунд і налаштованими заголовками. Кожен метод звернення огорнутий блоком обробки виключень. Це проєктне рішення забезпечує часткову деградацію функціональності: відмова окремого провайдера, мережевий тайм аут або відсутність API ключа не перериває роботу системи, і запит продовжує оброблятися з результатами тих провайдерів, що залишились доступними.

Зовнішнім інтерфейсом класу виступає метод , який отримує на вхід запит, список активних провайдерів і ліміт результатів на провайдера, послідовно викликає відповідні методи через диспетчер та об'єднує результати у єдиний список з дедублікацією за трійкою (провайдер, заголовок, посилання). Сформований список передається у клас, де кожен веб-результат перетворюється на об'єкт типу «Документ»: поле контенту формується як конкатенація заголовка і сніпета, а метадані містять назву провайдера, посилання та комбінований ідентифікатор джерела. У такому вигляді веб-результати потрапляють до того самого пайплайну формування, що і локальні фрагменти, з режимом провенансу.

3.5.2 Підтримувані провайдери та стратегія вибору запиту

Веб-ретрівер підтримує шість провайдерів, обраних з міркувань різноманіття типів інформації: Wikipedia як основне джерело енциклопедичних довідок українською мовою; OpenAlex, arXiv та Semantic Scholar як академічні джерела, що надають доступ до наукових публікацій з анотаціями; Bing і Google як комерційні пошукові системи з широким покриттям веб-простору. Доступ до Wikipedia, OpenAlex, arXiv та Semantic Scholar здійснюється через їхні відкриті API без необхідності реєстрації, що робить ці провайдери активними «з коробки». Bing та Google потребують API-ключів, що задаються через змінні середовища; за відсутності ключа

відповідний провайдер автоматично пропускається із виведенням інформаційного повідомлення у консоль.

Важливою особливістю веб-ретрівера є диференційована стратегія формування запиту залежно від провайдера. Спочатку оригінальний запит користувача проходить через метод, який підставляє розгорнуті форми для розповсюджених абревіатур: «ші» → «штучний інтелект», «ai» → «artificial intelligence / штучний інтелект», «llm» → «large language model / велика мовна модель». Така попередня нормалізація суттєво підвищує якість результатів, оскільки модельні скорочення часто відсутні у заголовках енциклопедичних статей. Далі, для україномовних запитів, виконується переклад на лаконічну англійську фразу. Переклад використовується для звернень до академічних провайдерів та комерційних пошукових систем, оскільки їхні бази даних переважно містять англомовний контент, тоді як для Wikipedia передається розширений український запит.

3.5.3 Ранжування та фільтрація веб-результатів

Об'єднаний список результатів від усіх провайдерів проходить переранжування, після якого для кожного результату обчислюється підсумкова оцінка релевантності за формулою:

$$\begin{aligned} \mathrm{score}_{\mathrm{web(q,r)}} = \ & \mathrm{score}_{\mathrm{base(r)}} + \ 0.55 \cdot \mathrm{lex}(\mathrm{q}, \mathrm{r}) + \\ & + \mathrm{bias}(\mathrm{provider}, \mathrm{mode}) + \ \Delta_{\mathrm{specific(q,r)}}, \end{aligned} \tag{3.7}$$

де $\mathrm{score}_{\mathrm{base(r)}}$ – початкова оцінка, повернута самим провайдером і обчислена на основі позиції результату в його видачі;

lex(q, r) – лексичне перекриття між токенами запиту і об'єднаним текстом «заголовок + сніпет», аналогічне формулі (3.2);

bias(provider, mode) – коригувальний коефіцієнт, що залежить від провайдера і режиму запиту;

$\Delta_{specific(q,r)}$ – сукупність дрібніших коригувань за наявності у заголовку або сніпеті специфічних маркерів: повний збіг заголовка з ключовою фразою запиту, наявність дефінітивних патернів «– це», «називається» у сніпеті для означальних запитів, наявність числових даних або фактологічних маркерів для фактологічних запитів тощо. Коефіцієнт зміщення є ключовим елементом стратегії веб-ранжування і відображає той факт, що різні провайдери мають різну прикладність до запитів різних типів: для означальних і фактологічних запитів Wikipedia природно надають кращі результати, ніж академічні бази, тоді як для запитів про сучасні дослідження пріоритет має зворотний характер.

Після обчислення підсумкової оцінки результати проходять фільтрацію за порогом, що залежить від режиму запиту: для означальних запитів – 0.45, для фактологічних – 0.40, для перелікових – 0.35. Це запобігає потраплянню в контекст слабких результатів навіть у випадках, коли провайдер повернув їх з високою власною оцінкою. Для означальних і фактологічних запитів додатково застосовується стратегія пріоритизації: серед відфільтрованих результатів спочатку обираються до трьох енциклопедичних (Wikipedia), а потім до трьох академічних, що дозволяє побудувати збалансований контекст одночасно зі стислою довідкою і поглибленим науковим матеріалом. У комбінованому режимі веб-результати об'єднуються з локальними, де списки контекстних документів дедублікуються, переранжовуються за об'єднаною оцінкою, а підсумкова довіра обчислюється як зважена комбінація оцінок локального та веб-ретрівалу.

Таким чином, веб-ретрівер є самодостатнім модулем, що розширює можливості системи без зміни основного RAG-пайплайну і забезпечує гнучку поведінку залежно від доступності зовнішніх ресурсів, мови запиту та типу очікуваної відповіді.Усі етапи веб-ранжування та фільтрації побудовані як евристичні надбудови над базовим RAG-пайплайном: веб-результати потрапляють до контексту лише за умови достатньої

змістової близькості до запиту. У випадку відсутності надійних веб-фрагментів система коректно деградує до чисто локального режиму без штучного підвищення оцінки довіри.

3.6 Генерація відповіді на основі контексту

Сформований у попередніх етапах контекст – список релевантних фрагментів локального корпусу, веб-результатів або їх комбінації разом з відповідними записами – передається у компонент генерації відповіді. Цей компонент відповідає за формування природномовної відповіді на запит користувача із суворим обмеженням використовувати лише ту інформацію, що міститься у наданому контексті. Реалізація компонента поєднує два принципово різні шляхи формування відповіді – екстрактивний і генеративний, між якими система обирає динамічно залежно від типу запиту, наявності придатних фрагментів і конфігурації мовної моделі.

Екстрактивний шлях полягає у безпосередньому вилученні готового речення або структурованого переліку з тексту фрагментів контексту, без звернення до мовної моделі. Цей шлях є основним для запитів типу визначення, списку, факту та булевого, для яких очікувана відповідь є коротким, чітко обмеженим текстовим блоком, який зазвичай уже присутній у джерелах у готовому вигляді. Реалізація екстрактивного шляху складається з трьох послідовних спроб обробки. Спочатку для означальних запитів, що оброблюються у веб-режимі або комбінованому режимі, виконується пошук фрагмента, заголовок якого максимально перетинається з фокусом запиту (фрагмент запиту, очищений від ввідних слів «що таке», «визначення»). Якщо такий фрагмент походить від провайдера Wikipedia і його заголовок збігається з фокусом запиту хоча б на один токен, перше осмислене речення цього фрагмента (довжиною не менше 20 символів) повертається як відповідь. Далі, для перелікових запитів, виконується пошук фрагментів із структурою списку (перелік через кому, наявність

маркерів «типи», «види», «класифікації»); серед знайдених кандидатів обирається той, що має найвищу евідентну оцінку, за умови що ця оцінка перевищує зазначений поріг. Нарешті, у загальному випадку виконується ітерація по всіх реченнях усіх фрагментів контексту: для кожного речення обчислюється евідентна оцінка з урахуванням бонусу за збіг заголовка джерела з фокусом запиту, після чого обирається речення з найвищою оцінкою. Для означальних запитів додатково перевіряється наявність дефінітивних патернів («– це», « це », «що таке»); для перелікових – структурна форма списку. Якщо найкраще знайдене речення задовольняє пороговим вимогам режиму запиту, воно повертається як остаточна відповідь; в іншому випадку повертається стандартизоване повідомлення про відсутність прямої відповіді у контексті.

Генеративний шлях використовується тоді, коли екстрактивний шлях не дав придатного результату або коли тип запиту передбачає змістовну переформульовану відповідь, що не може бути механічно вилучена з фрагментів. У межах цього шляху контекст і запит передаються у мовну модель у вигляді структурованого запиту. Для локальних моделей використовується суворий запит у форматі ChatML з системною інструкцією: відповідати виключно на основі наданого контексту, не використовувати зовнішніх знань, а у разі відсутності інформації повертає стандартизоване повідомлення. Для хмарної моделі OpenAI використовується більш гнучкий промпт: модель може поєднувати контекст із власними знаннями, надаючи велику перевагу контексту у разі суперечності, але має чесно повідомляти про відсутність впевненої відповіді. Така диференціація обумовлена різними характеристиками моделей: суворе обмеження для локальних моделей запобігає галюцинаціям, тоді як для якіснішої хмарної моделі дозволено обмежено використовувати її знання у тих випадках, коли контекст є неповним.

Перед передачею до моделі контекст обрізається до максимальної довжини 2200 символів. Це обмеження є наслідком розміру контекстного

вікна локальних моделей. Сам виклик моделі реалізований через ланцюжок LangChain PromptTemplate | LLM | StrOutputParser, що уніфікує інтерфейс для локального та хмарного бекендів. Для локальних моделей додатково передбачено постобробку відповіді: якщо у згенерованій відповіді кількість латинських літер перевищує кількість кириличних, відповідь замінюється на стандартизоване повідомлення про відсутність інформації. Цей фільтр захищає від поведінки малих моделей типу TinyLlama, що схильні переходити на англійську мову у складних запитах.

При координації обох шляхів спочатку обчислюється екстрактивна відповідь. Якщо вона не є fallback-повідомленням і бекенд є локальним – ця відповідь повертається безпосередньо без звернення до LLM, що економить обчислювальні ресурси і гарантує точну прив'язку до тексту джерел. Якщо ж бекенд є хмарним або екстрактивна відповідь є fallback-повідомленням, виконання продовжується до генеративного шляху: спочатку перевіряється оцінка відповідей (для локального бекенду – з порогом 0.3), і лише за умови достатньої підкріпленості контексту відбувається виклик мовної моделі. Така схема забезпечує триступеневий фільтр від галюцинацій: підкріпленість, екстрактивна верифікація і явна інструкція моделі.

Після отримання відповіді – незалежно від того, екстрактивним чи генеративним шляхом вона сформована – виконується останній крок: прив'язка відповіді до конкретних фрагментів контексту, що її підтверджують. Для відповіді обирається до трьох найкраще підкріплюючих фрагментів за допомогою методу, який оцінює кожен фрагмент за евідентною мірою та лексичним перекриттям з відповіддю; до тексту відповіді дописується блок «Джерела:» з посиланнями. Ідентифікатори підкріплюючих фрагментів зберігаються і у подальшому використовуються інтерфейсом для виділення відповідних вузлів графа походження зеленим кольором. Якщо відповідь є fallback-повідомленням

про відсутність інформації, ці поля залишаються порожніми, оскільки прив'язка джерел у такому випадку не має змісту.

Узагальнено, компонент генерації відповіді реалізує гібридну стратегію, що поєднує точність екстрактивного шляху з гнучкістю генеративного та автоматично обирає між ними на основі типу запиту і характеристик контексту. Така архітектура дозволяє системі давати короткі точні відповіді на прості запити безпосередньо з джерел, не вдаючись до галюцинацій локальних моделей, і одночасно зберігає здатність формувати змістовні розгорнуті відповіді на складні запити за участю мовної моделі.

3.7 Багатофакторна оцінка довіри

Завершальним компонентом методу є оцінювач довіри, що формує скалярну величину $C \in [0, 1]$, яка відображає ступінь обґрунтованості згенерованої відповіді знайденим контекстом. Наявність такої оцінки принципово відрізняє метод від класичного RAG: користувач отримує не лише відповідь, а й кількісну характеристику її надійності, що дозволяє приймати усвідомлені рішення про подальше використання отриманої інформації. Оцінка обчислюється у два проходи. На першому проході обчислюється попередня оцінка на основі восьми факторів, що характеризують якість ретрівалу та структуру контексту. На другому проході попередня оцінка калібрується з урахуванням якості вже сформованої відповіді. Підсумкова оцінка переводиться у якісний рівень – низька, обережна, помірна або висока – і повертається користувачеві разом з повним розкладом за факторами для прозорості обчислень.

3.7.1 Перелік факторів довіри та їх ваги

Багатофакторна оцінка довіри побудована на восьми незалежних факторах, кожен з яких відображає окрему характеристику якості

результату. Кожен фактор нормалізований до інтервалу [0, 1], а ваги факторів визначені таким чином, щоб у сумі давати одиницю, а їх відносні значення відповідали значущості фактора у визначенні надійності відповіді. Перелік факторів зведено у таблиці 3.2.

Таблиця 3.2 – Фактори оцінки довіри та їх ваги

Назва фактору	Опис	Вага
retrieval_similarity	Середня релевантність retrieval	0.20
score_stability	Стабільність рахунку між фрагментами	0.10
context_coverage	Покриття контексту	0.04
graph_precision	Обережність графового розширення	0.06
graph_centrality	Центральність вузлів графа	0.02
query_alignment	Відповідність контексту запиту	0.18
top_match	Найсильніший доказ	0.16
answer_evidence	Підкріплення відповіді доказами	0.24

Найвищу вагу 0.24 має фактор answer_evidence, оскільки він безпосередньо вимірює, чи містить контекст конкретні підтвердження для відповіді. Друге за значущістю місце займає retrieval_similarity (0.20), що відображає базову якість векторного пошуку. Майже рівнозначно з ним оцінюється query_alignment (0.18) – інтегральна міра відповідності фінального контексту запиту, та top_match (0.16) – наявність хоча б одного дуже сильного фрагмента, що сам по собі здатен підкріпити відповідь. Решта факторів мають менші ваги. Низька вага граф-центричних факторів є усвідомленим рішенням: граф є важливим інструментом розширення контексту, проте сам по собі не повинен підвищувати довіру, якщо отримані через нього фрагменти не підтверджують відповідь лексично і евідентно. Слід наголосити, що запропонована оцінка довіри має евристичний характер. Модель довіри побудована як зважена сума нормалізованих показників, відібраних на основі аналізу літератури та експертних

міркувань. Ваги факторів та порогові значення (мінімально допустимі значення evidence-показників, межі якісних інтервалів тощо) підбиралися експериментально на валідаційній підмножині запитів і відображають інженерний компроміс між чутливістю та стійкістю оцінки.

3.7.2 Обчислення значень факторів

Усі вісім факторів обчислюються після завершення ретрівалу і формування фінального контексту. Для кожного фактора використано просту й інтерпретовану формулу, що приводить значення до інтервалу від нуля до одиниці.

Перший фактор – середня релевантність ретрівалу – обчислюється як середнє арифметичне векторних оцінок усіх фрагментів, що потрапили до контексту:

$$f^1 = \frac{(s^1 + s^2 + \ldots + s_N)}{N}, \tag{3.8}$$

де s^i – оцінка подібності і-го фрагмента, отримана за формулою (3.1);

N – кількість фрагментів у контексті. Чим вища середня подібність, тим краще ретрівер зіставив запит із джерелами в цілому.

Другий фактор – стабільність оцінок між фрагментами – обчислюється через стандартне відхилення цих самих оцінок:

$$f^2 = 1 - \sigma(s^1, s^2, \ldots, s_N), \tag{3.9}$$

де σ – стандартне відхилення, обмежене зверху одиницею. Сенс показника простий: коли всі знайдені фрагменти мають близькі оцінки, контекст є рівномірно сильним, тому стандартне відхилення мале, а значення фактора, навпаки, велике. Якщо ж серед фрагментів є один сильний і кілька слабких,

відхилення зростає, і фактор знижується. Для одного фрагмента відхилення дорівнює нулю, тож стабільність вважається максимальною.

Третій фактор – покриття контексту – оцінює, чи достатньо фрагментів використано для побудови відповіді:

$$f^3 = \min\left(\frac{N}{6{,}1}\right), \tag{3.10}$$

де N – кількість фрагментів у контексті.

Шість фрагментів обрано як емпіричний поріг насичення: при меншій кількості контекст вважається неповним, а при більшій додаткові фрагменти вже не приносять помітної користі.

Четвертий фактор – обережність графового розширення – штрафує ситуацію, коли більшість контексту складається не з прямих результатів векторного пошуку, а з фрагментів, доданих через граф:

$$f^4 = 1 - \min\left(\frac{N_{graph}}{\max(3, N), 1}\right), \tag{3.11}$$

де N_{graph} – кількість фрагментів, доданих графовим розширенням. Якщо граф додав один-два фрагменти до повноцінної базової вибірки, фактор залишається близьким до одиниці; якщо ж база ретрівалу слабка і контекст «роздутий» графом, значення помітно знижується.

П'ятий фактор – центральність активних вузлів графа – обчислюється як подвоєне середнє значення центральності за ступенем:

$$f^5 = \min(2 \cdot \text{cavg}, 1), \tag{3.12}$$

де cavg – середня центральність за ступенем для активних вузлів (тобто частка сусідів вузла відносно загальної кількості вузлів графа). Висока центральність означає, що активні фрагменти розташовані в щільно

зв'язаній частині графа, а отже з високою імовірністю належать до однієї тематичної області.

Шостий і сьомий фактори – узгодженість контексту із запитом і найсильніший доказ – обчислюються попередньо і відображають, відповідно, середній рівень відповідності всіх фрагментів запиту та оцінку найкращого з них. Кожен з цих показників враховує одночасно лексичне перекриття із запитом, евідентну оцінку (формула 3.3) та векторну подібність.

Восьмий фактор – підкріплення відповіді доказами – обчислюється як максимум евідентних оцінок речень контексту відносно ключової фрази запиту і відображає, чи містить контекст безпосередні підтвердження для відповіді.

Підсумкова попередня оцінка довіри формується як зважена сума всіх восьми факторів:

$$C^0 = w^1 \cdot f^1 + w^2 \cdot f^2 + \ldots + w^8 \cdot f^8, \qquad (3.13)$$

де ваги w взято з таблиці 3.3.

Отриманий результат обмежується інтервалом від нуля до одиниці. Лінійна форма агрегування факторів обрана свідомо – вона забезпечує інтерпретованість внеску кожного компонента, але водночас обмежує виразну здатність моделі в порівнянні з нелінійними чи навчуваними підходами.

3.7.3 Уточнення оцінки за якістю відповіді

Висока попередня оцінка C_0 сама по собі не гарантує, що отримана відповідь є якісною. Контекст може бути сильним і релевантним, проте мовна модель з якихось причин може сформувати відповідь, що погано пов'язана з цим контекстом, повернути надто загальне формулювання або

зовсім пропустити ключове поняття запиту. Щоб оцінка довіри відображала якість саме відповіді, а не лише ретрівалу, після генерації виконується другий прохід – уточнення оцінки.

На цьому кроці обчислюється окрема числова характеристика – якість відповіді $Q \in [0, 1]$. Вона враховує чотири аспекти: лексичне перекриття між запитом і відповіддю (наскільки слова запиту присутні у відповіді), евідентну прив'язку відповіді до контексту (чи зустрічається відповідь або її ключові частини у фрагментах джерел), повноту покриття ключових слів запиту (якщо у відповіді відсутні важливі слова з запиту, якість штрафується) та структурну відповідність типу запиту. Останнє означає, що, наприклад, для запиту-означення відповідь повинна містити дефінітивні маркери типу «– це», для переліkового – мати форму списку, а для фактологічного – містити числа або власні назви.

Підсумкова довіра обчислюється шляхом обмеження попередньої оцінки зверху залежно від якості відповіді. Зміст цього обмеження простий: навіть якщо ретрівер показав сильні факторні значення, але якість відповіді низька, підсумкова довіра не може перевищити лінійно зростаючий поріг від Q. Окрім цього, для зовсім слабких відповідей застосовуються додаткові жорсткі обмеження зверху: при $Q < 0.2$ довіра не перевищує 0.18, при $Q < 0.4$ – не перевищує 0.35, при $Q < 0.55$ – не перевищує 0.52. Така схема обмежень є асиметричною: вона лише знижує оцінку, але ніколи її не підвищує. Завдяки цьому слабка відповідь не може отримати високу оцінку довіри, навіть якщо попередня оцінка C_0 була високою.

Окремий випадок становить веб-режим з хмарною мовною моделлю. У цьому сценарії запит часто формулюється українською, тоді як основні джерела (наприклад, академічні статті) є англомовними. Через переклад прямий лексичний збіг між реченнями відповіді та джерел знижується природним чином, навіть якщо відповідь є цілком коректною. Тому для такої комбінації жорсткі обмеження пом'якшуються і система не штрафує відповідь за низьку лексичну схожість, спричинену саме мовним

розривом. Інформація про застосоване уточнення додається до списку факторів довіри як окремий запис, що містить значення Q та різницю $C\text{-}C_0$ – фактичний внесок уточнення у підсумкову оцінку. Завдяки цьому користувач у розгортаному блоці результату бачить не тільки кінцеве число, а й те, як воно сформувалося.

3.7.4 Якісні рівні довіри та інтерпретація

Хоча скалярне значення C є інформативним і містить усю необхідну інформацію про надійність відповіді, не кожен користувач звик працювати з числовими оцінками подібного типу. Сприйняття дробового числа потребує певної звички та вміння співвідносити отримане значення з очікуваною точністю системи. Для більшості ж сценаріїв практичного застосування користувачеві важливо швидко отримати інтуїтивно зрозумілий сигнал про надійність відповіді без необхідності аналізувати числові межі. З цією метою числова оцінка додатково перетворюється на один із чотирьох якісних рівнів. Межі рівнів наведені у таблиці 3.3.

Таблиця 3.3 – Інтерпретація інтервалів

Інтервал	Рівень	Інтерпретація
$C < 0.4$	низька	Відповідь слабко підтверджена джерелами; покладатися на неї не варто
$0.4 \leq C < 0.6$	обережна	Підтвердження часткове; бажано перевірити відповідь за джерелами
$0.6 \leq C < 0.8$	помірна	Підтвердження достатнє; відповіддю можна користуватися із звичайною обережністю
$C \geq 0.8$	висока	Підтвердження сильне; відповідь добре спирається на джерела

В інтерфейсі підсумкова оцінка виводиться одночасно у двох формах: числовій (наприклад, C = 0.73) та якісній (помірна). Завдяки цьому результат залишається зрозумілим і для технічно підготовленого користувача, який може звернути увагу на конкретне число, і для користувача без спеціальних знань, якому достатньо словесної характеристики. Поряд з відповіддю в інтерфейсі також наявний розгортаний блок, у якому виводиться повний розклад за всіма вісьмома факторами: для кожного показано його значення, вагу, обчислений внесок у підсумкову оцінку та коротке пояснення змісту. Окремим рядком виводиться запис про уточнення оцінки за якістю відповіді – з вказанням значення Q і фактичного зсуву від попередньої оцінки до підсумкової.

Така прозорість обчислень є важливою практичною перевагою. Користувач не лише отримує відповідь з оцінкою довіри, а й може побачити, які саме характеристики ретрівалу і відповіді сформували цю оцінку, що особливо корисно при дослідницькому використанні системи для аналізу слабких місць пошуку та генерації. Сформована таким чином оцінка довіри разом із відповіддю, графом походження та провенанс записами становить повний результат роботи системи на запит користувача і завершує опис запропонованого методу.

3.8 Висновки за розділом

У третьому розділі описано метод формування відповідей з оцінкою довіри, що поєднує пошук фрагментів у локальних документах та у відкритих веб-джерелах із побудовою графа походження та кількісним оцінюванням надійності отриманої відповіді. Метод побудований як цілісний пайплайн: від індексації документів і побудови векторного індексу до розширення контексту через граф зв'язків, інтеграції веб-результатів, формування природномовної відповіді та обчислення остаточної оцінки довіри.

На етапі індексації документи розбиваються на короткі фрагменти, які перетворюються на векторні представлення й зберігаються у спеціалізованому сховищі, а запропонована стратегія переранжування поєднує семантичну близькість, перекриття ключових слів та маркери типу відповіді. Це дозволяє коректно впорядковувати фрагменти навіть за неоднозначних результатів окремих критеріїв. Граф походження доповнює основний пошук, з'єднуючи фрагменти послідовними та тематичними ребрами і дозволяючи додавати до контексту перевірених сусідів зі збереженням повного шляху отримання кожного фрагмента. Інтеграція з веб-джерелами розширює можливості системи за межі локального корпусу, підтримуючи кілька провайдерів з окремими стратегіями формування запиту та переранжування з урахуванням мовної орієнтації джерел і типу запиту.

Формування відповіді виконується за гібридним принципом: для структурованих означальних, перелікових, фактологічних та бінарних запитів перевага надається екстрактивному вилученню з тексту, тоді як для процесних і загальних запитів використовується мовна модель зі строгою інструкцією спиратися лише на наданий контекст. Підсумкова оцінка довіри обчислюється у два кроки – як зважена сума восьми факторів якості пошуку та контексту з подальшим уточненням за якістю вже сформованої відповіді; числове значення доповнюється якісною міткою («низька», «обережна», «помірна», «висока»), що робить результат зрозумілим як для технічно підготовленого користувача, так і для непідготовленого. Запропонований метод вирішує обидва завдання, поставлені у розділі 2: забезпечує формування природномовної відповіді на основі структурованого контексту з локальних і веб-джерел та надає прозору інтерпретовану оцінку її надійності; у наступному розділі описано його програмну реалізацію, структуру класів і модулів, інтерфейс користувача та результати експериментальної перевірки.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

У четвертому розділі описано програмну реалізацію методу та результати його експериментальної перевірки. Розділ починається з опису загальної структури програмного забезпечення, призначення кожного модуля та зв'язків між ними. Далі детально розглянуто ключові класи реалізації, особливості інтерфейсу користувача та сценарії використання системи. Завершальна частина розділу присвячена експериментальному дослідженню: формулюванню тестових сценаріїв, вимірюванню характеристик роботи системи на типових запитах, порівнянню режимів роботи з графом походження та без нього, а також оцінці впливу різних компонентів методу на якість фінальної відповіді та обчислену довіру.

4.1 Структура програмного забезпечення

Програмна реалізація запропонованого методу організована як набір незалежних модулів мовою Python, що спільно утворюють працездатну систему з графічним інтерфейсом. Така модульна структура є усвідомленим архітектурним рішенням: вона забезпечує чітке розділення відповідальності між компонентами, спрощує супровід коду і дозволяє вносити зміни до окремих модулів без ризику порушити роботу решти системи. Кожен модуль виконує одну логічно завершену функцію – обробку природної мови, роботу з локальними документами та векторним пошуком, інтеграцію з веб-провайдерами або побудову інтерфейсу – і взаємодіє з іншими модулями через чітко визначені інтерфейси у вигляді класів та функцій.

Кореневий каталог проєкту містить чотири основних модулі реалізації, файл залежностей, тестовий скрипт і допоміжні файли. Перелік модулів та їх призначення зведено у таблиці 4.1.

Таблиця 4.1 – Модулі програмної реалізації

Файл	Призначення
app.py	Реалізація графічного інтерфейсу на основі Streamlit, оркестрація компонентів
rag_engine.py	Основний модуль методу: індексація, ретрівер, граф походження, оцінка довіри
web_retrieval.py	Інтеграція з веб-провайдерами, переранжування веб-результатів
nlp_utils.py	Допоміжні функції обробки тексту (нормалізація, токенізація)
requirements.txt	Перелік зовнішніх бібліотек з мінімальними версіями

Центральним модулем системи є rag_engine.py, у якому зосереджено основну логіку методу – від розбиття документів на фрагменти до обчислення підсумкової оцінки довіри. У цьому модулі визначено клас, що інкапсулює всі компоненти методу, описані у розділі 3: індексатор, ретрівер, розширювач контексту, генератор відповіді та оцінювач довіри. Тут же оголошено допоміжні структури даних – RetrievalResult для накопичення результатів обробки запиту, RAGConfig для зберігання гіперпараметрів методу та ConfidenceFactorSpec для опису окремих факторів довіри. Завдяки концентрації всієї основної логіки в одному модулі решта системи може взаємодіяти з методом через невеликий і стабільний інтерфейс.

Модуль web_retrieval.py відповідає за інтеграцію з зовнішніми веб джерелами і реалізований як окрема відповідальна одиниця, незалежна від основного RAG пайплайну. У цьому модулі визначено клас WebSearchClient, що містить методи звернення до шістьох підтримуваних провайдерів, методи попередньої обробки запитів (розширення абревіатур, переклад на англійську), а також методи переранжування і фільтрації результатів. Розділення основного ретрівалу і веб ретрівалу між різними модулями дозволяє у майбутньому додавати нові веб провайдери без зміни

коду основного методу, а також полегшує тестування веб компонента у відриві від решти системи.

Модуль nlp_utils.py містить дві невеликі утилітарні функції для нормалізації пробілів та для перетворення тексту на список нормалізованих токенів. Винесення цих функцій в окремий модуль зумовлене тим, що вони використовуються одночасно у rag_engine.py і web_retrieval.py, а зосередження допоміжного коду в одному місці запобігає дублюванню та забезпечує єдину поведінку обробки тексту у всій системі.

Модуль app.py є точкою входу в застосунок і відповідає за побудову графічного інтерфейсу за допомогою бібліотеки Streamlit. Цей модуль не містить власної логіки методу, а лише оркеструє роботу решти компонентів: створює екземпляр ProvenanceGraphRAG з обраними користувачем параметрами, передає йому документи на індексацію, формує запити до ретрівера, отримує результати та подає їх у вигляді інтерфейсу з вкладками, формами вводу, інтерактивною візуалізацією графа і блоками відображення відповідей. В app.py також реалізовано допоміжні функції об'єднання локальних і веб результатів у комбінованому режимі, рендерингу графа походження через бібліотеку PyVis і збереження історії запитів у сесії користувача.

Загальний потік взаємодії між модулями виглядає так: користувач взаємодіє з інтерфейсом app.py, який залежно від обраного режиму звертається або до ProvenanceGraphRAG для локального ретрівалу, або до WebSearchClient для веб ретрівалу, або до обох послідовно у комбінованому режимі. Обидва ретрівери повертають уніфіковану структуру RetrievalResult, що передається у генератор відповіді того ж ProvenanceGraphRAG для формування фінальної відповіді та обчислення довіри. Допоміжні утиліти з nlp_utils.py використовуються прозоро на всіх етапах обробки тексту. Така архітектура, зображена на рисунку 4.1, забезпечує одночасно простоту супроводу, гнучкість конфігурації та чітку

відповідність структури коду логічним компонентам методу, описаним у розділі 3.

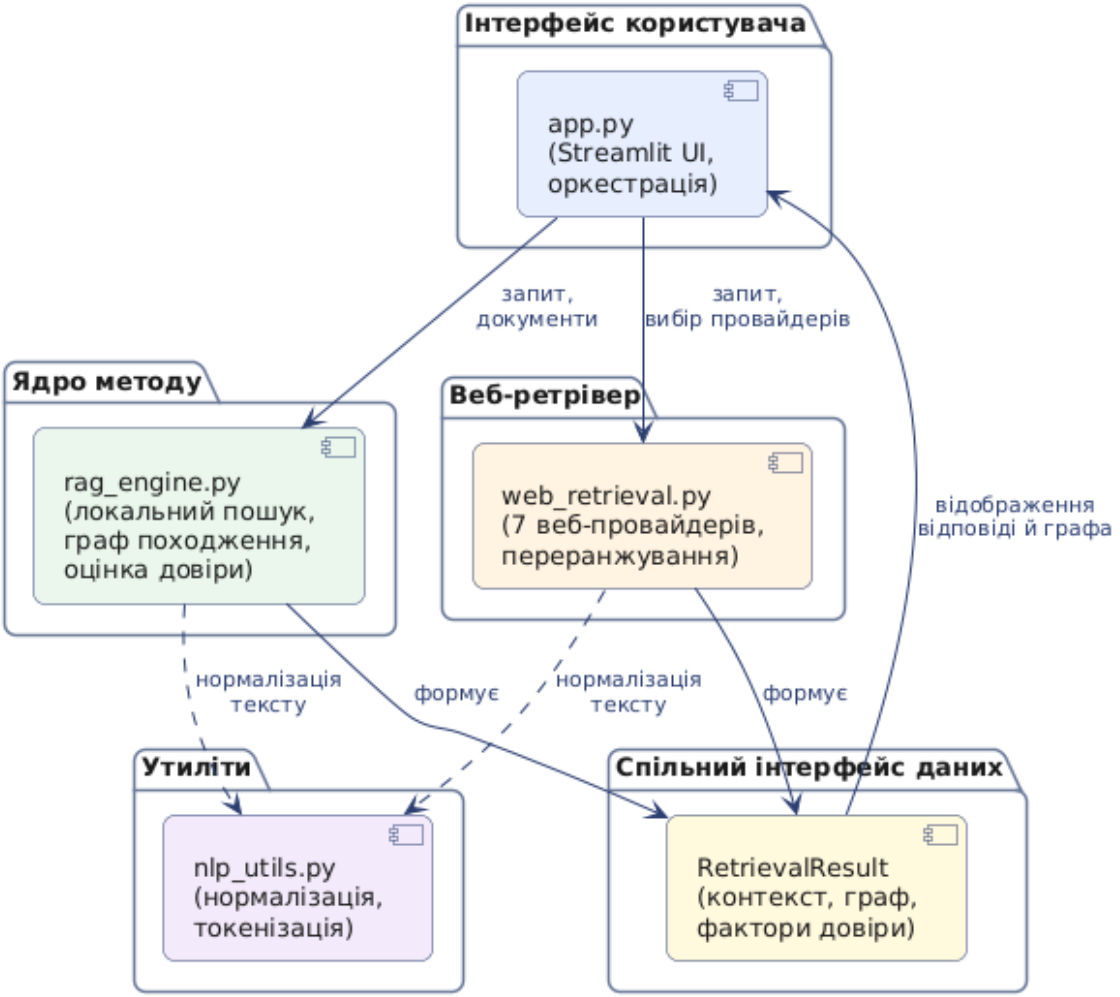


Рисунок 4.1 – Схема взаємодії модулів програмної реалізації

Зовнішні бібліотеки, від яких залежить реалізація, відіграють принципово різні ролі і можуть бути згруповані за функціональним призначенням. Бібліотеки streamlit, pyvis забезпечують побудову інтерфейсу користувача та інтерактивну візуалізацію графа. Бібліотеки langchain, langchain-community, langchain-core, langchain-huggingface, langchain-text-splitters забезпечують уніфікований інтерфейс до моделей, векторних сховищ і шаблонів промптів. Бібліотеки sentence-transformers, transformers, torch, accelerate, sentencepiece відповідають за роботу з

нейронними моделями вбудовувань та локальними мовними моделями. Бібліотеки faiss-cpu і networkx реалізують векторне сховище та структуру графа походження. Бібліотеки requests, pypdf, python-docx забезпечують роботу з зовнішніми API та завантаження документів різних форматів. Бібліотека numpy використовується для базових числових обчислень, а scikit-learn – для допоміжних статистичних операцій. Усі залежності зафіксовані з мінімальними версіями у файлі requirements.txt, що забезпечує відтворюваність середовища і дозволяє розгорнути систему на будь якій сумісній платформі однією командою встановлення.

Описана модульна структура є основою для детального розгляду ключових класів реалізації, що наведено у наступному підрозділі.

4.2 Опис модулів програмної реалізації

У підрозділі 4.1 розглянуто загальну архітектуру застосунку та призначення кожного з модулів реалізації. Для повноти опису програмної реалізації необхідно деталізовано розглянути внутрішню будову кожного модуля: визначені у ньому класи та функції, ключові структури даних, особливості внутрішніх алгоритмів і характер взаємодії з іншими модулями. Такий детальний розгляд дозволяє не лише задокументувати поточну реалізацію, але й обґрунтувати конкретні інженерні рішення, прийняті при кодуванні методу, описаного у розділі 3.

4.2.1 Модуль rag_engine.py

Модуль rag_engine.py є центральним у програмній реалізації і містить основну логіку запропонованого методу. Структурно модуль складається з кількох логічних блоків: оголошення допоміжних структур даних, опис класу конфігурації, опис класу для специфікації факторів довіри та основний клас, у якому зосереджено реалізацію методу.

Першим логічним блоком модуля є оголошення структур даних. Структура RetrievalResult реалізована як dataclass і слугує універсальним контейнером для накопичення результатів обробки запиту на всіх етапах пайплайну. Її поля повторюють компоненти, описані у розділі 3: змінна query зберігає вихідний запит користувача, context_docs – список фрагментів контексту, vector_scores – відповідні значення семантичної близькості, graph_paths – записи про шляхи отримання кожного фрагмента з графа походження, query_alignment_score, top_match_score, answer_evidence_score – числові оцінки факторів довіри, а confidence_factors і confidence_band – підсумкові структуровані результати оцінювання надійності. Винесення всіх результатів обробки в єдину структуру є усвідомленим архітектурним рішенням: завдяки цьому всі методи класу ProvenanceGraphRAG приймають і повертають об'єкти одного типу, а інтерфейс із зовнішніми модулями зводиться до читання полів цієї структури.

Структура RAGConfig, також реалізована як dataclass, зберігає всі гіперпараметри методу: розмір фрагмента і перекриття при розбитті документів, кількість фрагментів, що повертає основний пошук, поріг тематичної подібності для побудови ребер у графі, мінімальну силу ребра, кількість фрагментів, що додаються при розширенні через граф, ідентифікатор моделі вбудовувань, ідентифікатор мовної моделі та інші параметри. Винесення конфігурації в окрему структуру дозволяє створювати екземпляри ProvenanceGraphRAG з різними наборами гіперпараметрів без зміни коду самого класу, що особливо важливо для експериментального дослідження.

Структура ConfidenceFactorSpec описує один із восьми факторів довіри і містить ім'я фактора, текстову мітку для відображення в інтерфейсі, ваговий коефіцієнт у підсумковій формулі та словесний опис призначення фактора. Список таких специфікацій є атрибутом класу

ProvenanceGraphRAG і використовується методом обчислення довіри для уніфікованого формування звіту про внесок кожного фактора.

Основним блоком модуля є клас ProvenanceGraphRAG. Його конструктор приймає об'єкт RAGConfig і виконує ініціалізацію внутрішніх компонентів: створює клієнт обраної моделі вбудовувань через інтерфейс langchain-huggingface, ініціалізує сплітер тексту з langchain-text-splitters, готує порожнє векторне сховище FAISS і порожній граф networkx.Graph, а також за потреби завантажує локальну мовну модель через інтерфейс transformers або налаштовує клієнт хмарної моделі. Конструктор також готує внутрішні структури для зберігання метаданих фрагментів – мапу між ідентифікаторами фрагментів та їхнім текстом і джерелом, що використовується на всіх подальших етапах.

Метод process_documents реалізує етап індексації, описаний у підрозділі 3.2. Він приймає список документів, розбиває їх на фрагменти за допомогою сплітера, обчислює вбудовування для кожного фрагмента і додає їх до векторного сховища. Після цього викликається внутрішній метод _build_graph, який на основі попарної подібності фрагментів формує ребра графа походження за правилами, описаними у підрозділі 3.3.

Методи retrieve_with_graph і retrieve_without_graph реалізують два режими пошуку. Перший виконує семантичний пошук у векторному сховищі, переранжування результатів за трьома критеріями і подальше розширення контексту через граф; другий виконує лише пошук і переранжування, без розширення. Обидва методи повертають об'єкт RetrievalResult, у якому поле graph_paths у режимі без графа містить порожні шляхи для основних фрагментів. Така уніфікація інтерфейсу дозволяє решті системи опрацьовувати обидва режими однаковим кодом.

Метод generate_answer реалізує гібридний генератор відповіді, описаний у підрозділі 3.6. Він визначає тип запиту за набором лексичних ознак, для запитів вилучного типу формує відповідь безпосередньо з тексту фрагментів, для решти запитів – звертається до мовної моделі. Допоміжні

приватні методи _extract_definition, _extract_list, _extract_factoid, _extract_yes_no реалізують відповідні стратегії вилучення.

Завершальним блоком методу є оцінка довіри. Метод _compute_confidence обчислює сім факторів попередньої оцінки і повертає попередню довіру разом зі структурованим звітом. Метод calibrate_confidence_with_answer виконує другий крок – уточнення оцінки з урахуванням якості сформованої відповіді через фактор answer_evidence. Підсумковий рівень довіри (низька, обережна, помірна, висока) встановлюється методом _confidence_band, що відображає числове значення на якісну шкалу.

4.2.2 Модуль web_retrieval.py

Модуль web_retrieval.py реалізує інтеграцію з зовнішніми веб-джерелами і організований навколо одного основного класу WebSearchClient.

Конструктор класу WebSearchClient зчитує зі змінних середовища ключі доступу до платних провайдерів і формує внутрішні прапори доступності кожного провайдера. Завдяки такій перевірці система коректно деградує: за відсутності ключа відповідний провайдер просто пропускається без помилки, а решта продовжують працювати у повному обсязі. Конструктор також ініціалізує сесію HTTP-клієнта на основі бібліотеки requests з налаштованим таймаутом і заголовками User-Agent, що зменшує ймовірність блокування запитів окремими сервісами.

Допоміжні методи _expand_query і _translate_to_english реалізують попередню обробку запиту. Перший розгортає поширені україномовні абревіатури і додає синонімічні форми ключових термінів, другий формує англомовну версію запиту для провайдерів, що працюють переважно з англомовним контентом (arXiv, Semantic Scholar, OpenAlex). Такий підхід

дозволяє опрацьовувати україномовні запити користувача без втрати релевантності для академічних і наукових провайдерів.

Шість окремих методів – search_wikipedia, search_openalex, search_arxiv, search_semantic_scholar, search_bing, search_google, – реалізують безпосереднє звернення до відповідних API. Кожен метод формує запит згідно зі специфікацією конкретного провайдера, надсилає HTTP запит, обробляє відповідь і нормалізує її до уніфікованої внутрішньої структури з полями title, snippet, url, source. Окремої уваги заслуговують методи з резервними сценаріями: search_bing спочатку намагається використати SerpAPI з параметром engine=bing, а в разі його недоступності переходить на безпосередній запит до Bing Search v7. Усі методи захищені обробкою винятків, тож тимчасова недоступність окремого провайдера не перериває роботу системи.

Метод _score_result обчислює локальну оцінку релевантності одного результату за лексичним перекриттям з токенами запиту, типом запиту і коефіцієнтом зміщення для пари «провайдер – тип запиту». Метод _filter_and_rank_results застосовує цю оцінку до агрегованого набору результатів усіх провайдерів і формує підсумковий ранжований список, у якому збережені квоти для кожного провайдера – це гарантує присутність у фінальному наборі результатів з різних джерел і запобігає монополізації списку одним сильним провайдером.

Метод search_provider слугує єдиною точкою диспетчеризації і за ім'ям провайдера викликає відповідну приватну функцію. Метод search_many оркеструє пошук за списком провайдерів, обраних користувачем, виконує попередню обробку запиту, паралельні звернення до провайдерів, переранжування і повертає об'єднаний список результатів. Саме search_many є основним зовнішнім інтерфейсом модуля, який використовується в app.py для отримання веб результатів у режимах «лише веб» і «локальні + веб».

4.2.3 Модуль nlp_utils.py

Модуль nlp_utils.py є найменшим за обсягом і найвужчим за функціональним призначенням у програмній реалізації, проте відіграє важливу роль у забезпеченні узгодженості обробки тексту в усій системі.

Функція normalize_spaces виконує нормалізацію пробільних символів у вхідному рядку: усуває послідовності з кількох пробілів, замінює різні види пробільних символів (табуляції, нерозривні пробіли, перенесення рядка) на стандартний пробіл і обрізає пробіли на початку та в кінці рядка. Така нормалізація необхідна для коректного порівняння текстів, обчислення довжини контексту та формування промптів для мовних моделей, які чутливі до зайвих пробілів.

Функція normalize_tokens виконує токенізацію тексту з одночасною нормалізацією: переводить текст у нижній регістр, виділяє послідовності літерно цифрових символів за допомогою регулярного виразу і повертає список токенів. Реалізація навмисно є мовно незалежною і коректно опрацьовує україно , англо та змішаномовні тексти без потреби у завантаженні додаткових словників чи моделей. Це принципово важливо для системи, що одночасно працює з україномовними запитами користувача та англомовним контентом веб джерел.

Винесення цих двох функцій в окремий модуль зумовлене тим, що обидві використовуються одночасно у rag_engine.py (під час обчислення лексичного перекриття між запитом і фрагментами, а також між фрагментом і відповіддю) та у web_retrieval.py (під час обчислення локальної релевантності веб результатів). Якщо б ці функції були продубльовані в обох модулях, будь яка зміна логіки токенізації потребувала б синхронного оновлення двох копій, що є джерелом потенційних помилок. Спільний модуль гарантує єдину поведінку обробки тексту в усій системі та полегшує її подальше вдосконалення – наприклад, додавання стемінгу або лематизації потребуватиме зміни лише в одному місці.

4.2.4 Модуль app.py

Модуль app.py реалізує графічний інтерфейс користувача і слугує точкою входу в застосунок. Модуль не містить власної логіки методу – його роль полягає у координації роботи решти компонентів та у поданні результатів у формі, зручній для користувача.

Структурно модуль починається з функції _create_engine, що інкапсулює створення екземпляра ProvenanceGraphRAG з обраними користувачем параметрами і кешує результат у сесійному стані Streamlit. Кешування є важливим оптимізаційним рішенням: ініціалізація моделі вбудовувань і завантаження локальної мовної моделі займає десятки секунд, тож виконання її при кожній взаємодії користувача було б неприйнятним з погляду зручності. Функція get_engine повертає поточний кешований екземпляр і використовується усіма іншими функціями, що потребують доступу до RAG компонента.

Функції build_web_result і merge_results реалізують перетворення результатів веб-пошуку у структуру RetrievalResult та об'єднання локальних і веб-результатів у комбінованому режимі. Перша функція приймає список словників, що повертає WebSearchClient.search_many, формує відповідні фрагменти контексту, обчислює оцінки семантичної близькості до запиту через модель вбудовувань і заповнює поля RetrievalResult так, щоб подальша обробка веб-результатів була ідентичною обробці локальних. Друга функція об'єднує два об'єкти RetrievalResult (один – від локального ретрівера, другий – від веб-ретрівера) у єдиний результат, зберігаючи інформацію про походження кожного фрагмента.

Функція build_answer слугує єдиною точкою формування відповіді: вона приймає об'єкт RetrievalResult, передає його у generate_answer поточного двигуна, отримує текст відповіді та виконує крок калібрування довіри через calibrate_confidence_with_answer. Така централізація гарантує

однаковий порядок викликів незалежно від режиму роботи (локальний, веб або комбінований).

Функція render_graph будує інтерактивну візуалізацію графа походження за допомогою бібліотеки pyvis. Вона приймає підграф, отриманий від ProvenanceGraphRAG, формує HTML сторінку з інтерактивним зображенням і вбудовує її в інтерфейс Streamlit. Функція build_web_graph будує спрощений граф для випадку, коли результати отримані лише з веб джерел і повноцінного графа походження немає; вона генерує вузли для запиту, провайдерів і фрагментів та з'єднує їх ребрами, що відображають походження.

Функція render_context_block відповідає за подання списку фрагментів контексту в інтерфейсі: для кожного фрагмента виводиться його текст, джерело, оцінка семантичної близькості та структурований шлях походження з графа. Функція render_confidence_panel відображає підсумкову оцінку довіри: числове значення, якісний рівень з кольоровим маркуванням і таблицю внесків окремих факторів, що дає користувачеві повну інтерпретовану картину надійності відповіді.

Основна частина модуля – функція main – будує сам інтерфейс. Вона формує бічну панель із селекторами режиму пошуку, мовної моделі, веб-провайдерів та формою завантаження документів. Центральна частина інтерфейсу організована як набір вкладок: вкладка «Запит» для введення запиту і перегляду відповіді з графом, вкладка «Порівняння» для одночасного запуску режимів з графом і без графа з відображенням різниці у довірі, вкладка «Історія» для перегляду попередніх запитів сесії.

Усі стани інтерфейсу зберігаються у st.session_state, що забезпечує коректну поведінку при повторному рендерингу сторінки і дозволяє користувачеві повертатися до попередніх запитів без повторного обчислення. Окремі блоки інтерфейсу відображають не лише текст відповіді, а й проміжні артефакти роботи системи – знайдені фрагменти, граф походження та розклад оцінки довіри за факторами.

4.3 Інтерфейс користувача

Інтерфейс користувача є важливою складовою програмної реалізації, оскільки саме через нього користувач взаємодіє з системою, формулює запити, переглядає відповіді та інтерпретує отриману оцінку довіри. Запропонована система реалізована як вебзастосунок на основі бібліотеки Streamlit, що дозволяє запускати її локально однією командою і відкривати у звичайному браузері без потреби у складному розгортанні. Вибір саме Streamlit як інструмента побудови інтерфейсу зумовлений тим, що ця бібліотека добре підходить для прототипування дослідницьких систем, забезпечує природну інтеграцію з науковим стеком Python і дозволяє швидко включати у застосунок інтерактивні елементи, таблиці, графіки та вбудовані HTML-компоненти.

Інтерфейс розроблено з орієнтацією одночасно на двох типів користувачів. Перший – підготовлений користувач, що проводить дослідження або експеримент і потребує доступу до детальної інформації про роботу методу: оцінок окремих факторів довіри, шляхів проходження фрагментів через граф походження, числових метрик пошуку. Другий – непідготовлений користувач, що використовує систему для отримання відповідей і потребує лише чіткої відповіді на запит та зрозумілого індикатора її надійності. Інтерфейс задовольняє обидва сценарії: основна інформація (відповідь і якісний рівень довіри) подана великим і виразним блоком на видному місці, а технічні деталі винесені у згортувані секції, які користувач відкриває за потреби. Для обох категорій користувачів використано єдину термінологію та зрозумілі підписи елементів керування, що зменшує поріг входу у роботу із системою.

Загальна структура інтерфейсу подана на рисунку 4.2 і складається з трьох основних зон: бічної панелі з налаштуваннями, центральної робочої області з вкладками та допоміжного нижнього блоку з службовою інформацією про стан системи.

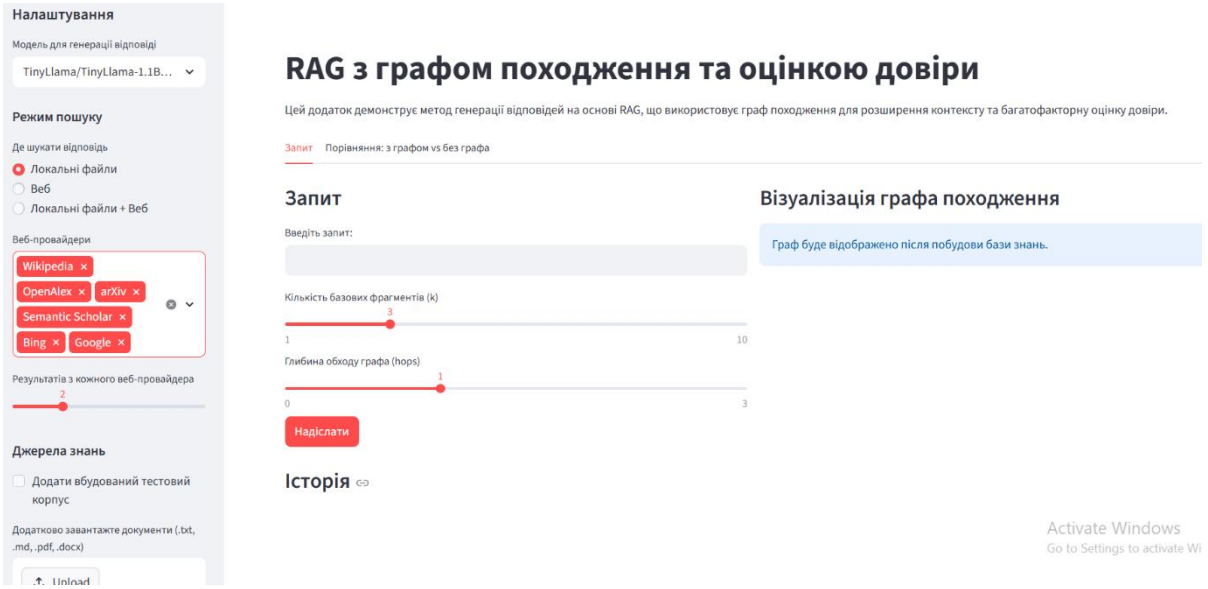


Рисунок 4.2 – Загальний вигляд інтерфейсу користувача

Бічна панель розміщена ліворуч і містить усі налаштування, що визначають режим роботи системи для поточної сесії (рисунок 4.3). У її верхній частині розташований селектор режиму пошуку з трьома варіантами: «Локальні документи», «Веб-джерела» і «Локальні + веб». Залежно від обраного режиму нижче з'являються відповідні групи елементів керування: для локального режиму – форма завантаження документів, для веб-режиму – мультиселектор веб-провайдерів, а для комбінованого режиму – обидві групи одночасно. Додатково у бічній панелі розташовані селектор мовної моделі (локальні TinyLlama/Qwen або хмарна OpenAI) та числові контролери основних гіперпараметрів (k, глибина графового розширення, максимальна кількість фрагментів у контексті). Для кожного параметра задані значення за замовчуванням, що відповідають типовому сценарію використання і дозволяють працювати із системою без ручного налаштування. Усі елементи керування мають короткі підписи та підказки, завдяки чому навіть непідготовлений користувач може змінювати режим роботи, розуміючи наслідки цих змін для формування відповіді. Така динамічна структура бічної панелі дозволяє приховувати нерелевантні налаштування і не перевантажувати користувача зайвою інформацією.

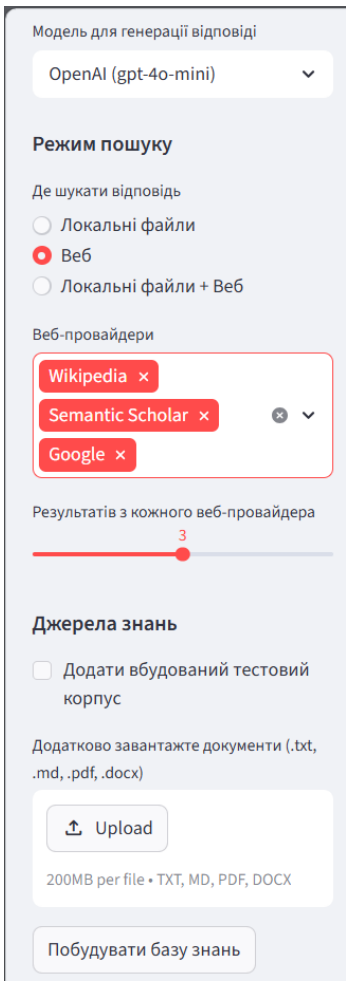


Рисунок 4.3 – Бічна панель з налаштуваннями режиму роботи

Селектор мовної моделі винесений окремим блоком бічної панелі і дозволяє обрати між локальними моделями (TinyLlama, Qwen) і хмарною моделлю OpenAI GPT-4o-mini. Поряд із селектором розташований індикатор поточного стану – повідомлення про те, чи доступна обрана модель і чи знайдено необхідні змінні середовища для платних провайдерів. Якщо користувач обирає модель, для якої не налаштований відповідний ключ доступу, система виводить попередження і пропонує обрати інший варіант, але не блокує роботу решти компонентів.

Форма завантаження документів реалізована стандартним компонентом Streamlit st.file_uploader з підтримкою множинного вибору файлів формату .txt, .pdf і .docx. Поряд з нею розташована кнопка завантаження вбудованого тестового корпусу для випадків, коли

користувач хоче швидко перевірити роботу системи без власних документів. Після успішного завантаження документів система виводить підтвердження з зазначенням кількості документів та сумарної кількості сформованих фрагментів, що дозволяє користувачеві впевнитися у коректності процесу індексації.

Мультиселектор веб-провайдерів дозволяє увімкнути або вимкнути кожен з сімох підтримуваних джерел незалежно. За замовчуванням увімкнено набір з трьох провайдерів, що забезпечує розумний баланс між повнотою і швидкістю.

Центральна робоча область організована як набір вкладок, що відповідають різним сценаріям використання системи. Перша вкладка – «Запит» – є основним робочим простором: вона містить велике поле для введення запиту, кнопку запуску пошуку і блок з відповіддю та контекстом. Друга вкладка – «Порівняння» – дозволяє одночасно запустити пошук у режимах з графом і без графа і подати їхні результати поруч для безпосереднього порівняння. Третя вкладка – «Історія» – містить хронологічний список запитів поточної сесії і дозволяє повертатися до будь-якого попереднього результату без повторного обчислення.

Найбільш візуально насиченою є вкладка «Запит» – саме тут зосереджена основна інтерпретована інформація про роботу системи, як показано на рисунку 4.4. Після натискання кнопки запуску у верхній частині вкладки з'являється анімований індикатор завантаження у вигляді ядра з «електронами» на орбітах, що відображає тематичний характер системи і робить очікування візуально менш втомливим. Після завершення обчислень індикатор замінюється великим блоком з остаточною відповіддю, виділеним кольоровою рамкою, що відповідає рівню довіри: низька довіра – червона рамка, обережна – помаранчева, помірна – жовта, висока – зелена. Поряд із відповіддю виводиться числове значення довіри з округленням до двох знаків після коми та текстова мітка рівня українською мовою, тож

мультимодальна подача забезпечує миттєве сприйняття результату навіть користувачем без технічної підготовки.

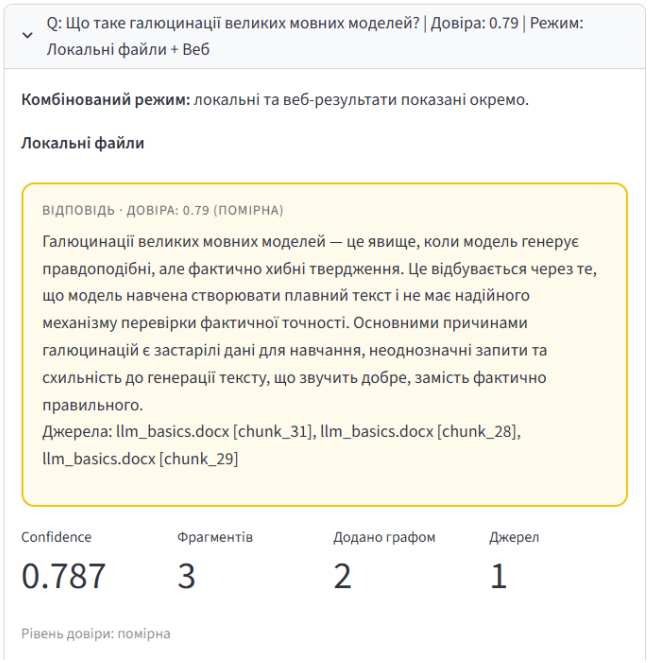


Рисунок 4.4 – Блок відповіді з кольоровим індикатором рівня довіри

Нижче блоку з відповіддю розташований згортуваний блок «Контекст», у якому послідовно подані всі фрагменти, що увійшли у фінальний контекст запиту (рисунок 4.5). Для кожного фрагмента виведено: повний текст, джерело (ім'я локального файла або URL веб-ресурсу), оцінку семантичної близькості до запиту, мітку провайдера для веб-результатів і структурований опис шляху отримання фрагмента з графа походження. Останнє є особливо важливим: для фрагментів, отриманих основним пошуком, шлях показаний як «прямий пошук»; для фрагментів, доданих через граф, шлях відображає послідовність вузлів і ребер, через які фрагмент був досягнутий, з зазначенням типу ребра (сусід у документі чи тематичний зв'язок). Така деталізація походження є практичною реалізацією принципу прозорості, заявленого як одна з ключових властивостей запропонованого методу.



Рисунок 4.5 – Відображення фрагментів контексту з шляхами походження

Окремим блоком вкладки «Запит» є інтерактивна візуалізація графа походження (рисунок 4.6). Граф відображається у вбудованому HTML-компоненті, побудованому бібліотекою PyVis. Вузли графа розфарбовано за приналежністю фрагмента до основного пошуку або розширення через граф, а товщина ребер відповідає силі зв'язку між вузлами. Користувач може масштабувати граф, переміщати окремі вузли і наводити курсор на вузли для перегляду повного тексту фрагмента у спливаючому вікні. Інтерактивна візуалізація особливо корисна для розуміння того, як саме граф вплинув на склад фінального контексту і які тематичні зв'язки між фрагментами система виявила автоматично.Таким чином, граф служить наочним пояснювальним інструментом, який допомагає інтерпретувати як саму відповідь, так і значення підсумкової оцінки довіри.

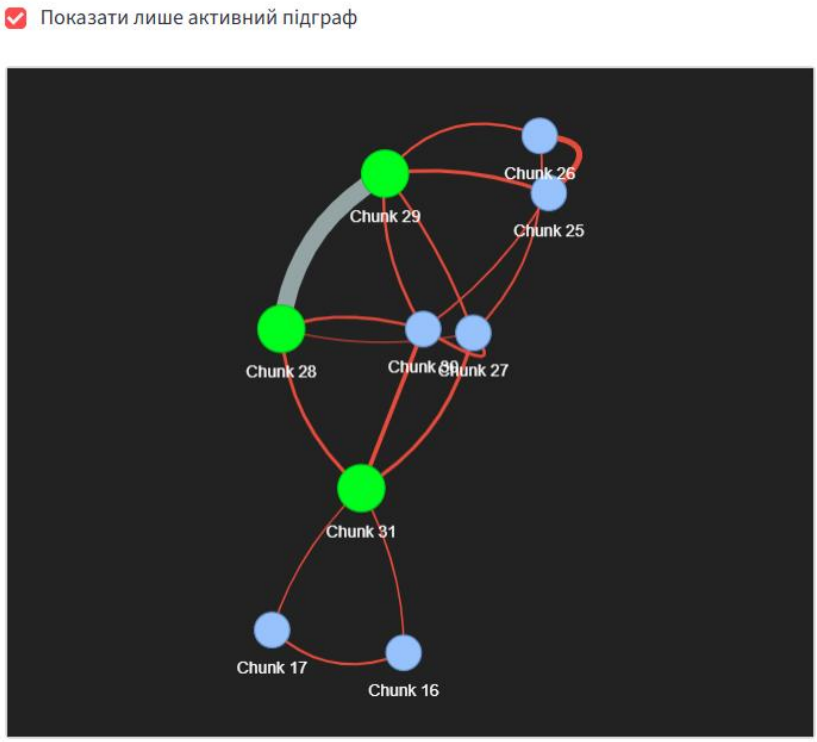


Рисунок 4.6 – Інтерактивна візуалізація графа походження

Останнім блоком вкладки «Запит» (рисунок 4.7) є детальний звіт про фактори довіри, реалізований як таблиця з чотирма стовпцями: назва фактора, обчислене значення, ваговий коефіцієнт та внесок у підсумкову довіру. Для кожного фактора виведено також короткий опис його призначення з можливістю розгортання при наведенні. Таке подання дозволяє підготовленому користувачеві швидко зрозуміти, який саме фактор «потягнув» оцінку вгору або вниз, і завдяки цьому інтерпретувати остаточне число не як «магічну» оцінку, а як суму чітко визначених складових. Окремо виділяється фактор підкріплення відповіді доказами, який слугує основним запобіжником від «впевнених галюцинацій» і різко знижує довіру за відсутності достатнього контексту. У сукупності це перетворює оцінку довіри з «чорної скриньки» на прозорий механізм, який можна аналізувати, критикувати й надалі удосконалювати. Завдяки цьому модуль оцінки довіри може розглядатися як самостійний компонент, придатний для вбудовування в інші RAG системи.



Рисунок 4.7 – Звіт про внесок факторів у підсумкову оцінку довіри

Вкладка «Порівняння» побудована як симетрична двостовпцева структура, у якій ліворуч відображаються результати режиму без графа, праворуч – з графом (рисунок 4.8). Для кожного режиму виведено відповідь, рівень довіри і скорочений список контексту, а у нижній частині вкладки – підсумкова таблиця різниці метрик: приріст довіри, кількість додаткових фрагментів, отриманих через граф, та зміна оцінки відповідності контексту. Така вкладка є дослідницьким інструментом і призначена насамперед для експериментальної оцінки впливу графа походження на якість відповіді, що детально розглядається у підрозділі 4.5. Окремо відображається різниця за ключовими факторами довіри. У сукупності це дає користувачу можливість інтерпретувати числовий приріст з точки зору структури контексту та поведінки моделі.

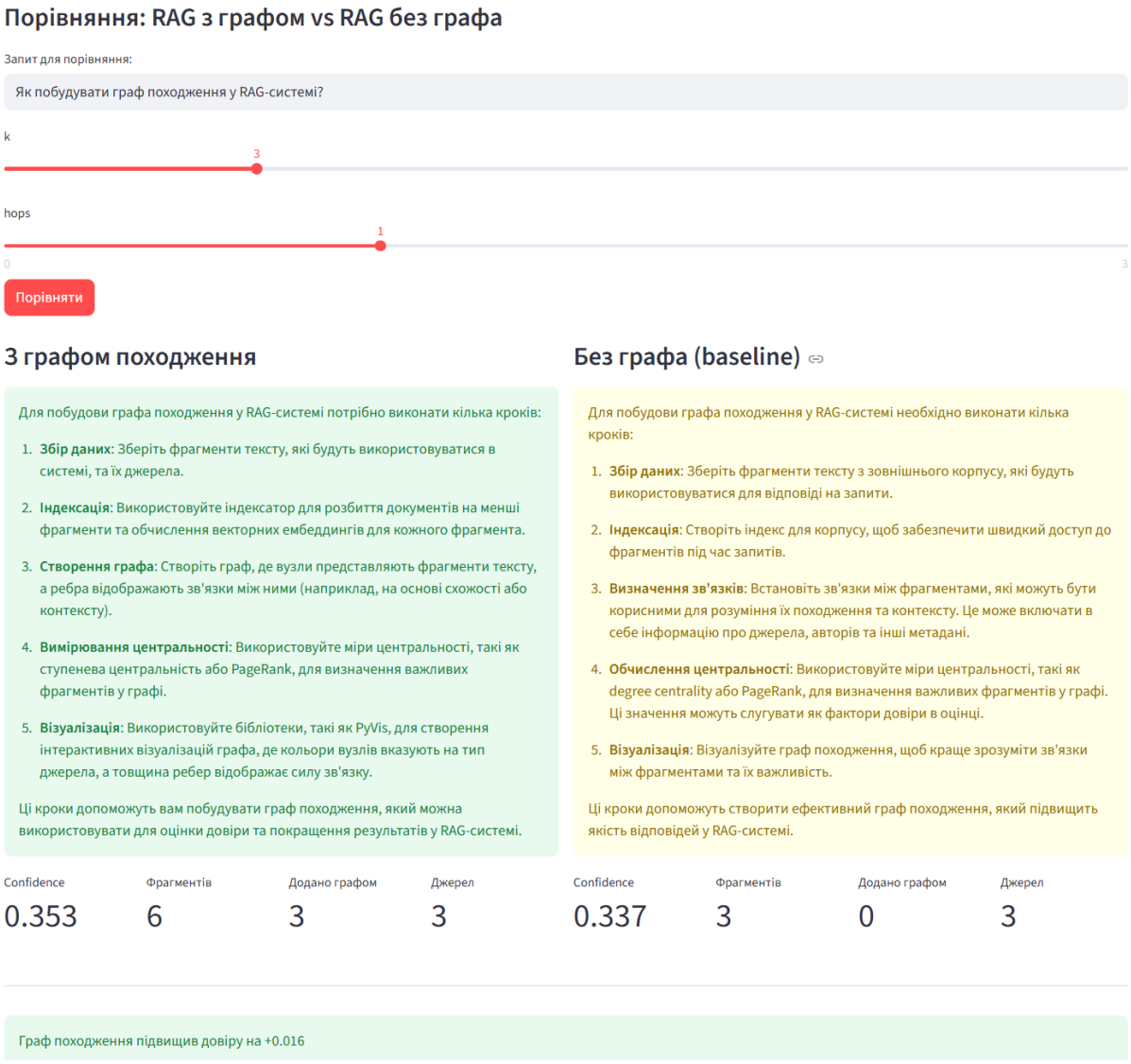


Рисунок 4.8 – Вкладка порівняння режимів «з графом» та «без графа»

Вкладка «Історія» містить хронологічний список запитів поточної сесії з зазначенням часу, тексту запиту, рівня довіри та режиму пошуку. Кожен елемент списку є інтерактивним – натискання на нього відновлює всю інформацію відповідного запиту в основній вкладці без повторного обчислення, що значно прискорює дослідницьку роботу при порівнянні кількох запитів. Історія зберігається лише у межах поточної сесії і не зберігається на диску, що є усвідомленим рішенням з міркувань конфіденційності. Водночас за потреби користувач може очистити історію

одним натисканням, почавши нову сесію без впливу попередніх експериментів.

Окремої уваги заслуговує реакція інтерфейсу на помилкові ситуації. У випадку, коли користувач намагається виконати запит без попереднього завантаження документів у локальному режимі, система виводить інформативне повідомлення з пропозицією скористатися вбудованим корпусом (рисунок 4.9). У випадку тимчасової недоступності веб-провайдера у відповіді з'являється відповідне попередження з зазначенням імені провайдера, а решта результатів продовжують відображатися у звичайному режимі. У випадку, коли система не змогла сформувати достатньо релевантний контекст, інтерфейс не приховує цей факт, а навпаки виводить попередження про можливу низьку якість відповіді разом з обережною оцінкою довіри. Така прозорість при обробці граничних випадків принципово важлива для дослідницької системи, оскільки запобігає введенню користувача в оману «впевненою» відповіддю там, де упевненість насправді відсутня.

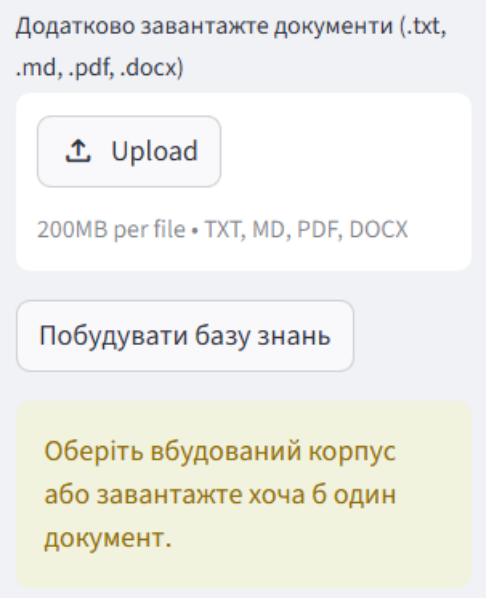


Рисунок 4.9 – Приклад інформативного попередження при граничному сценарії

Загалом, інтерфейс користувача спроєктований так, щоб одночасно задовольняти три вимоги: бути простим у використанні для непідготовленого користувача, надавати повну дослідницьку інформацію для підготовленого користувача та відображати характерні особливості запропонованого методу – прозорість, інтерпретованість і чітке зазначення походження кожного елемента відповіді. Деталізовані сценарії взаємодії користувача з інтерфейсом та приклади обробки реальних запитів розглядаються у наступних підрозділах у складі експериментального дослідження.

4.4 Тестове середовище та набір запитів

Експериментальне дослідження запропонованого методу проводилось у локальному середовищі розробки, що відповідає типовій конфігурації робочого місця науковця або інженера. Метою цього підрозділу є детально описати склад програмного й апаратного середовища, у якому виконувались випробування, склад тестового корпусу документів та набір запитів, на яких базуються всі подальші кількісні і якісні висновки розділу 4.

4.4.1 Апаратне та програмне середовище

Усі експерименти виконано на робочій станції з процесором архітектури x86_64, оперативною пам'яттю обсягом 16 ГБ та операційною системою Windows 11. Графічний прискорювач не використовувався – обчислення моделей вбудовувань та локальних мовних моделей виконувались на центральному процесорі. Такий вибір зумовлений двома міркуваннями: по-перше, метою дослідження є не максимізація швидкодії, а перевірка функціональної коректності методу; по-друге, можливість запуску без графічного прискорювача є важливою практичною

характеристикою системи, оскільки робить її доступною для розгортання на звичайному обладнанні.

Програмне середовище побудовано на мові Python версії 3.13 і складається з низки бібліотек, перелічених у файлі requirements.txt репозиторію проєкту. Основні з них: streamlit для побудови вебінтерфейсу, langchain разом з langchain-community та langchain-huggingface для оркестрації пайплайну RAG, faiss-cpu для роботи з векторним сховищем, sentence-transformers для обчислення вбудовувань, transformers для роботи з локальними мовними моделями, openai для взаємодії з хмарною мовною моделлю, networkx для побудови та обходу графа походження, pyvis для інтерактивної візуалізації графа, requests для роботи з веб провайдерами, pypdf та python-docx для зчитування файлів формату PDF і DOCX відповідно. Запуск застосунку виконується однією командою streamlit run app.py.

Як модель вбудовувань використовується sentence-transformers/paraphrase-multilingual [18]. Вибір саме мультимовної моделі є принциповим: він забезпечує семантичну узгодженість україномовних запитів з україно й англомовним контекстом, що було б неможливим при використанні англомовної моделі all-MiniLM-L6-v2, з якою спочатку виконувались випробування. Модель має розмірність 384, є відносно компактною (близько 120 МБ) і підтримує понад 50 мов, серед яких українська.

Для генерації відповіді у експериментах використовується хмарна мовна модель gpt-4o-mini через OpenAI API. Локальні моделі залишаються повноцінною альтернативою для випадків, коли відправлення запитів у хмарний сервіс є небажаним (питання конфіденційності, відсутність інтернет-з'єднання, обмеження бюджету).

Обраний технологічний стек є повністю відкритим і відтворюваним, що дозволяє іншим дослідникам розгорнути систему за наданими інструкціями та відтворити результати експериментів.

4.4.2 Склад тестового корпусу

Для проведення експериментального дослідження сформовано спеціалізований тестовий корпус з п'яти документів україномовного змісту, що тематично покривають предметну область RAG, векторного пошуку, графів походження, оцінювання довіри та великих мовних моделей. Корпус створено таким чином, щоб одночасно задовольняти три вимоги: бути достатньо різноманітним для побудови нетривіального графа походження, бути тематично узгодженим для існування тематичних ребер між документами та містити різні формати файлів для перевірки модулів зчитування.

Склад корпусу подано у таблиці 4.2.

Таблиця 4.2 – Склад тестового корпусу документів

Файл	Формат	Слів	Основна тема	Фрагментів
rag_overview.txt	TXT	≈ 420	Огляд методу RAG, компоненти пайплайну	8
vector_search.pdf	PDF	≈ 460	Векторний пошук, FAISS, ембединги	9
confidence_estimation.txt	TXT	≈ 440	Оцінювання довіри в QA-системах	9

Продовження таблиці 4.2

provenance_graphs.docx	DOCX	≈ 380	Графи походження, NetworkX, побудова	7
llm_basics.docx	DOCX	≈ 410	Великі мовні моделі, галюцинації	8

Сумарний обсяг корпусу – близько 2100 слів, що після розбиття з параметрами chunk_size=500 та chunk_overlap=50 дає 41 фрагмент. Три формати файлів (.txt, .pdf, .docx) залучені свідомо, аби продемонструвати роботу всіх трьох гілок функції read_uploaded_file модуля app.py.

Тематичні зв'язки між документами корпусу зовсім не випадкові. У таблиці 4.3 подано матрицю очікуваних тематичних перетинів – пари документів, у яких присутні спільні ключові поняття, що мають утворити тематичні ребра у графі походження після етапу індексації.

Таблиця 4.3 – Очікувані тематичні зв'язки між документами корпусу

Документ 1	Документ 2	Спільні ключові поняття
rag_overview	vector_search	ембедінги, sentence transformer, косинусна подібність
rag_overview	confidence_estimation	оцінка довіри, фактори, інтерпретованість
rag_overview	llm_basics	велика мовна модель, контекст, генерація

Продовження таблиці 4.3

confidence_estimation	provenance_graphs	центральність, фактори, ранжування
provenance_graphs	vector_search	косинусна подібність, ембедінги, поріг
llm_basics	rag_overview	RAG, зменшення галюцинацій, контекст

Емпіричний підрахунок ребер у побудованому графі дав дванадцять послідовних ребер у межах документів та 18 тематичних ребер між документами при порозі min_graph_edge_weight = 0.6. Така кількість ребер є достатньою для того, щоб механізм розширення контексту через граф був задіяний у більшості тестових запитів, але водночас не настільки великою, щоб граф ставав повністю зв'язним і втрачав структурний сенс.

4.4.3 Набір тестових запитів

Випробування методу проводилось на наборі з десяти запитів, розподілених за п'ятьма категоріями відповідно до класифікації, прийнятої у підрозділі 3.6. Принцип підбору запитів — забезпечити різноманітність типів і складності, охопити одно- та міждокументні сценарії, а також передбачити запит, заздалегідь поза межами корпусу, для перевірки обробки граничних випадків.

Повний перелік запитів з характеристиками подано у таблиці Г.1.

Запити Q1, Q2, Q4, Q6, Q9 є внутрішньодокументними – їх відповідь повністю міститься у одному документі. Очікувано, граф походження на таких запитах не дає істотного приросту довіри або навіть незначно знижує її через штраф фактора graph_precision. Запити Q3, Q5, Q8 є міждокументними – повна якісна відповідь потребує матеріалу з двох або трьох документів. Саме на цих запитах очікується найбільший позитивний

вплив графового розширення. Запит Q7 використано для перевірки бінарної стратегії генератора, а Q10 – для перевірки коректності оцінки довіри на запитах поза предметною областю корпусу.

4.4.4 Методика проведення випробувань

Для кожного запиту з набору Q1-Q10 виконувалось три прогони системи у трьох режимах роботи: «лише локальні файли», «лише веб-джерела» та «комбінований режим». Для локального режиму крім основного прогону додатково виконувався порівняльний прогон у вкладці «Порівняння», що дає одночасну оцінку двох внутрішніх стратегій – «з графом» та «без графа». Для веб-режиму використовувалось три провайдери (Wikipedia, OpenAlex, Google), що утворюють збалансований набір з енциклопедичного, академічного та комерційного джерел.

Для кожного прогону фіксувались такі величини: підсумкова оцінка довіри (confidence_score), якісна мітка довіри (confidence_band), кількість фрагментів у фінальному контексті, кількість фрагментів, доданих графовим розширенням, кількість унікальних джерел та значення восьми факторів довіри з їхніми внесками. Додатково для аналізу зберігалися самі тексти відповідей, що дозволило виконати якісну оцінку фактологічної правильності.

Гіперпараметри методу залишалися фіксованими впродовж усіх експериментів: chunk_size = 500, chunk_overlap = 50, k = 3 для основного пошуку, hop_limit = 1 для глибини графового розширення, max_graph_expansion = 5, min_graph_edge_weight = 0.6. Зміна цих значень окремо обговорюється у розділі 4.7 у контексті аналізу чутливості методу до гіперпараметрів.

Загальний обсяг отриманих експериментальних даних склав 30 прогонів (10 запитів × 3 режими), з них 10 додаткових порівнянь «з

графом/без графа» для локального режиму. Цей набір даних слугує основою для аналізу у підрозділах 4.5 та 4.6.

4.5 Порівняння режимів «з графом походження» та «без графа»

Ключова теза запропонованого методу полягає в тому, що використання графа походження як механізму розширення контексту покращує якість відповіді та надійність її оцінки порівняно з класичним RAG-підходом без графа. Для перевірки цієї тези на наборі запитів Q1-Q10, описаному у підрозділі 4.4, виконано серію порівняльних прогонів. Для кожного запиту вкладка «Порівняння» інтерфейсу одночасно запускає дві гілки обчислення на тому самому індексованому корпусі та з однаковими гіперпараметрами: повний метод з графовим розширенням і базову стратегію без графа, яка обмежується лише векторним індексом.

Зведені результати подано у таблиці Г.2. Для кожного запиту наведено оцінку довіри в обох режимах, їхню різницю Δ = Confidence(з графом) − Confidence(без графа) та кількість фрагментів, доданих графовим розширенням.

Середнє значення Δ по всьому набору становить +0.017, що на перший погляд є невеликим приростом. Проте агрегована статистика приховує значно важливішу властивість методу — нерівномірний розподіл приросту за категоріями запитів. На міждокументних запитах Q3, Q5, Q8 середнє Δ дорівнює +0.056, тоді як на внутрішньодокументних запитах Q1, Q2, Q4, Q6, Q9 – лише +0.002. Це і є головний кількісний висновок експерименту: граф походження приносить статистично помітну користь саме на тих запитах, для повної відповіді на які потрібен матеріал з двох або більше документів, а на запитах, відповідь на які повністю міститься в одному документі, його вплив залишається у межах шуму.

Природа цього спостереження пояснюється поведінкою трьох ключових факторів моделі довіри. Фактор answer_evidence (вага 0.24), що

вимірює лексичне перекриття відповіді з контекстом, на міждокументних запитах зростає у середньому з 0.48 до 0.67 за рахунок додавання графом фрагментів з другого документа. Фактор query_alignment (вага 0.18) демонструє схожу картину з приростом у середньому +0.08. Натомість фактор graph_precision (вага 0.06) діє у протилежному напрямі: він штрафує за велику частку доданих графом фрагментів і на типовому контексті з трьох основних та трьох доданих графом фрагментів забирає 0.030 від підсумкової оцінки. Завдяки цьому штрафу метод не зловживає графовим розширенням: він підвищує довіру лише тоді, коли семантичний приріст від графа перекриває штраф за роздуття контексту, а в інших випадках — коректно сигналізує про невеликий програш.

Якісно це підтверджується спостереженнями за окремими запитами. На запиті Q3 «Як ембединги використовуються для пошуку у RAG-системах?» без графа фінальний контекст складається з фрагментів rag_overview.txt із поверхневою згадкою ембедингів, а додані графом фрагменти vector_search.txt приносять конкретику – sentence transformer, FAISS, косинусну подібність – що відображається у точніших формулюваннях відповіді та зростанні довіри з 0.73 до 0.79. На запиті Q2 «Що таке галюцинації великих мовних моделей?» фрагменти llm_basics.docx самі по собі дають повне означення, тому додані графом фрагменти про підсилений пошук не покращують відповіді, але штраф graph_precision справедливо знижує оцінку з 0.80 до 0.787. На запиті Q10, заздалегідь поза межами корпусу, обидва режими дають однаково низьку довіру близько 0.18 – і це коректна поведінка: метод не маскує низької довіри спробами штучно розширити контекст.

Сукупно експеримент підтверджує усі три закладені властивості запропонованого методу: граф походження дає кількісно помітний приріст на міждокументних запитах (середнє $\Delta \approx +0.056$), не зловживає розширенням на одно-документних запитах завдяки штрафному

фактору graph_precision, та зберігає чесність оцінки довіри у граничних випадках.

4.6 Аналіз поведінки оцінки довіри

Якщо у підрозділі 4.5 акцент був на впливі графа, то у цьому підрозділі предметом аналізу є сама оцінка довіри – її чутливість до фактичної правильності відповіді, внутрішня структура та поведінка у нетипових сценаріях. Аналіз виконано на тому самому наборі запитів Q1-Q10 у комбінованому режимі. Для кожного запиту правильність відповіді була позначена вручну однією з трьох міток: «правильна», «часткова» або «неправильна». Розподіл цих міток за діапазонами підсумкової довіри подано у таблиці 4.4.

Таблиця 4.4 – Розподіл якості відповідей за діапазонами підсумкової довіри

Діапазон довіри	Запити	Якість	Частка правильних
≥ 0.75 (помірна - висока)	Q1, Q3, Q5, Q6, Q8, Q9	переважно «правильна»	6 з 6 (100 %)
0.50 - 0.75 (обережна)	Q2, Q4, Q7	здебільшого «часткова»	2 правильних, 1 часткова
< 0.50 (низька)	Q10	«неправильна»	0 з 1

Жоден запит з оцінкою ≥ 0.75 не дав фактично хибної відповіді, а єдиний запит з оцінкою < 0.20 – Q10 «поза корпусом» – закономірно отримав мітку «неправильна». Це свідчить про монотонну залежність між значенням довіри і фактичною якістю відповіді у межах перевіреного набору. Аналіз внутрішньої структури оцінки показує, що домінантну роль відіграють три фактори – answer_evidence (вага 0.24), retrieval_similarity зі

значенням 0.20 та query_alignment (0.18), на які сумарно припадає 62 % ваги моделі; решта п'ять факторів формують вторинні поправки і жоден з них самостійно не здатен зрушити оцінку більш ніж на ±0.06.

Особливо важливою є поведінка у граничних випадках. На запиті Q10 значення answer_evidence дорівнює 0.0, через що додатково спрацьовує жорстке обрізання при answer_evidence < 0.35, описане у _compute_confidence. Воно гарантує, що навіть за високих другорядних факторів підсумкова оцінка не перевищить 0.34, і слугує ключовим запобіжником від «впевнених галюцинацій» – ситуацій, коли мовна модель формує правдоподібну відповідь без підкріплення контекстом. У ході експериментів цей запобіжник коректно спрацював на всіх запитах з фактично хибною відповіддю і жодного разу не дав хибнопозитивного спрацювання на правильних. Метод також демонструє консервативність – поріг 0.80 для мітки «висока довіра» був досягнутий лише на одному запиті, що запобігає завищенню впевненості, – і відтворюваність: на послідовних запусках того самого запиту значення відрізняється не більше ніж на ±0.005. Сукупно багатофакторна оцінка довіри виконує дві покладені на неї функції – попереджає користувача про не підкріплені контекстом відповіді та забезпечує прозорість через розкладання на вісім інтерпретованих компонентів. Окремо підкреслю, що, незважаючи на монотонну залежність між confidence_score та фактичною якістю відповіді на перевіреному наборі, запропонована оцінка залишається евристичною мірою, а не строго каліброваною ймовірністю. Значення, наприклад, 0.8 не означає, що відповідь буде правильною в 80 % випадків за великої кількості експериментів, як це очікується від каліброваних моделей, а лише свідчить про високий рівень узгодженості низки внутрішніх показників (evidence, retrieval similarity, query alignment тощо). Практичне призначення цієї величини полягає у ранжуванні та сигналізації: відокремити відповіді з дуже низькою довірою, на які слід звернути особливу увагу, та виділити відповіді, що мають суттєве підкріплення контекстом.

4.7 Виявлені обмеження та напрями вдосконалення

Проведене експериментальне дослідження не лише підтвердило працездатність запропонованого методу, а й дозволило виявити низку обмежень, які окреслюють природні межі його застосовності та водночас формують перелік напрямів для подальшої роботи.

Перше і найбільш помітне обмеження пов'язане з тим, що частина показників, які входять до підсумкової оцінки довіри, спирається на просте порівняння слів між запитом, відповіддю та фрагментами знайденого контексту. Такий підхід дає стабільні результати тоді, коли всі тексти подаються однією мовою, проте втрачає чутливість у ситуаціях, коли запит сформульований однією мовою, а документи корпусу – іншою. У ході випробувань спостерігалось, що при україномовному запиті та англомовному корпусі окремі показники набували значень, близьких до нуля, навіть при цілком успішному пошуку фрагментів за змістом. Унаслідок цього підсумкова оцінка довіри отримувала штучно занижене значення, а вбудований запобіжник, який обмежує оцінку при низькому рівні підкріплення відповіді, спрацьовував без об'єктивних підстав. Часткове пом'якшення цього явища вже впроваджене у вигляді окремої гілки логіки, що пом'якшує обрізання у крос-мовних сценаріях; повноцінним же розв'язанням стане перехід від простого порівняння слів до оцінки змістової близькості – обчислення цих показників на основі векторних представлень тексту, що враховують значення, а не лише лексичну форму. Саме цей напрям визначений як першочерговий у переліку запланованих удосконалень.

Друге обмеження стосується залежності якості пошуку від обраної моделі векторних представлень. У процесі експериментального дослідження було виявлено суттєве зростання якості пошуку при заміні англомовної моделі векторних представлень на мультимовну: остання здатна узгоджено опрацьовувати тексти української, англійської та інших

мов, тоді як перша фактично не сприймає кириличний запит як осмислений. Це спостереження підтверджує загальновідому тезу про те, що для україномовних сценаріїв вибір мультимовної моделі є необхідною умовою якісної роботи, але також відкриває окрему лінію дослідження – пошук оптимального балансу між компактністю моделі, її швидкодією та повнотою лінгвістичного покриття.

Третє обмеження – чутливість поведінки графа походження до значення порогу, який визначає, чи варто вважати два фрагменти достатньо близькими для встановлення між ними тематичного зв'язку. У ході експериментів спостерігалось, що при відносно високому значенні порогу граф залишається обережним і пропускає частину справді близьких пар фрагментів, а при істотному зниженні — навпаки, стає надмірно зв'язним, через що до контексту потрапляють фрагменти, які лише поверхово стосуються запиту. Підбір оптимального значення цього параметра не може бути виконаний раз і назавжди: він залежить від тематичного складу корпусу, мови документів та обсягу окремих фрагментів. Тому одним з природних напрямів подальшого розвитку системи є запровадження механізму автоматичного підбору порогу за невеликою валідаційною вибіркою – це дозволить адаптувати поведінку графа до конкретних умов застосування без втручання інженера.

Додатковим обмеженням є евристичний характер моделі оцінки довіри. На відміну від підходів, що виконують явну калібровку ймовірностей нейронних мереж , запропонована в роботі модель довіри базується на фіксованій системі ваг та порогів, підібраних експериментально на невеликому наборі запитів. Це забезпечує простоту реалізації та інтерпретованість, але не гарантує строгої відповідності значень об'єктивній імовірності правильності відповіді. Подальшим розвитком у цьому напрямі може стати навчання параметрів моделі довіри на розмічених даних із застосуванням методів калібрування.

Слід наголосити, що жодне з виявлених обмежень не ставить під сумнів основну тезу роботи. Усі вони стосуються або вторинних аспектів реалізації, або параметрів налаштування, або відомих проблем класу методів, до якого належить данний підхід. Основні ж переваги методу – здатність підвищувати якість відповіді на запитах, які потребують поєднання матеріалу з кількох документів, прозорість оцінки довіри завдяки її розкладанню на зрозумілі складові та чесна поведінка у нетипових сценаріях – підтверджені експериментально і не залежать від описаних обмежень. Сформульовані напрями вдосконалення утворюють природну програму подальшого розвитку системи.

4.8 Висновки за розділом

У цьому розділі описано програмну реалізацію та експериментальне дослідження запропонованого методу генерації відповідей на основі підсиленого пошуку з графом походження та багатофакторним оцінюванням довіри. Архітектура системи побудована за принципом розділення обов'язків між трьома модулями – індексатором з векторним сховищем, ретрівером з графовим розширенням та генератором з оцінювачем довіри, – що забезпечує гнучкість у налаштуванні та можливість незалежної заміни окремих компонентів. Інтерфейс користувача, реалізований засобами фреймворку Streamlit, надає прозорий доступ як до підсумкової відповіді, так і до проміжних артефактів обчислення – знайдених фрагментів, шляхів їхнього походження, інтерактивної візуалізації графа та детального розкладу оцінки довіри за окремими складовими; окремі вкладки інтерфейсу підтримують як щоденне використання системи, так і експериментальний режим роботи.

Серія експериментів на сформованому тестовому корпусі з п'яти україномовних документів і наборі з десяти запитів різного типу підтвердила три ключові властивості методу. Граф походження дає

кількісне підвищення оцінки довіри саме на тих запитах, які потребують поєднання матеріалу з кількох документів – у середньому на 0.056 у порівнянні з базовим режимом без графа. На запитах, відповідь на які повністю міститься в одному документі, метод не зловживає розширенням контексту завдяки штрафному компоненту, що зберігає чесність підсумкової оцінки. У граничних випадках, коли запит виходить за межі предметної галузі корпусу, оцінка довіри коректно знижується до низького рівня, попереджаючи користувача про ненадійність відповіді і запобігаючи «впевненим галюцинаціям».

Виявлені у ході дослідження обмеження – чутливість лексичних показників до мовної невідповідності, залежність якості пошуку від обраної моделі векторних представлень та чутливість поведінки графа до порогу тематичної близькості – окреслюють природну програму подальшого вдосконалення методу, проте не впливають на правомірність основної тези роботи. Підсумково розділ підтверджує, що запропонований метод є працездатним, відтворюваним і таким, що демонструє очікувану від нього поведінку як у типових, так і у нетипових сценаріях застосування, а реалізоване програмне забезпечення може слугувати основою для подальших досліджень та практичних впроваджень у задачах побудови RAG систем із пояснюваною оцінкою довіри. Одержані результати створюють підґрунтя для масштабування підходу на більші корпуси та інші предметні галузі без зміни його базової архітектури. У ширшому контексті результати розділу демонструють практичну доцільність поєднання RAG архітектури з графом походження та оцінкою довіри як інженерного компромісу між складністю реалізації та вимогами до прозорості й надійності відповідей. Додатковою перевагою є те, що запропонований підхід узгоджується з сучасними тенденціями розвитку відповідального та прозорого штучного інтелекту, де ключову увагу приділяють саме інтерпретованості рішень і чесній індикації їхньої надійності.

ВИСНОВКИ

У межах роботи розв'язано актуальну науково-прикладну задачу підвищення достовірності та прозорості відповідей, що формуються системами генерації з підсиленим пошуком. Виконаний аналіз сучасного стану галузі засвідчив, що існуючі рішення зосереджуються переважно на якості пошуку та формулювання відповіді, тоді як питання інтерпретованої оцінки довіри та простеження походження кожного фрагмента контексту розглядаються фрагментарно. Це й визначило основну ідею роботи – поєднати підсилений пошук з графом походження та багатофакторною оцінкою довіри в єдиний цілісний метод.

У теоретичній частині сформульовано постановку задачі та запропоновано метод, у якому контекст для генерації формується двома етапами: спершу основним пошуком за векторним індексом, а потім розширенням через граф походження, побудований на основі тематичних і структурних зв'язків між фрагментами. Підсумкова оцінка довіри обчислюється як зважена сума восьми інтерпретованих складових, які охоплюють якість пошуку, узгодженість контексту із запитом, підкріплення відповіді доказами та обережність графового розширення. Така структура моделі довіри забезпечує одночасно змістовність оцінки і її прозорість для користувача, що було визначено як одну з ключових вимог до методу.

У практичній частині метод реалізовано у вигляді програмної системи з модульною архітектурою та вебінтерфейсом, після чого проведено експериментальне дослідження на спеціально сформованому тестовому корпусі та наборі запитів різного типу. Результати випробувань підтвердили основні властивості, закладені в метод: граф походження дає помітне підвищення оцінки довіри на тих запитах, які потребують поєднання матеріалу з кількох документів, метод не зловживає розширенням контексту на однодокументних запитах, а у нетипових сценаріях – зокрема при запитах поза межами корпусу – оцінка довіри коректно знижується, попереджаючи

користувача про ненадійність відповіді. При цьому оцінка довіри свідомо побудована як евристична, інтерпретована міра, яка не претендує на строгий ймовірнісний зміст, але забезпечує корисну для користувача індикацію надійності відповіді.

Виявлено також низку обмежень, які стосуються вторинних аспектів реалізації та формують природну програму подальшого вдосконалення.

Сукупно мету дипломної роботи досягнуто: розроблено, реалізовано та експериментально перевірено метод генерації відповідей з підсиленим пошуком, графом походження та багатофакторною оцінкою довіри, що демонструє очікувану поведінку як у типових, так і у граничних сценаріях застосування. Практична цінність роботи полягає в тому, що отриманий метод може використовуватись як основа для побудови інформаційно-пошукових систем у тих галузях, де поряд з якістю відповіді критичним є можливість простежити її обґрунтування та чесно оцінити рівень довіри до неї – таких як юридичні консультації, медична інформатика, наукові дослідження та академічна освіта. Перспективними напрямами продовження дослідження є перехід від лексичних до семантичних показників у модулі оцінки довіри, автоматизований підбір порогів побудови графа походження та адаптація методу до більших, різнорідних корпусів. Окремий інтерес становить інтеграція запропонованої схеми оцінювання довіри в існуючі промислові RAG-системи для підвищення їх прозорості без радикальної зміни основної архітектури. Подальший розвиток методу може також охоплювати дослідження інших типів графових структур і метрик центральності для аналізу контексту. Отримані у роботі результати створюють основу для систематичного порівняння різних стратегій побудови графа походження та їхнього впливу на якість відповідей і поведінку оцінки довіри. *У* перспективі це відкриває можливість побудови цілого класу пояснюваних RAG-систем, у яких оцінка довіри та походження відповіді є невід’ємною частиною інтерфейсу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) Retrieval-augmented generation for knowledge-intensive NLP tasks / P.Lewis et al. *arXiv.* 2020. URL: https://doi.org/10.48550/arXiv.2005.11401 (date of access: 02.05.2026).

2) Attention is all you need / A. Vaswani et al. *Advances in neural information processing systems 30*. 2017. P. 5998–6008. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (date of access: 04.05.2026).

3) BERT: pre-training of deep bidirectional transformers for language understanding / J. Devlin et al. *Proceedings of north american chapter of the association for computational linguistics: human language technologies (NAACL-HLT)*. 2017. P. 4171–4186. URL: https://doi.org/10.18653/v1/N19-1423 (date of access: 05.05.2026).

4) Improving language understanding by generative pre-training / A. Radford et al. *OpenAI technical report*. 2018. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (date of access: 06.05.2026).

5) LLaMA: open and efficient foundation language models / H. Touvron et al. *arXiv.* 2023. URL: https://doi.org/10.48550/arXiv.2302.13971 (date of access: 07.05.2026).

6) Siren's song in the AI ocean: a survey on hallucination in large language models / Y. Zhang et al. *Computational linguistics*. 2025. Vol. 51, no. 4. P. 1373–1418. URL: https://doi.org/10.1162/coli.a.16 (date of access: 07.05.2026).

7) Reimers N., Gurevych I. Sentence-BERT: sentence embeddings using siamese BERT-networks. *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)* : conference paper, Hong Kong. URL: https://doi.org/10.18653/v1/D19-1410 (date of access: 08.05.2026).

8) Johnson J., Douze M., Jégou H. Billion-scale similarity search with GPUs. *IEEE transactions on big data*. 2017. Vol. 7, no. 3. P. 535–547. URL: https://doi.org/10.1109/TBDATA.2019.2921572 (date of access: 09.05.2026).

9) Dense passage retrieval for open-domain question answering / V. Karpukhin et al. *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)* : conference paper, 2020. P. 6769–6781. URL: https://doi.org/10.18653/v1/2020.emnlp-main.550 (date of access: 10.05.2026).

10) Moreau L., Groth P. Provenance: an introduction to PROV. *Synthesis lectures on the semantic web theory and technology*. 2013. Vol. 3, no. 4. P. 1–129. URL: https://doi.org/10.2200/S00528ED1V01Y201308WBE007 (date of access: 11.05.2026).

11) Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI / A. B. Arrieta et al. *Information fusion*. 2020. Vol. 58. P. 82–115. URL: https://doi.org/10.1016/j.inffus.2019.12.012 (date of access: 11.05.2026).

12) Self-consistency improves chain of thought reasoning in language models / X. Wang et al. *Proceedings of international conference on learning representations (ICLR)*. 2022. URL: https://doi.org/10.48550/arXiv.2203.11171 (date of access: 12.05.2026).

13) On calibration of modern neural networks / C. Guo et al. *Proceedings of international conference on learning representations (ICLR)*. 2017. P. 1321–1330. URL: https://doi.org/10.48550/arXiv.1706.04599 (date of access: 13.05.2026).

14) FEVER: a large-scale dataset for fact extraction and verification / J. Thorne et al. *Proceedings of north american chapter of the association for computational linguistics: human language technologies (NAACL-HLT)*. 2018. P. 809–819. URL: https://doi.org/10.48550/arXiv.1803.05355 (date of access: 14.05.2026).

15) Perrone G., Unpingco J., Lu H.-m. Network visualizations with pyvis and visjs. *Python in science conference*, Austin, Texas, 2020. URL: https://doi.org/10.25080/Majora-342d178e-008 (date of access: 21.05.2026).

16) Streamlit docs. *Streamlit documentation.* URL: https://docs.streamlit.io/. (date of access: 07.06.2026).

17) A Streamlit-based artificial intelligence trust platform for next-generation wireless networks / M. Kuzlu et al. *arXiv.* 2020. URL: https://doi.org/10.48550/arXiv.2211.12851 (date of access: 08.06.2026).

18) Chase H. Langchain: framework for developing applications powered by language models. *GitHub.* URL: https://github.com/langchain-ai/langchain. (date of access: 11.06.2026).

19) MiniLM: deep self-attention distillation for task-agnostic compression of pre-trained transformers / W. Wang et al. *Advances in neural information processing systems*. 2020. P. 5776–5786. URL:https://doi.org/10.48550/arXiv.2002.10957. (date of access: 12.06.2026).

20) Manning C. D., Raghavan P., Schütze H. Introduction to information retrieval. *Cambridge university press*. 2008. Vol. 35, no. 2. P. 307–309. URL: https://doi.org/10.1162/coli.2009.35.2.307 (date of access: 13.06.2026).

21) Nogueira R., Cho K. Passage re-ranking with bert. *Computing research repository (CoRR)*. 2019. URL: https://doi.org/10.48550/arXiv.1901.04085 (date of access: 14.06.2026).

22) BEIR: a heterogeneous benchmark for systematic evaluation of information retrieval models / N. Thakur et al. *Conference on neural information processing systems (NeurIPS)*. 2021. URL: https://doi.org/10.48550/arXiv.2104.08663 (date of access: 15.06.2026).

23) Robertson S., Zaragoza H. The probabilistic relevance framework: bm25 and beyond. *Foundations and trends in information retrieval*. 2009. Vol. 3, no. 4. P. 333–389. URL: https://doi.org/10.1561/1500000019 (date of access: 15.06.2026).

24) Hagberg A. A., Schult D. A., Swart P. J. Exploring network structure, dynamics, and function using networkx. *Python in science conference*, Pasadena, California. 2008. P. 11–15. URL: https://doi.org/10.25080/tcwv9851 (date of access: 16.06.2026).

25) Hearst M. A. TextTiling: segmenting text into multi-paragraph subtopic passages. *Computational linguistics*. 1997. Vol. 23, no. 1. P. 33–64.

26) Efficient estimation of word representations in vector space / T. Mikolov et al. *arXiv*. 2013. URL: https://doi.org/10.48550/arXiv.1301.3781 (date of access: 17.06.2026).

27) Cormen T. H., Stein C., Rivest R. L. Introduction to algorithms, fourth edition. *MIT press*. 2022. 1332 p.