МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

ШЕВЧУК Віталій Олександрович

Інтелектуальна інформаційна система семантичного пошуку з використанням технології Retrieval-Augmented Generation (RAG) / Intelligent Information System For Semantic Search Using Retrieval-Augmented Generation (RAG) Technology

Спеціальність 122 «Комп'ютерні науки»
освітньо-професійна програма – Штучний інтелект
Кваліфікаційна робота

Виконав студент групи КНШІ-41
В.О. Шевчук
_____

Науковий керівник
к.т.н., доцент Т.В. Лендюк
_____

Кваліфікаційну роботу
допущено до захисту
«___»_____20___р.

Завідувач кафедри
_____Н.В. Дзюбановська

ТЕРНОПІЛЬ – 2026

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність: 122 «Комп'ютерні науки»
освітньо-професійна програма – Штучний інтелект

«ЗАТВЕРДЖУЮ»
В.о. завідувача кафедри
_____ Н.В. Дзюбановська

«____» _____20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Шевчуку Віталію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Інтелектуальна інформаційна система семантичного пошуку з використанням технології Retrieval-Augmented Generation (RAG) / Intelligent Information System For Semantic Search Using Retrieval-Augmented Generation (RAG) Technology

керівник роботи к.т.н., доцент Т. В. Лендюк
затверджений наказом по університету від 01 грудня 2025 року № 894.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:
- проаналізувати предметну область семантичного пошуку у корпоративних базах знань;
- виконати порівняльний аналіз векторних баз даних;
- сформулювати постановку задачі, функціональні та нефункціональні вимоги до програмного засобу;
- спроєктувати архітектуру системи за принципом багаторівневого поділу на шість рівнів;
- розробити алгоритмічне забезпечення системи;
- реалізувати програмний засіб мовою Python;
- провести експериментальне дослідження якості системи;
- провести тестування програмного засобу.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
- структурна схема програмного засобу семантичного пошуку;
- UML-діаграма класів основних компонентів системи;
- UML-діаграма послідовності обслуговування запиту користувача.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Огляд літературних джерел відповідно до предметної області та складання плану роботи	до 01.01.2026 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02.2026 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 15.03.2026 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05.2026 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2026 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2026 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2026 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2026 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06.2026 р.	

Студент _____ *Шевчук В.О.*
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи _____ *Лендюк Т. В.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи: 71 с., 21 рис., 12 табл., 2 додатки, 44 джерела.

Дослідження зосереджене на семантичному пошуку інформації в корпоративній базі знань технічної документації, де знайдені фрагменти стають основою для розгорнутих відповідей природною мовою.

У роботі представлено інтелектуальну інформаційну систему семантичного пошуку на основі технології Retrieval-Augmented Generation. Система індексує 8 654 фрагменти технічної документації та відповідає на запитання природною мовою з посиланнями на першоджерела, а не у формі вільної генерації.

Під час індексування кожен документ розбивається на фрагменти по 512 токенів із перекриттям 64 токени за рекурсивною ієрархією роздільників. Щільні векторні представлення формує модель multilingual-e5-base (768 вимірів), а розріджену гілку забезпечує BM25. По 20 найкращих кандидатів від кожної гілки об'єднуються методом Reciprocal Rank Fusion з k = 60. Крос-енкодер bge-reranker-base перевпорядковує топ-20 до топ-5, які передаються до моделі gpt-4o-mini або Llama-3.1-8B для формування підсумкової відповіді. Якість оцінюють метриками Recall@k, MRR і NDCG на рівні пошуку та метриками faithfulness і answer relevancy з бібліотеки ragas для всього конвеєра.

На 8 654 фрагментах документації та 80 оцінювальних запитах система досягає Recall@5 = 87,6 %, MRR = 80,3 %, NDCG@10 = 84,1 % і faithfulness 92,7 %. Медіанний час відгуку становить 1,84 с у хмарному режимі (OpenAI API) і 1,32 с під час локального запуску на Ollama з моделлю Llama-3.1-8B-Instruct та GPU.

Систему доцільно застосовувати в IT-компаніях, сервісних організаціях та освітніх установах, що створюють корпоративні довідкові служби, асистентів технічної підтримки чи навчальні помічники для окремих курсів, – усюди, де запити стосуються змісту, а не лише збігу ключових слів.

Ключові слова: СЕМАНТИЧНИЙ ПОШУК, RETRIEVAL-AUGMENTED GENERATION, ВЕЛИКІ МОВНІ МОДЕЛІ, ВЕКТОРНІ ПРЕДСТАВЛЕННЯ, ГІБРИДНИЙ ПОШУК, BM25, HNSW, ПЕРЕРЕЙТИНГ, КОРПОРАТИВНА БАЗА ЗНАНЬ, ОБРОБКА ПРИРОДНОЇ МОВИ.

ANNOTATION

Explanatory note to the qualification work: 71 p., 21 fig., 12 tab., 2 appendices, 44 references.

The research focuses on semantic information retrieval inside a corporate knowledge base of technical documentation, where the matched fragments become the basis for comprehensive natural-language responses.

The thesis presents an intelligent information system for semantic search built on Retrieval-Augmented Generation. It indexes 8,654 fragments of technical documentation and answers questions in natural language with citations to the source fragments rather than free-form generation.

Indexing splits each document into 512-token fragments with 64-token overlap using a recursive separator hierarchy. Dense embeddings come from multilingual-e5-base (768 dimensions); the sparse path runs BM25. Top-20 candidates from each path go through Reciprocal Rank Fusion with k=60. A cross-encoder bge-reranker-base reorders the top-20 down to top-5, which feed into gpt-4o-mini or Llama-3.1-8B for the final answer. Quality is measured by Recall@k, MRR and NDCG on the retrieval side, and by faithfulness and answer relevancy from the ragas library on the full pipeline.

On 8,654 documentation fragments and 80 evaluation queries the system reaches Recall@5 = 87.6 %, MRR = 80.3 %, NDCG@10 = 84.1 % and faithfulness 92.7 %. Median response time is 1.84 s in cloud mode (OpenAI API) and 1.32 s when running locally on Ollama with Llama-3.1-8B-Instruct and a GPU.

The system fits IT companies, service organizations and educational institutions building corporate help-desks, technical-support copilots or per-course study aids – anywhere queries care about meaning, not just keyword overlap.

Keywords – semantic search, retrieval-augmented generation, large language models, vector embeddings, hybrid retrieval, BM25, HNSW, reranking, corporate knowledge base, NLP.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Позначення	Розшифрування
ШІ	штучний інтелект
ОПМ	обробка природної мови (Natural Language Processing)
МН	машинне навчання (Machine Learning)
ГН	глибоке навчання (Deep Learning)
ВММ	велика мовна модель (Large Language Model)
ПЗ	програмне забезпечення
БД	база даних
СУБД	система управління базами даних
API	програмний інтерфейс застосунку (Application Programming Interface)
REST	архітектурний стиль розподілених систем (Representational State Transfer)
HTTP	протокол передавання гіпертексту (HyperText Transfer Protocol)
JSON	текстовий формат обміну даними (JavaScript Object Notation)
UML	уніфікована мова моделювання (Unified Modeling Language)
SQL	мова структурованих запитів (Structured Query Language)
TF-IDF	міра ваги терміна (Term Frequency – Inverse Document Frequency)
BM25	функція релевантності пошуку Best Matching 25
BERT	двонапрямлені трансформери для представлення (Bidirectional Encoder Representations from Transformers)
RAG	генерація з доповненням пошуком (Retrieval-Augmented Generation)
LLM	велика мовна модель (Large Language Model)
NLP	обробка природної мови (Natural Language Processing)
IR	інформаційний пошук (Information Retrieval)
ANN	наближений пошук найближчих сусідів (Approximate Nearest Neighbor)
kNN	метод k найближчих сусідів (k-Nearest Neighbors)
HNSW	ієрархічно орієнтований малий світ (Hierarchical Navigable Small World)
IVF	індекс на основі інвертованих файлів (Inverted File Index)
PQ	квантування за добутком (Product Quantization)
FAISS	бібліотека індексів векторного пошуку (Facebook AI Similarity Search)
SBERT	сіамські трансформери для речень (Sentence-BERT)
MRR	середнє обернене ранжування (Mean Reciprocal Rank)
NDCG	нормалізована знижена кумулятивна вигода (Normalized Discounted Cumulative Gain)
MAP	усереднена середня точність (Mean Average Precision)

OOV	слово поза словником (Out-of-Vocabulary)
ETL	видобування, перетворення, завантаження (Extract, Transform, Load)
GPU	графічний процесор (Graphics Processing Unit)
ORM	об'єктно-реляційне відображення (Object-Relational Mapping)
YAML	формат серіалізації даних (YAML Ain't Markup Language)
CRUD	операції створення, читання, оновлення, видалення (Create, Read, Update, Delete)

ВСТУП

Актуальність теми. Сучасні підприємства накопичують значні обсяги неструктурованої текстової інформації: технічна документація, регламенти, інструкції з налаштування програмних продуктів, бази знань служби підтримки, протоколи нарад, звіти. За оцінками галузевих аналітиків, частка неструктурованих даних у загальному інформаційному фонді організацій сягає 80 %, а їхній обсяг подвоюється кожні два-три роки. Відсутність зручних засобів пошуку зумовлює те, що співробітники витрачають на пошук потрібної інформації від 1,5 до 2,5 годин робочого часу щодня, а до 30 % робочих звернень виявляються повторними через те, що відповідь уже існує, але не може бути знайдена.

Класичний пошук за ключовими словами – Boolean, TF-IDF, BM25 – погано впораються із синонімією. Запит «зміна налаштувань мережі» не дасть результатів, якщо в документі вжито «модифікація конфігурації з'єднання». До того ж класичні системи повертають перелік посилань, а не пряму відповідь, тож користувачеві доводиться самому шукати потрібне у знайдених документах. Великі мовні моделі (LLM) пишуть зв'язну відповідь природною мовою, але їхні тренувальні дані застарівають уже через рік-два, до приватних корпоративних документів у них доступу немає, а «галюцинації» – правдоподібні, але вигадані твердження – змушують перевіряти кожну відповідь вручну.

Технологію генерації з доповненням пошуком (Retrieval-Augmented Generation, RAG) описали П. Льюїс і співавтори у статті 2020 року. Ідея проста – перед відповіддю мовна модель отримує контекст із зовнішнього сховища знань. Семантичний пошук у векторному просторі знаходить релевантні фрагменти, а модель пише відповідь, спираючись виключно на них і додаючи посилання на джерела. Сьогодні на RAG будують корпоративні чат-боти, асистенти технічної підтримки та внутрішні пошукові системи; звіти Gartner і McKinsey зараховують підхід до стратегічних технологій штучного інтелекту найближчого десятиліття.

Попри інтенсивний розвиток окремих компонентів RAG-систем (моделей векторних представлень, алгоритмів наближеного пошуку найближчих сусідів,

великих мовних моделей), задача побудови комплексного програмного засобу семантичного пошуку для корпоративної бази знань залишається відкритою. Зокрема, не сформовано загальноприйнятих рекомендацій щодо вибору стратегії розбиття документів на фрагменти, поєднання щільного й розрідженого пошуку, формування контекстного вікна для мовної моделі та оцінювання якості відповідей. Тому розроблення інтелектуальної інформаційної системи семантичного пошуку на основі технології RAG є актуальною науково-практичною задачею.

Мета роботи – розробити інтелектуальну інформаційну систему семантичного пошуку, побудовану на технології Retrieval-Augmented Generation і здатну формувати точну, контекстно обґрунтовану відповідь природною мовою з підтвердженням джерелами.

Для досягнення поставленої мети в роботі сформульовано такі завдання:

– проаналіювати предметну область семантичного пошуку у корпоративних базах знань;

– виконати порівняльний аналіз векторних баз даних та фреймворків оркестрації RAG-конвеєрів;

– сформулювати постановку задачі, функціональні та нефункціональні вимоги до програмного засобу;

– спроєктувати архітектуру системи за принципом багаторівневого поділу на шість рівнів;

– розробити алгоритмічне забезпечення системи;

– реалізувати програмний засіб мовою Python;

– провести експериментальне дослідження якості системи.

– провести трирівневе тестування програмного засобу.

Дослідження зосереджене на процесі семантичного пошуку інформації у корпоративній базі знань технічної документації, де знайдені фрагменти стають основою для розгорнутих відповідей природною мовою.

Об'єкт дослідження – процес семантичного пошуку інформації у корпоративній базі знань технічної документації з формуванням розгорнутих відповідей природною мовою.

Предмет дослідження – методи та програмні засоби побудови інтелектуальних інформаційних систем семантичного пошуку на основі поєднання щільних векторних представлень, наближеного пошуку найближчих сусідів та великих мовних моделей за технологією Retrieval-Augmented Generation.

Методи дослідження. Для розв'язання поставлених завдань застосовано: методи обробки природної мови (токенізація, нормалізація, виділення речень) – для попереднього опрацювання текстів технічної документації; методи побудови щільних векторних представлень на основі трансформерних моделей (Sentence-BERT, моделі сімейства E5) – для семантичного подання текстових фрагментів; алгоритми наближеного пошуку найближчих сусідів у багатовимірному просторі (HNSW, IVF-PQ) – для ефективного добору релевантних фрагментів; статистична функція ранжування BM25 – для розрідженої компоненти гібридного пошуку; методи перерейтингу на основі крос-енкодерів – для уточнення порядку результатів; методи об'єктно-орієнтованого проектування та архітектурного моделювання – для розроблення структури системи; метрики інформаційного пошуку (Recall@k, MRR, NDCG) та метрики якості RAG-систем (faithfulness, answer relevancy) – для експериментального оцінювання якості рішення.

Практичне значення одержаних результатів полягає в тому, що розроблений програмний засіб дає змогу скоротити час пошуку інформації співробітниками за рахунок розуміння системою сенсу запиту, а не лише ключових слів; зменшити навантаження на службу підтримки шляхом автоматизованих відповідей із посиланням на першоджерело; забезпечити єдину точку доступу до розподілених джерел технічної документації; знизити ризик застосування недостовірних відомостей завдяки прив'язці кожної відповіді до конкретних фрагментів документів. Запропоновані рішення можуть бути використані IT-підприємствами, сервісними організаціями, освітніми установами та іншими організаціями, що працюють зі значним обсягом внутрішньої документації.

Технічна та програмна платформа. Програмний засіб реалізовано мовою програмування Python із застосуванням бібліотек семантичного пошуку sentence-transformers і rank_bm25, фреймворку оркестрації RAG-конвеєрів LangChain, векторної бази даних Chroma, великої мовної моделі сімейства GPT,

вебфреймворку FastAPI та засобу побудови інтерфейсів Streamlit. Експериментальне дослідження якості пошуку проведено за допомогою бібліотеки ragas.

Апробація результатів роботи. Основні результати роботи висвітлено в тезах доповіді, поданих до участі в науково-практичній конференції за тематикою інтелектуальних інформаційних систем та обробки природної мови.

Пояснювальна записка складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків.

1 СТАН ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Семантичний пошук у корпоративних базах знань

Корпоративна база знань – це впорядкована сукупність документів, що накопичує знання організації у формі, придатній для зберігання, пошуку та повторного використання. До типового наповнення такої бази належать інструкції з налаштування програмних продуктів, регламенти технічного супроводу, проєктна документація, протоколи інцидентів служби підтримки, методичні матеріали та внутрішні звіти. Особливістю корпоративної інформації є те, що понад 80 % її обсягу становлять неструктуровані текстові документи, тоді як на структуровані відомості у реляційних базах припадає не більше 20 % загального обсягу [1].

Спеціалісти аналітичної компанії IDC оцінюють середній час, який працівники сучасних організацій витрачають на пошук інформації, у 1,8 години на день, або близько 9 годин на тиждень. У великих компаніях до 30% звернень до служб технічної підтримки виявляються повторними: відповідь уже існує у внутрішніх документах, проте не може бути знайдена через брак ефективних засобів пошуку [2]. Це призводить до прямих фінансових втрат, дублювання роботи різних відділів та відтоку інституційних знань разом зі звільненням співробітників.

Задачу пошуку інформації у великих текстових колекціях традиційно розв'язує науково-прикладна галузь інформаційного пошуку (Information Retrieval, IR). Класична модель охоплює два процеси: індексування формує внутрішнє представлення документів, придатне для швидкого добору за запитом, а власне пошук добирає документи, релевантні запиту користувача, з ранжуванням за оцінкою релевантності.

За способом представлення документів і запитів виокремлюють три покоління моделей пошуку, узагальнені на рисунку 1.1.

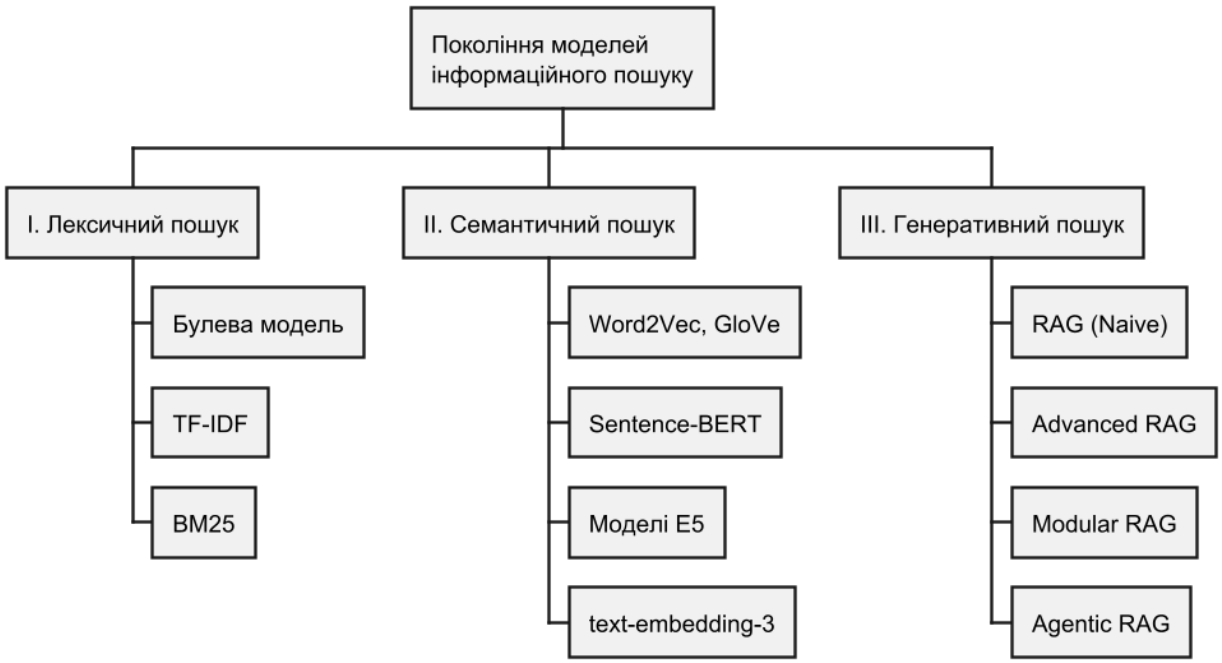

Рисунок 1.1 – Покоління моделей інформаційного пошуку

Лексичний пошук (перше покоління) оперує поверхневим зіставленням ключових слів запиту і документа; найвідомішими представниками є булева модель, векторна модель з зважуванням TF-IDF та статистична функція ранжування BM25 (Best Matching 25). Такі моделі прості у реалізації, інтерпретовані й обчислювально ефективні, але мають принципові обмеження. Синонімія руйнує пошук на рівні формулювань – запит «зміна налаштувань мережі» не дасть результатів, якщо в документі вжито вислів «модифікація конфігурації з'єднання». Полісемія розмиває значення – слово «миша» однаково описує і гризуна, і пристрій введення. Запит у формі питання «як скинути пароль?» опрацьовується так само, як набір окремих ключових слів, без розуміння намірів користувача. Нарешті, лексичні моделі нечутливі до порядку та зв'язків між словами, тож «банк віддає кредит» і «банк забирає кредит» оцінюються майже однаково.

Семантичний пошук (друге покоління) ґрунтується на щільних векторних представленнях (dense embeddings), що ставлять у відповідність кожному фрагментові тексту вектор у багатовимірному дійсному просторі таким чином, щоб семантично близькі фрагменти отримували близькі вектори. Семантичне подання здобувають за допомогою нейромережевих моделей, навчених на великих

текстових корпусах. Першими широко вживаними моделями стали Word2Vec [3] і GloVe, які подавали окремі слова; згодом їх замінили моделі рівня речення на основі трансформерної архітектури – BERT [4], Sentence-BERT [5], сімейство моделей E5 та text-embedding-3 від OpenAI. Семантичний пошук розв'язує проблеми синонімії, полісемії та поверхневого зіставлення, проте сам по собі повертає користувачеві лише перелік фрагментів, не формулюючи відповіді на запит.

Генеративний пошук (третє покоління) доповнює семантичний пошук генеративною мовною моделлю, яка узагальнює знайдені фрагменти та формує природномовну відповідь. Саме до цього покоління належить технологія Retrieval-Augmented Generation, що розглядається у роботі.

Корпоративний пошук відрізняється від загального вебпошуку за п'ятьма ключовими особливостями, що безпосередньо визначають вимоги до програмного засобу. По-перше, він працює в обмеженому предметному просторі – замкненій колекції документів конкретної організації з власним термінологічним апаратом. По-друге, до достовірності висуваються підвищені вимоги: відповідь має посилатися на конкретний фрагмент документа, який користувач може перевірити. По-третє, приватність забороняє передавати корпоративну інформацію до публічних сервісів без додаткових заходів захисту. По-четверте, джерела гетерогенні – документи зберігаються у різних форматах (PDF, DOCX, HTML, Markdown, Confluence, SharePoint, Notion) і потребують уніфікованого опрацювання. По-п'яте, вміст динамічний: документація постійно оновлюється, що зумовлює потребу інкрементного оновлення індексу.

Узагальнену схему процесу подано на рисунку 1.2.

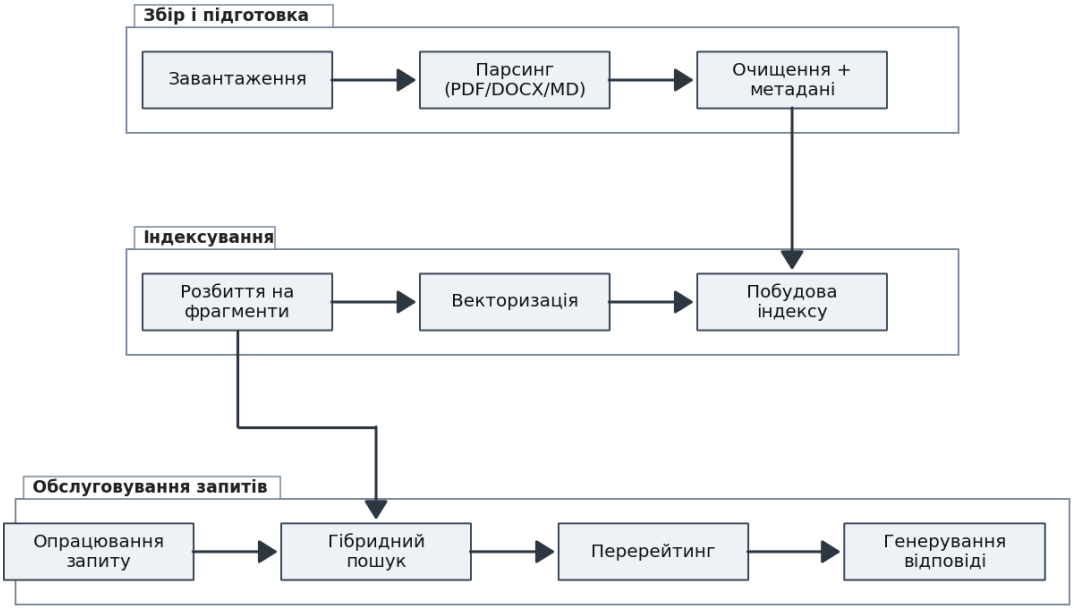

Рисунок 1.2 – Бізнес-процес пошуку інформації в корпоративній базі знань

Процес охоплює три основні фази: збір і підготовка документів із розрізнених джерел; формування індексу, що забезпечує швидкий пошук; та обслуговування запитів користувачів. Кожна фаза породжує специфічні задачі: завантаження та парсинг файлів різних форматів, очищення тексту та виділення метаданих, розбиття на смислові фрагменти, обчислення векторних представлень, побудова індексу для наближеного пошуку, опрацювання запиту, добір кандидатів, перерейтинг, формування контексту для мовної моделі та генерування відповіді.

Перелічені особливості та задачі утворюють предметну область інтелектуальних інформаційних систем семантичного пошуку, у межах якої виконується ця кваліфікаційна робота.

1.2 Технологія Retrieval-Augmented Generation як основа інтелектуального семантичного пошуку

Технологія Retrieval-Augmented Generation, або генерація з доповненням пошуком, є архітектурним шаблоном побудови інтелектуальних систем, що

поєднує два процеси: пошук релевантної інформації в зовнішньому джерелі знань та її використання великою мовною моделлю для формування відповіді. Технологію було запропоновано П. Льюїсом та співавторами у дослідницькому центрі Facebook AI Research у 2020 році [6] як рішення для задач відкритого запитання-відповіді (open-domain question answering).

Появу RAG спричинили принципові обмеження великих мовних моделей. Сучасні моделі (GPT-4, Claude 3.5, Llama 3, Gemini 1.5) володіють широкою параметричною пам'яттю, проте їхнє знання обмежене моментом завершення тренування, не охоплює приватної доменної інформації та схильне до галюцинацій – формування правдоподібних, але недостовірних тверджень [7]. Доповнення моделі зовнішнім сховищем знань, з якого релевантний контекст добирається безпосередньо під час обслуговування запиту, дає змогу долати ці обмеження без перетренування моделі.

Узагальнену схему подано на рисунку 1.3.

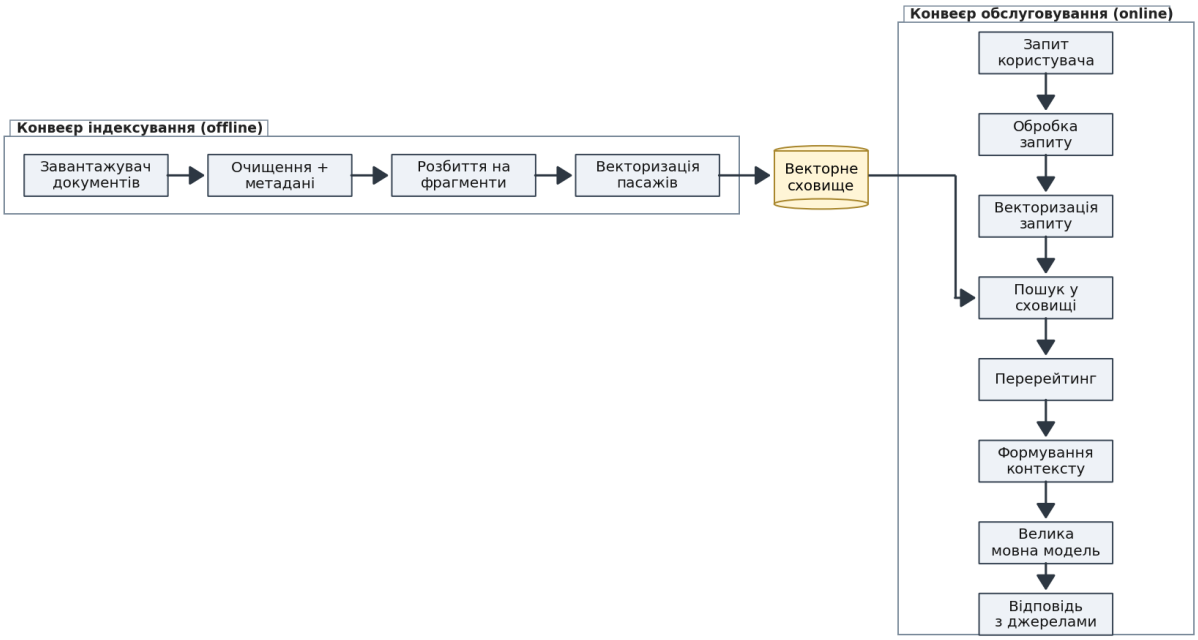

Рисунок 1.3 – Узагальнена архітектура системи Retrieval-Augmented Generation

Архітектура поділяється на два самостійні конвеєри.

Конвеєр індексування (offline) виконується одноразово під час підготовки системи та оновлюється у міру зміни корпусу. Він починається із завантаження

документів з первинних джерел (файлова система, корпоративний портал, система контролю версій), далі іде очищення тексту – вилучення розмітки, нормалізація пробілів, об'єднання багаторядкових заголовків – і виділення метаданих (назва, автор, дата, розділ, тип). Тексти розбиваються на смислові фрагменти (chunking) розміром зазвичай 200–1000 токенів, для кожного фрагмента обчислюються векторні представлення заздалегідь натренованою моделлю, а самі фрагменти разом із векторами зберігаються у векторній базі даних із побудовою індексу для наближеного пошуку.

Конвеєр обслуговування запиту (online) виконується для кожного запиту користувача. Спершу запит проходить попередню обробку – нормалізацію та за потреби трансформацію (наприклад, переформулювання мовною моделлю для покращення пошуку), – а потім перетворюється на вектор тією самою моделлю, що й під час індексування. У векторному просторі відбувається пошук релевантних фрагментів – добір топ-k кандидатів за косинусною подібністю або скалярним добутком, – після чого крос-енкодер уточнює їх порядок (re-ranking), оцінюючи подібність пари «запит – кандидат» точніше за біенкодер. Відібрані фрагменти зливаються у єдиний текстовий блок із метаданими про джерела й передаються великій мовній моделі разом із підказкою, що зобов'язує відповідати лише на основі наданого контексту. Завершується конвеєр постобробкою – додаванням посилань на джерела, валідацією відповіді та журналюванням діалогу.

Класифікацію за рівнем складності подано на рисунку 1.4.

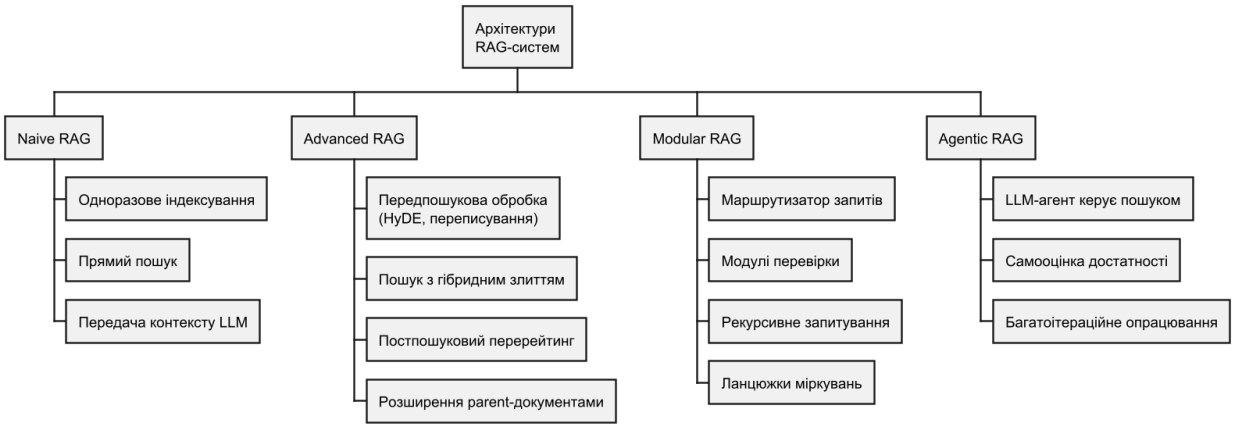

Рисунок 1.4 – Класифікація архітектур RAG-систем

Розрізняють чотири основні різновиди RAG-архітектур.

Naive RAG – найпростіша реалізація: одноразове індексування, одноразовий пошук, пряма передача відібраних фрагментів мовній моделі. Підходить для дослідницьких прототипів та невеликих колекцій документів, проте має очевидні слабкі місця: чутливість до якості розбиття на фрагменти, відсутність контролю достовірності відповіді, проблеми зі складними багатоетапними запитами.

Advanced RAG доповнює базову схему попередньою та постпошуковою обробкою: трансформацією запиту (HyDE, переписування запиту мовною моделлю, генерування підзапитів), розширенням контексту суміжними фрагментами (parent document retrieval), перерейтингом за допомогою крос-енкодерів, дедуплікацією та об'єднанням результатів від кількох пошуковців.

Modular RAG реалізує RAG як набір незалежних модулів, які можна вільно комбінувати у конвеєр під конкретну задачу: маршрутизатор запитів за типом, модуль міркувань ланцюжком думок, модуль перевірки достовірності, модуль рекурсивного запитування. Modular RAG є стандартом промислових впроваджень.

Agentic RAG надає мовній моделі активну роль: модель самостійно вирішує, чи потрібно виконувати пошук, формулює запити до пошуковця, оцінює достатність зібраної інформації та за потреби виконує додаткові ітерації. Цей різновид поєднує RAG з парадигмою інтелектуальних агентів і дає найкращі результати для складних запитів, що потребують міркувань і поєднання інформації з кількох джерел.

Розглянемо ключові техніки, що використовуються у сучасних RAG-системах.

Стратегії розбиття на фрагменти. Розбиття довгого документа на фрагменти прийнятного розміру є нетривіальним: занадто короткі фрагменти втрачають контекст, занадто довгі знижують точність векторного представлення. Найпростіша стратегія – розбиття на блоки фіксованого розміру з перекриттям; вона швидка, але розриває смислові межі. Розбиття за реченнями та абзацами зберігає природні смислові межі, а рекурсивне розбиття з ієрархією роздільників комбінує абзаци та речення з контролем максимального розміру фрагмента. Семантичне розбиття об'єднує сусідні речення за подібністю векторних

представлень, а структурне використовує заголовки та логічні розділи документа – зручне для Markdown, HTML і LaTeX.редставлення та переповнюють контекстне вікно мовної моделі. Найпоширеніші стратегії:

Гібридний пошук об'єднує щільний (семантичний) і розріджений (BM25) пошук для досягнення переваг обох. Розріджений пошук точніше зіставляє рідкісні терміни (назви продуктів, абревіатури, ідентифікатори), щільний – узагальнює перифрази та запити природною мовою. Об'єднання результатів виконують методом Reciprocal Rank Fusion (RRF).

Перерейтинг крос-енкодерами. Перший етап пошуку добирає, як правило, 20–50 кандидатів за біенкодером (швидкий, але менш точний); другий етап перевпорядковує їх крос-енкодером (повільний, але точний), який спільно опрацьовує запит і кандидата та видає точнішу оцінку релевантності.

Інженерія підказок (prompt engineering). Конструкція підказки безпосередньо впливає на якість відповіді. Стандартними елементами є системна роль («ви – асистент, що відповідає на основі наданого контексту»), інструкції щодо формату та стилю відповіді, явні обмеження («якщо відповідь не міститься у контексті, скажіть про це»), вимога цитування фрагментів.

Оцінювання якості RAG-систем. На відміну від класичного пошуку, оцінювання RAG ускладнюється тим, що відповіддю є природномовний текст, а не перелік документів. Сучасні методики поділяють оцінювання на пошукову складову (Recall@k, MRR, NDCG) та генеративну (faithfulness – відповідність генерованого тексту контексту, answer relevancy – відповідність запиту користувача, context precision/recall – якість відібраного контексту). Розраховувати ці метрики автоматично дозволяють бібліотеки на кшталт ragas [8].

1.3 Огляд та аналіз існуючих рішень

Сучасний ринок інструментів для побудови RAG-систем швидко розвивається та представлений значною кількістю рішень. Аналіз існуючих засобів

проведено у трьох категоріях: векторні бази даних, фреймворки оркестрації RAG-конвеєрів та готові комерційні платформи семантичного пошуку.

Векторні бази даних є базовою інфраструктурою для зберігання щільних векторних представлень та виконання наближеного пошуку найближчих сусідів. Порівняння провідних рішень подано в таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика векторних баз даних

Характеристика	FAISS	Chroma	Qdrant	Weaviate	Pinecone
Тип рішення	Бібліотека	Вбудована БД	Сервер	Сервер	Хмарний сервіс
Мова реалізації	C++	Python	Rust	Go	Закрита
Алгоритми ANN	HNSW, IVF, PQ	HNSW	HNSW	HNSW	Закритий
Гібридний пошук	Ні	Так (через додатки)	Так (BM25)	Так (BM25)	Так
Фільтрація за метаданими	Обмежена	Так	Так (повна)	Так	Так
Локальне розгортання	Так	Так	Так	Так	Ні
Реплікація і шардинг	Ні	Ні	Так	Так	Так
Ліцензія	MIT	Apache 2.0	Apache 2.0	BSD 3-Clause	Закрита, платна

FAISS (Facebook AI Similarity Search) – бібліотека, розроблена компанією Meta, що пропонує найповніший набір алгоритмів наближеного пошуку (HNSW, IVF, IVF-PQ, IVF-OPQ) із підтримкою прискорення на графічному процесорі [9]. Не є базою даних у класичному розумінні: не зберігає метадані, не має серверного інтерфейсу, не підтримує оновлень окремих векторів. Найдоцільніше застосовувати у складі більших систем як суто алгоритмічний компонент.

Chroma – вбудована векторна база даних на Python, що поєднує простоту локального розгортання з достатнім функціональним покриттям: збереження векторів, метаданих, документів, фільтрація, перевірка унікальності. Орієнтована на прототипування та системи малого та середнього масштабу. Має офіційну інтеграцію з LangChain і LlamaIndex.

Qdrant – самостійний векторний сервер на Rust із гнучкою системою фільтрації за метаданими, гібридним пошуком, реплікацією, шардингом та

механізмом снапшотів. Доступний як для самостійного розгортання, так і у вигляді хмарного сервісу. Виокремлюється високою продуктивністю та підтримкою складних запитів типу «знайти векторно схоже з фільтром за метаданими».

Weaviate – векторна база на Go з нативною підтримкою графа знань, вбудованими модулями векторизації, гібридним пошуком та GraphQL-інтерфейсом. Орієнтована на корпоративні впровадження.

Pinecone – повністю керований хмарний сервіс із простим API, але без можливості локального розгортання. Платна модель тарифікації за обсягом збережених векторів та кількістю запитів.

З огляду на вимоги до приватності корпоративних даних, можливості локального розгортання та достатньої функціональної повноти для прототипу базою для практичної реалізації обрано Chroma з можливістю міграції на Qdrant у промисловому впровадженні.

Фреймворки оркестрації RAG-конвеєрів надають готові абстракції для побудови повного конвеєра: завантаження документів, розбиття, векторизація, пошук, формування підказки, виклик мовної моделі. Зведення подано в таблиці 1.2.

Таблиця 1.2 – Порівняльна характеристика фреймворків оркестрації RAG-конвеєрів

Характеристика	LangChain	LlamaIndex	Haystack
Розробник	LangChain Inc.	LlamaIndex Inc.	deepset
Основна мова	Python, JS	Python	Python
Концепція	Ланцюги та агенти	Індекси та запитувачі	Конвеєри з вузлами
Кількість інтеграцій	Дуже велика (>500)	Велика (>300)	Середня (>100)
Орієнтація	Загальні LLM-застосунки	RAG над даними	Промислові пошукові системи
Документація	Велика, мінлива	Структурована	Якісна
Зрілість API	У стадії стабілізації	Стабільне	Стабільне

LangChain – найпопулярніший фреймворк для побудови застосунків на основі великих мовних моделей. Пропонує абстракції завантажувачів документів, текстових розбивачів, векторних сховищ, мовних моделей, ланцюгів і агентів. Має найширшу екосистему інтеграцій. Слабким місцем є мінливість публічного API між версіями та надмірна абстрагованість у складних випадках.

LlamaIndex (раніше GPT Index) спеціалізується саме на RAG. Пропонує розвинений набір індексів (векторний, дерево, перелік, графовий), стратегій розбиття та запитувачів. Реалізація розширених технік (HyDE, sub-question query engine, recursive retrieval) є зрілою.

Haystack від deepset історично орієнтований на промислові пошукові системи. Концепція конвеєрів із явним графом вузлів забезпечує контрольованість і вимірність. Підтримує продакшн-розгортання, моніторинг, дистрибутивне виконання.

Для практичної реалізації обрано LangChain як основний фреймворк через широту інтеграцій та зрілість документації для основних сценаріїв RAG.

Готові комерційні платформи семантичного пошуку. На ринку існують готові SaaS-платформи (Glean, Mendable, Vectara, Sana AI, Notion AI), що пропонують RAG як послугу. Їх перевагою є мінімізація зусиль на розгортання, недоліком – повна залежність від постачальника, обмеженість налаштування, передача корпоративних даних до зовнішнього сервісу та висока вартість використання. Розроблення власного програмного засобу залишається доцільним для організацій, що мають вимоги до конфіденційності, специфіку предметної області або потребу глибокої інтеграції з внутрішніми системами.

Виявлені обмеження існуючих рішень. Аналіз векторних баз даних, фреймворків і платформ показав кілька обмежень, що залишають простір для розроблення власного програмного засобу. Більшість готових SaaS-платформ не підтримує локального розгортання, що суперечить вимогам приватності корпоративних даних. Фреймворки оркестрації надають лише загальні абстракції, тоді як побудова якісної системи потребує самостійного вибору стратегії розбиття, моделі векторизації, способу гібридного пошуку та формату відповіді. Стандартні приклади з документації фреймворків реалізують наївний RAG, який не забезпечує прийнятної якості на корпоративних даних. Проблема оцінювання якості семантичного пошуку залишається складною – автоматичне порівняння природномовних відповідей з еталоном неоднозначне, а ручне оцінювання трудомістке. Нарешті, готові моделі векторизації натреновані переважно на

англомовних корпусах, що знижує якість на українській та інших неанглійських мовах і змушує обґрунтовано добирати мультимовну модель.бу:

1.4 Постановка задачі

За результатами аналізу предметної області, технології RAG та існуючих рішень сформульовано постановку задачі кваліфікаційної роботи.

Формальне формулювання задачі. Нехай задано колекцію документів $D = \{d_1, d_2, \dots, d_N\}$ корпоративної бази знань, де кожен документ d_i – це текст довільної довжини з метаданими (назва, автор, дата, тип, шлях). Нехай також задано запит користувача q природною мовою. Потрібно розробити функцію $f: q \to a$, що формує природномовну відповідь a, спираючись на релевантні фрагменти документів колекції D, та супроводжує її переліком джерел, на основі яких побудовано відповідь.

Якість функції оцінюватиметься на пошуковому рівні через Recall@k (частка релевантних фрагментів у топ-k результатах пошуку), MRR (середнє обернене значення позиції першого релевантного фрагмента в ранжованому переліку) і NDCG (нормалізоване кумулятивне значення оцінок релевантності з урахуванням позиції). На рівні всієї відповіді додаються faithfulness – частка тверджень, що підтверджуються відібраним контекстом, відповідність природномовної відповіді запиту користувача (answer relevancy), а також повний час відгуку від отримання запиту до видачі результату.

Функціональні вимоги до програмного засобу охоплюють імпорт документів з PDF, DOCX, Markdown, HTML і TXT з автоматичним виділенням метаданих, розбиття на смислові фрагменти із заданими параметрами розміру та перекриття, обчислення щільних векторних представлень заздалегідь натренованою моделлю та зберігання фрагментів, метаданих і векторів у векторній базі даних з можливістю інкрементного оновлення. На рівні запиту засіб має виконувати гібридний пошук (щільний разом із розрідженим) природною мовою з фільтрацією за метаданими, перерейтингувати результати крос-енкодерною моделлю, формувати контекст для

великої мовної моделі з оптимізацією під розмір контекстного вікна та генерувати природномовну відповідь з посиланнями на конкретні джерела. Окремо передбачено вебінтерфейс для введення запиту, перегляду відповіді, аналізу обраних фрагментів і оцінювання якості, а також журналювання запитів і відповідей для подальшого аналізу та покращення системи.

Нефункціональні вимоги охоплюють продуктивність – час відгуку на запит не повинен перевищувати 5 секунд для колекції 10 000 фрагментів – і масштабованість архітектури до 1 млн фрагментів без принципових змін. Приватність вимагає підтримки повністю локального розгортання без передавання даних до зовнішніх сервісів, а модульність – взаємозамінних компонентів індексування, пошуку, перерейтингу та генерування. Спостережуваність забезпечується реєстрацією ключових метрик кожного запиту (час, кількість відібраних фрагментів, використана модель), переносність – запуском на Linux, Windows і macOS без модифікацій, а простота розгортання – повним запуском системи однією командою з підготовленим оточенням.

Обмеження. З огляду на навчальний характер роботи прийнято такі обмеження: колекція документів обмежується текстами однієї мови (англійською або українською); підтримка мультимодальних запитів (зображення, аудіо) не реалізовується; інтеграція з корпоративними системами єдиного входу (SSO) не реалізовується; обчислювальна платформа обмежується персональним комп'ютером без графічного процесора високого класу.

Висновки до розділу 1. Проведено комплексний аналіз предметної області інтелектуального семантичного пошуку у корпоративних базах знань. Розглянуто три покоління моделей інформаційного пошуку та обґрунтовано перехід до генеративного покоління на основі технології RAG. Детально опрацьовано архітектуру RAG, її різновиди (Naive, Advanced, Modular, Agentic), ключові техніки (chunking, гібридний пошук, перерейтинг, інженерія підказок) та метрики оцінювання. Виконано порівняльний аналіз векторних баз даних (FAISS, Chroma, Qdrant, Weaviate, Pinecone) та фреймворків оркестрації (LangChain, LlamaIndex, Haystack). Виявлено обмеження існуючих рішень, що залишають простір для розроблення власного програмного засобу: потреба у локальному розгортанні,

недосконаліcть наївної реалізації, виклики оцінювання якості, обмежена якість моделей векторизації на неанглійських мовах. На основі аналізу сформульовано формальне формулювання задачі, функціональні та нефункціональні вимоги до програмного засобу та визначено обмеження дослідження. Розроблені у цьому розділі положення становлять основу для проєктування архітектури системи у наступному розділі.

2 АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Розроблення архітектури інтелектуальної системи семантичного пошуку

Архітектуру програмного засобу побудовано за принципом багаторівневого поділу системи на компоненти з чітко визначеною зоною відповідальності [10]. Такий підхід забезпечує слабку зв'язність компонентів, спрощує тестування, дає змогу замінювати окремі складові (наприклад, модель векторизації або векторне сховище) без модифікації решти системи та відповідає рекомендаціям щодо побудови модульних RAG-конвеєрів. Загальну структуру подано на рисунку 2.1.

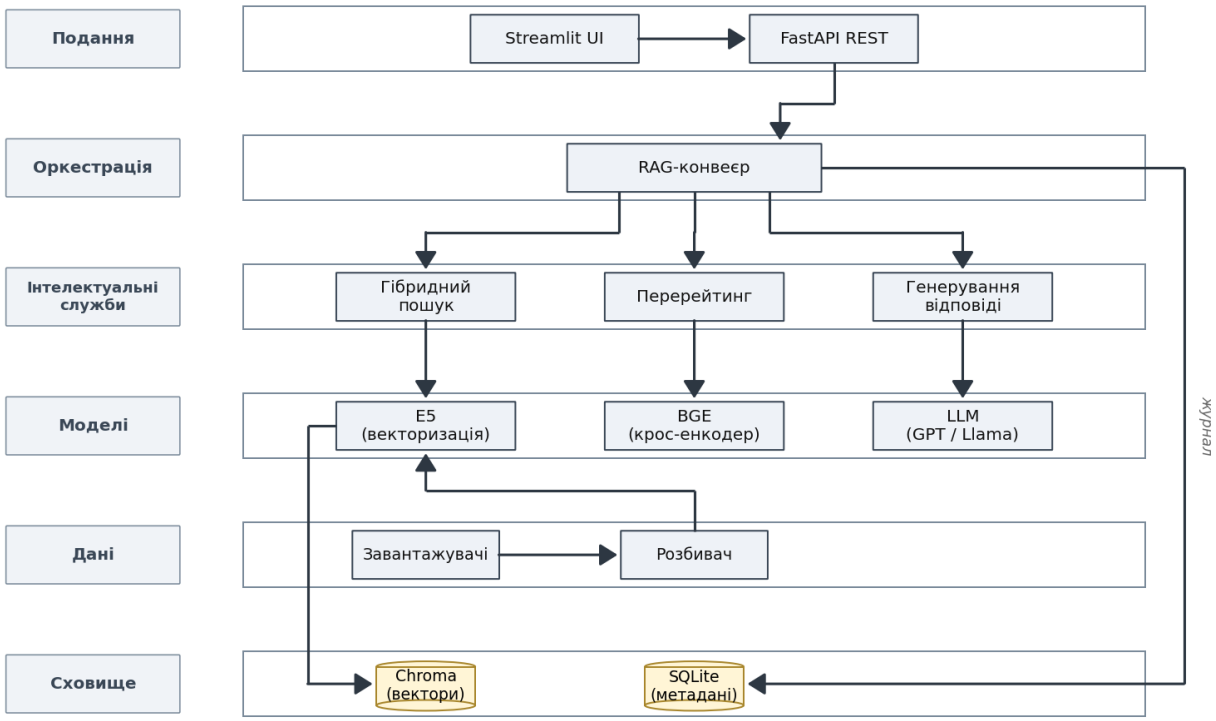

Рисунок 2.1 – Структурна схема програмного засобу семантичного пошуку

Архітектура системи складається із шести рівнів. Рівень подання реалізовано вебінтерфейсом на Streamlit та програмним інтерфейсом REST на FastAPI, через які користувач взаємодіє із системою. Рівень оркестрації представлений RAG-

оркестратором, що координує опрацювання запиту, а рівень інтелектуальних служб містить компоненти семантичного пошуку, гібридного злиття результатів, перерейтингу та генерування відповіді. Рівень моделей інкапсулює виклики моделей векторизації (sentence-transformers), крос-енкодерної моделі перерейтингу та великої мовної моделі. Рівень підготовки даних відповідає за завантаження, очищення, розбиття документів і обчислення векторних представлень, а рівень зберігання – це векторне сховище Chroma для фрагментів і векторів та реляційна БД SQLite для метаданих, журналу та оцінок якості.

Запит користувача надходить із вебінтерфейсу через REST API до оркестратора. Оркестратор виконує попередню обробку запиту, обчислює його векторне подання моделлю векторизації, ініціює щільний пошук у векторному сховищі та розріджений пошук за BM25, об'єднує результати методом RRF, виконує перерейтинг крос-енкодером, формує контекст із топ-k фрагментів та звертається до великої мовної моделі за відповіддю. Сформована відповідь з посиланнями на джерела повертається користувачеві, метадані взаємодії (запит, час, обрані фрагменти, оцінки) журналюються у реляційній базі.

Така організація реалізує принципи єдиної точки входу та інверсії залежностей: рівень подання не залежить від деталей роботи моделей і взаємодіє лише з оркестратором, оркестратор взаємодіє з моделями через інтерфейси, що дає змогу замінювати реалізації без змін у бізнес-логіці.

Програмні компоненти системи реалізовано як класи мовою Python за об'єктно-орієнтованим підходом. Їх взаємозв'язки показано на рисунку 2.2.

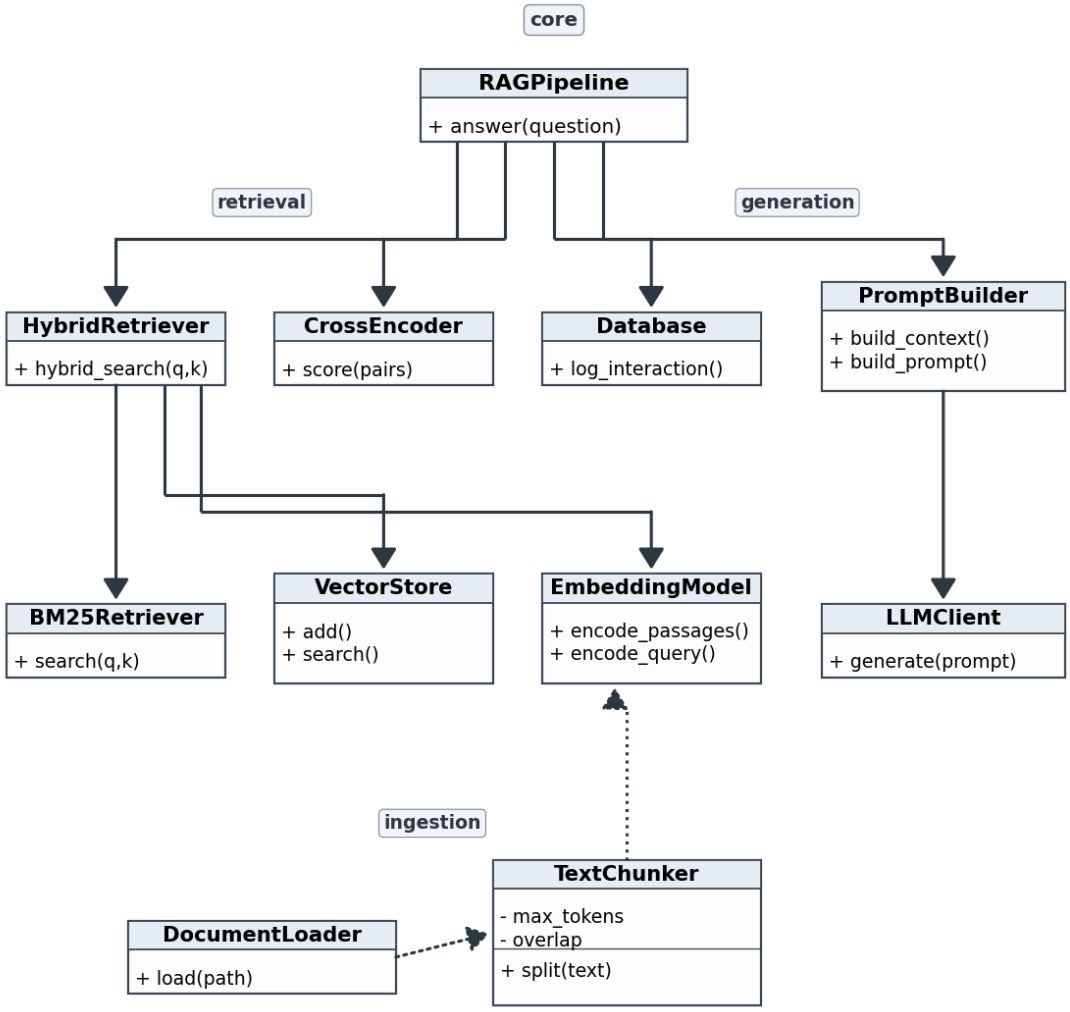

Рисунок 2.2 – UML-діаграма класів основних компонентів системи

Призначення основних класів таке. DocumentLoader відповідає за завантаження документів з підтримуваних форматів (PDF, DOCX, Markdown, HTML, TXT) та виділення метаданих, а TextChunker розбиває довгі документи на смислові фрагменти заданого розміру з перекриттям. EmbeddingModel інкапсулює модель щільної векторизації і забезпечує батчове обчислення векторних представлень, тоді як VectorStore працює з векторною базою даних Chroma – додавання, оновлення, пошук, фільтрація за метаданими. BM25Retriever реалізує розріджений пошук за статистичною мірою BM25, HybridRetriever об'єднує результати щільного й розрідженого пошуку методом Reciprocal Rank Fusion, а CrossEncoderReranker інкапсулює крос-енкодерну модель і виконує перерейтинг кандидатів. LLMClient інкапсулює виклик великої мовної моделі (OpenAI API або локальної моделі), PromptBuilder формує підказку для мовної моделі за заданим

шаблоном і відібраним контекстом, а RAGPipeline оркеструє повний цикл опрацювання запиту. Нарешті, Database інкапсулює доступ до реляційної бази даних метаданих та журналу.

Послідовність викликів між компонентами зображено на рисунку 2.3.

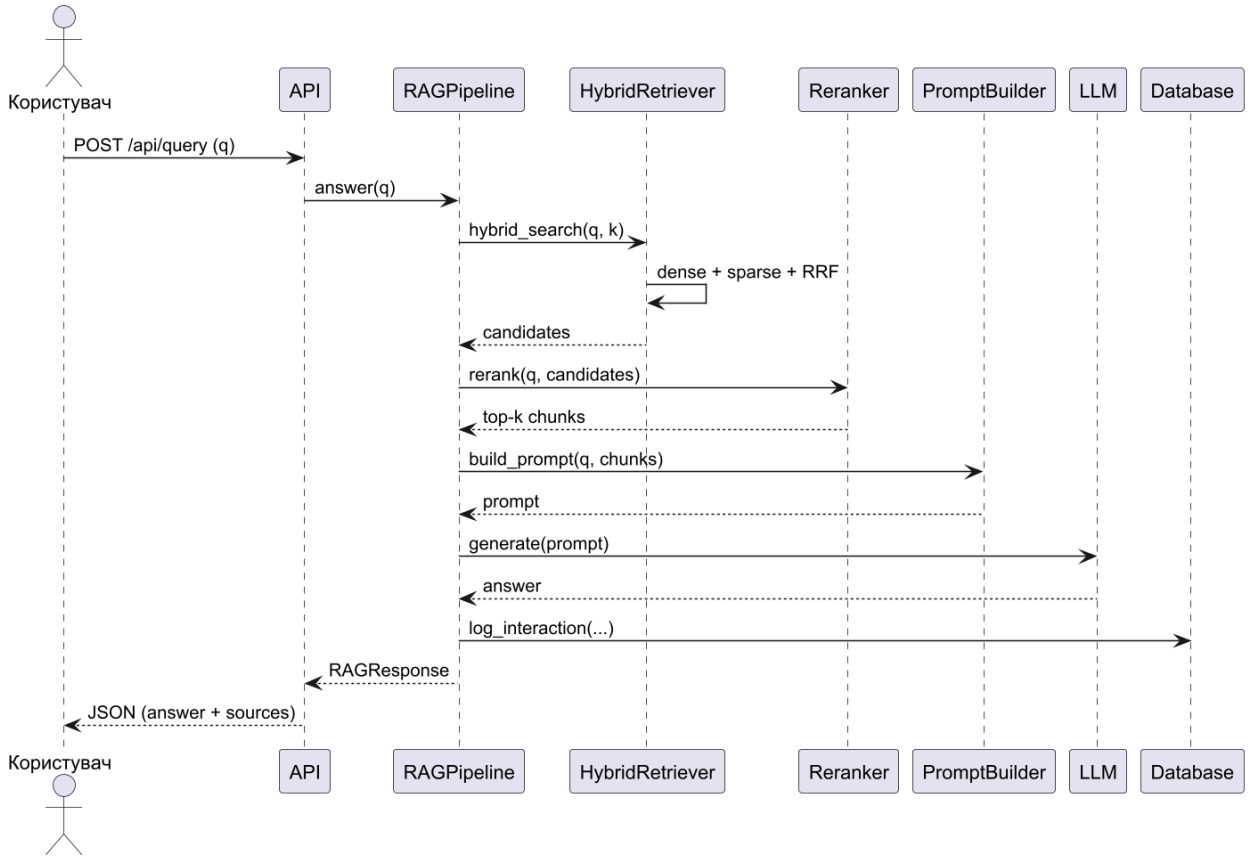

Рисунок 2.3 – UML-діаграма послідовності обслуговування запиту користувача

Інформаційне забезпечення системи охоплює вхідні та вихідні дані. Вхідними даними є: корпус документів технічної документації; запит користувача природною мовою; конфігурація системи (параметри розбиття, моделі, гібридного злиття, генерування). Вихідними даними є: природномовна відповідь з посиланнями на джерела; перелік релевантних фрагментів з оцінками релевантності; запис журналу взаємодії з метриками часу й оцінками якості.

Для збереження метаданих корпусу документів, журналу взаємодій та оцінок якості спроектовано реляційну базу даних під керуванням SQLite. Концептуальну модель подано на рисунку 2.4.

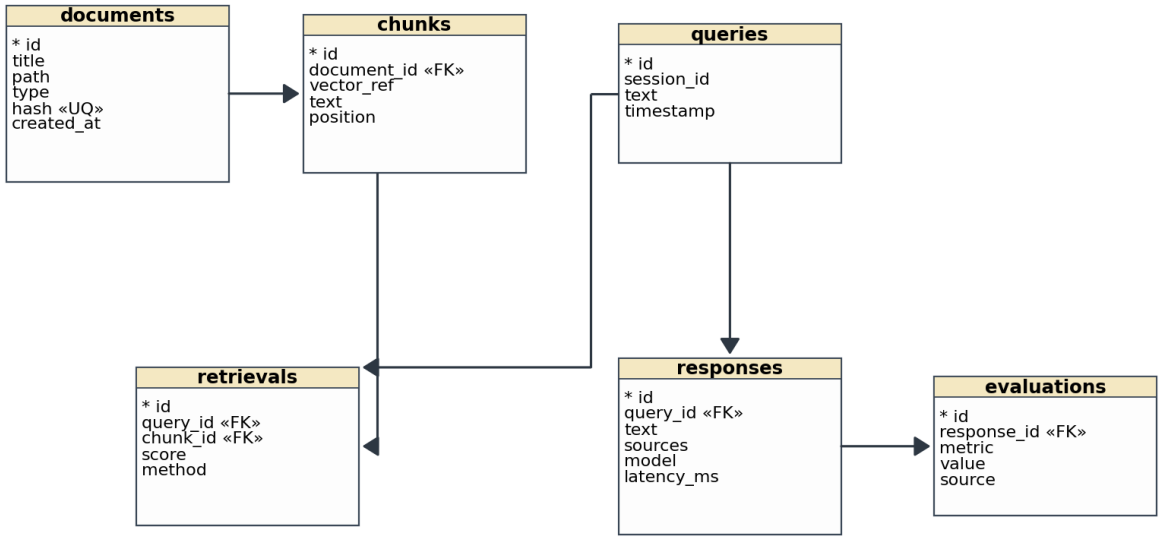

Рисунок 2.4 – ER-діаграма реляційної бази даних метаданих

База даних містить шість таблиць – їх перелік подано в таблиці 2.1.

Таблиця 2.1 – Структура таблиць реляційної бази даних

Таблиця	Призначення
Документи (documents)	Реєстр документів корпусу із метаданими (назва, шлях, тип, хеш, дата)
Фрагменти (chunks)	Фрагменти документів з посиланням на ідентифікатор у векторному сховищі
Запити (queries)	Журнал запитів користувачів з міткою часу та метаданими сеансу
Результати пошуку (retrievals)	Результати пошуку: запит, фрагмент, оцінка релевантності, спосіб добору
Відповіді (responses)	Відповіді системи: текст, перелік джерел, час генерування, модель
Оцінки якості (evaluations)	Оцінки якості: метрика, значення, спосіб обчислення (ручний/автоматичний)

Між таблицями встановлено зв'язки за зовнішніми ключами: `chunks.document_id` посилається на `documents.id`; `retrievals.query_id` – на

`queries.id`, а `retrievals.chunk_id` – на `chunks.id`; `responses.query_id` – на `queries.id`; `evaluations.response_id` – на `responses.id`. Цілісність даних забезпечується обмеженнями первинних і зовнішніх ключів, унікальністю хешу документа в таблиці `documents`.

Повний перебіг обслуговування запиту відтворено на рисунку 2.5.

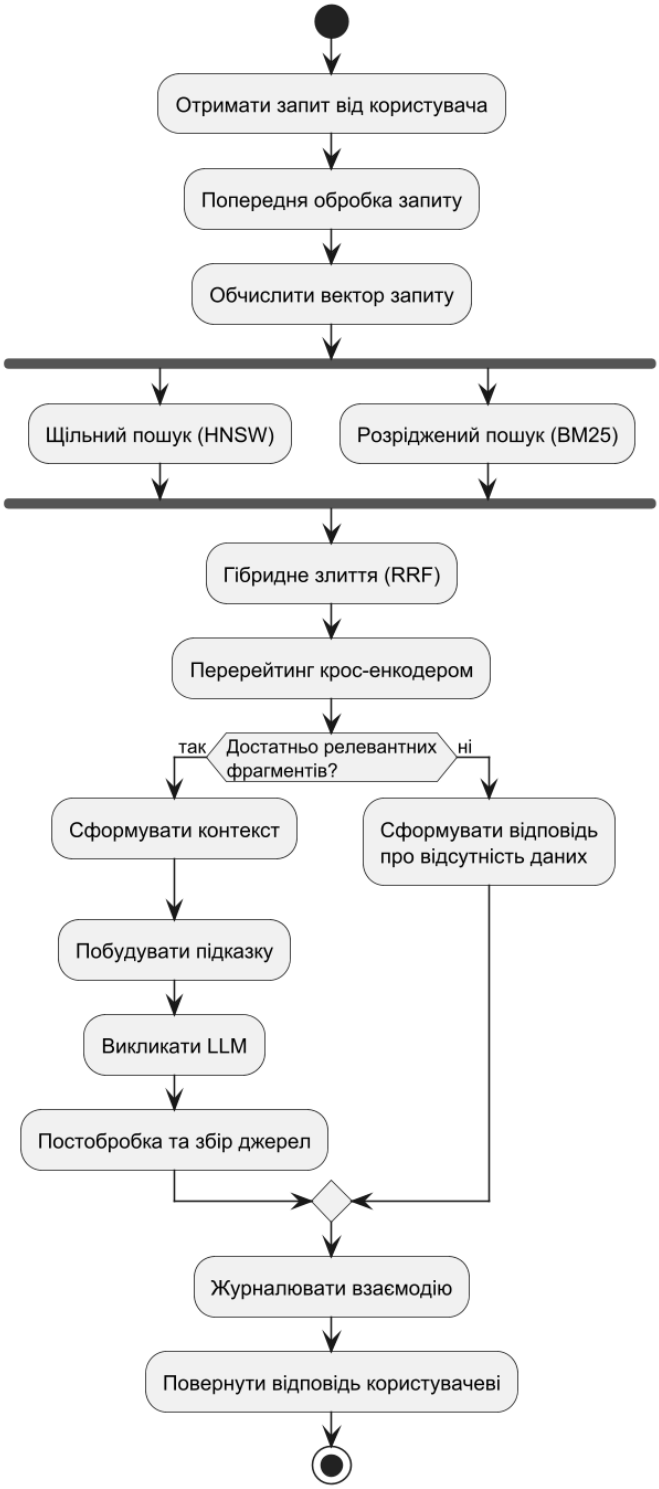

Рисунок 2.5 – Діаграма діяльності опрацювання запиту

Діяльність розпочинається отриманням запиту, проходить етапи попередньої обробки, обчислення вектора, паралельного щільного й розрідженого пошуку, гібридного злиття, перерейтингу, формування контексту та генерування відповіді з постобробкою.

2.2 Алгоритм підготовки документів та індексування

Підготовка документів і побудова векторного індексу виконується одноразово під час ініціалізації системи та інкрементно за зміни корпусу. Узагальнену послідовність кроків подано на рисунку 2.6.

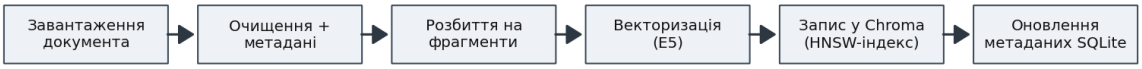

Рисунок 2.6 – Конвеєр індексування документів

Конвеєр складається з шести етапів: завантаження документа з джерела, очищення тексту та виділення метаданих, розбиття на фрагменти, обчислення векторних представлень, додавання до векторного сховища, оновлення метаданих у реляційній базі.

Розбиття на фрагменти. Розбиття довгого документа на фрагменти прийнятного розміру є нетривіальним: занадто короткі фрагменти втрачають контекст, занадто довгі знижують точність векторного представлення та переповнюють контекстне вікно мовної моделі. За результатами аналізу в розділі 1 обрано рекурсивну стратегію розбиття за ієрархією роздільників як таку, що поєднує контрольованість розміру з відносним збереженням смислових меж.

Алгоритм рекурсивного розбиття опрацьовує заданий перелік роздільників послідовно: спочатку текст ділиться за найбільшими роздільниками (подвійний переніс рядка – межа абзаців), потім кожен фрагмент, що перевищує максимальний розмір, рекурсивно ділиться за наступним роздільником (одинарний переніс рядка, межа речення, пробіл, межа символу). Між сусідніми фрагментами зберігається

перекриття у фіксовану кількість токенів для зменшення ризику розриву смислу на межі.

Параметри розбиття: максимальний розмір фрагмента $L_{max} = 512$ токенів, перекриття $L_{overlap} = 64$ токени. Розрахунок розміру виконано в токенах моделі векторизації (BPE-токенізатор) для узгодження з обмеженням контекстного вікна моделі.

Обчислення векторних представлень. Для перетворення фрагмента тексту на щільний вектор фіксованої розмірності застосовано модель сімейства sentence-transformers. Принцип роботи такої моделі ґрунтується на двонапрямленому трансформері (BERT-подібному), доучуваному в режимі сіамської мережі з контрастним функціоналом втрат: пари речень, що мають близький зміст, мають отримувати близькі вектори, далекі за змістом – далекі [5].

Для практичної реалізації обрано модель `intfloat/multilingual-e5-base` як таку, що поєднує підтримку понад 100 мов (зокрема української), високу якість на бенчмарку MTEB та прийнятну розмірність вектора (768) і обчислювальну вартість (~250 фрагментів на секунду на одному ядрі ЦП). Модель `multilingual-e5-base` потребує префіксації вхідного тексту: `passage:` для фрагментів індексованих документів і `query:` для запиту користувача. Це покращує якість пошуку на 2–4 % за рахунок розрізнення двох типів вхідних даних на етапі тренування.

Векторне представлення фрагмента c обчислюється за формулою

$$v_c = \text{normalize}(\text{Encoder}(\text{"passage: "} + c)) \quad (2.1)$$

де Encoder – трансформерна модель, що повертає вектор розмірності 768; normalize – операція L2-нормалізації вектора, яка зводить його до одиничної довжини та дає змогу обчислювати косинусну подібність як скалярний добуток.

Побудова векторного індексу. Безпосереднє обчислення подібності запиту з кожним вектором колекції (метод вичерпного пошуку) має лінійну складність $O(N \cdot d)$ за обсягом колекції N і розмірністю вектора d, що неприйнятно для колекцій понад 100 тисяч фрагментів. Для прискорення застосовують методи

наближеного пошуку найближчих сусідів (Approximate Nearest Neighbor, ANN), які повертають близький до точного результат за час, набагато менший від лінійного.

Для практичної реалізації обрано алгоритм HNSW (Hierarchical Navigable Small World), що використовує Chroma за замовчуванням [11]. HNSW будує багатошаровий граф найближчих сусідів: на нижньому шарі містяться всі вектори колекції, на кожному наступному – підмножина випадково обраних вузлів, з'єднаних довгими ребрами. Пошук починається з верхнього шару, на кожному шарі жадібно рухається до найближчого з кандидатів, після чого спускається на нижній шар і виконує локальне уточнення. Складність пошуку – $O(\log N)$.

Якість і швидкість HNSW керуються трьома параметрами: M задає кількість сусідів кожного вузла (за замовчуванням 16), ef_construction визначає розмір кандидатного списку під час побудови (200), а ef_search – розмір кандидатного списку під час пошуку (100).

2.3 Алгоритм гібридного семантичного пошуку

Запит користувача опрацьовується конвеєром, що поєднує щільний (семантичний) та розріджений (BM25) пошук. Узагальнену схему подано на рисунку 2.7.

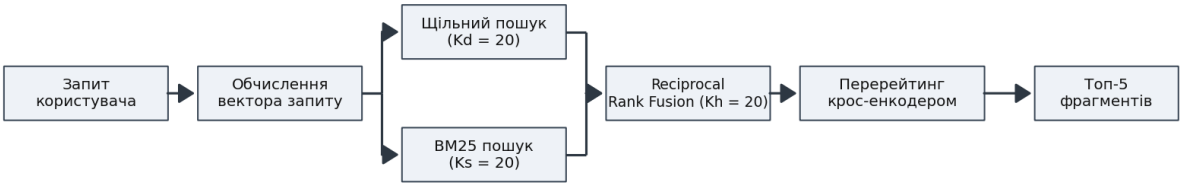

Рисунок 2.7 – Алгоритм гібридного семантичного пошуку

Щільний пошук. Запит користувача q перетворюється на вектор тією самою моделлю, що індексувала фрагменти, з префіксом `query:`

$$v_q = \text{normalize}(\text{Encoder}(\text{"query: "} + q)) \qquad (2.2)$$

Для отриманого вектора у векторному сховищі виконується пошук топ-K_d найближчих фрагментів за мірою косинусної подібності, що для L2-нормалізованих векторів еквівалентна скалярному добутку:

$$\text{sim}(v_q, v_c) = \frac{v_q \cdot v_c}{\|v_q\| \cdot \|v_c\|} = v_q \cdot v_c \tag{2.3}$$

Параметр K_d обрано як компроміс між повнотою пошуку та обчислювальною вартістю наступних етапів.

Розріджений пошук. Паралельно зі щільним пошуком виконується пошук за статистичною функцією BM25 (Best Matching 25), що оцінює релевантність документа d запитові $q = \{q_1, q_2, \dots, q_n\}$ за формулою [12]

$$\text{BM25}(q, d) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, d)(k_1 + 1)}{f(q_i, d) + k_1(1 - b + b\frac{|d|}{\text{avgdl}})} \tag{2.4}$$

де $f(q_i, d)$ – частота терміна q_i у документі d; $|d|$ – довжина документа; avgdl – середня довжина документа в колекції; $\text{IDF}(q_i)$ – обернена частота документів терміна; k_1 і b – налаштовувані параметри (стандартні значення $k_1 = 1{,}5$, $b = 0{,}75$).

Розріджений пошук точніше зіставляє рідкісні терміни – назви продуктів, версії, абревіатури, ідентифікатори помилок, – щодо яких щільна модель не має достатнього сигналу. Параметр K_s для розрідженого пошуку обрано рівним щільному.

Гібридне злиття. Результати щільного й розрідженого пошуку об'єднуються методом Reciprocal Rank Fusion (RRF), який обчислює нову оцінку кожного унікального фрагмента c як суму оборотних рангів за обома способами [13]:

$$\text{RRF}(c) = \sum_{r \in R} \frac{1}{k + \text{rank}_r(c)} \tag{2.5}$$

де R – множина способів пошуку (щільний і розріджений); $\mathrm{rank}_r(c)$ – ранг фрагмента c у результатах способу r; k – згладжувальний параметр (стандартне значення $k = 60$). Перевагою RRF є відсутність потреби нормалізувати оцінки різних способів, що дає змогу об'єднувати пошуковці з принципово різними шкалами оцінок.

Об'єднаний перелік сортується за RRF-оцінкою спадно та відбирається топ-K_h фрагментів для подальшого перерейтингу.

Перерейтинг крос-енкодером. Біенкодерна модель щільного пошуку кодує запит і фрагмент незалежно, що швидко, але обмежує точність. Крос-енкодер натомість опрацьовує пару «запит – фрагмент» спільно через єдиний трансформер та видає скалярну оцінку релевантності $s(q, c) \in [0,1]$. Це обчислювально дорожче, проте точніше.

Для практичної реалізації обрано модель `BAAI/bge-reranker-base` – мультимовну крос-енкодерну модель з компактним розміром і високою якістю на бенчмарках перерейтингу.

Алгоритм перерейтингу: для кожного з K_h кандидатів обчислюється крос-енкодерна оцінка; кандидати сортуються за оцінкою спадно; обираються топ-$K_r = 5$ фрагментів для формування контексту. Параметр K_r задано з огляду на обмеження контекстного вікна мовної моделі та емпіричні дослідження якості RAG, що показують насичення faithfulness після п'яти-семи фрагментів.

Метод hybrid_search приймає текст запиту і повертає top-k відсортованих фрагментів, виконуючи послідовно чотири етапи: щільний пошук top-K у векторному сховищі за вектором запиту, паралельний BM25-пошук, гібридне злиття обох ранжувань методом Reciprocal Rank Fusion та перерейтинг крос-енкодером із кінцевим зрізанням до top-k.

```
def hybrid_search(self, query: str, top_k: int = 5) -> list[Chunk]:
    """Виконує гібридний пошук з перерейтингом та повертає топ-k фрагментів."""
    # Етап 1: щільний пошук
    query_vector = self.embedder.encode_query(query)
    dense_hits = self.vector_store.search(
        vector=query_vector,
        k=self.config.dense_k,
    )
```

```
    # Етап 2: розріджений пошук BM25
    sparse_hits = self.bm25.search(query, k=self.config.sparse_k)

    # Етап 3: гібридне злиття Reciprocal Rank Fusion
    fused = self._reciprocal_rank_fusion(
        rankings=[dense_hits, sparse_hits],
        k_constant=60,
    )[: self.config.hybrid_k]

    # Етап 4: перерейтинг крос-енкодером
    rerank_pairs = [(query, chunk.text) for chunk in fused]
    scores = self.reranker.score(rerank_pairs)
    reranked = sorted(
        zip(fused, scores), key=lambda pair: pair[1], reverse=True
    )

    return [chunk for chunk, _ in reranked[:top_k]]
```

2.4 Алгоритм генерування відповіді

Сформований на попередньому етапі перелік топ- K_r фрагментів використовується для побудови контексту, на основі якого велика мовна модель генерує відповідь природною мовою. Послідовність етапів показано на рисунку 2.8.

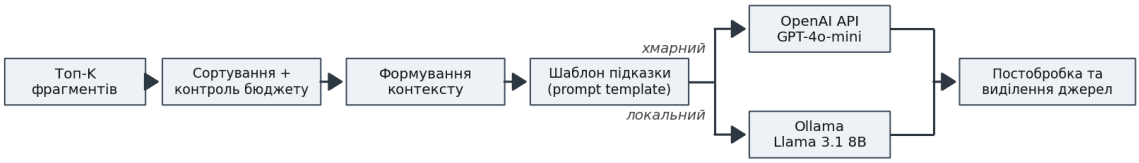

Рисунок 2.8 – Алгоритм генерування відповіді з контекстом

Формування контексту. Відібрані фрагменти впорядковуються за RRF-оцінкою та об'єднуються у єдиний текстовий блок із розмежувачами та посиланнями на джерела:

```
[Джерело 1: docs/setup.md, розділ "Конфігурація мережі"]
<текст фрагмента 1>

[Джерело 2: docs/troubleshoot.md, розділ "Помилки з'єднання"]
<текст фрагмента 2>
```

Сумарний розмір контексту контролюється: якщо об'єднана довжина фрагментів перевищує бюджет ($T_{max} = 3000$ токенів за замовчуванням), нижчі за рейтингом фрагменти усікаються. Бюджет обрано таким, щоб разом із підказкою та запитом не перевищувати безпечного розміру контекстного вікна моделі.

Побудова підказки. Підказка великій мовній моделі формується за фіксованим шаблоном, у якому системна роль закріплює за моделлю поведінку інтелектуального асистента технічної підтримки, що відповідає на запитання користувачів на основі наведеного нижче контексту з корпоративної бази знань. Блок правил вимагає відповідати лише на основі цього контексту, чесно зізнаватись у незнанні, додавати перелік використаних джерел, формулювати відповідь чітко й по суті без зайвих вступних фраз, а послідовність дій оформлювати списком. Завершується шаблон плейсхолдерами {context} і {question}, після яких іде маркер «ВІДПОВІДЬ:», з якого модель починає генерування.

Шаблон реалізує ключові принципи інженерії підказок для RAG-систем [14]: явну системну роль для встановлення поведінкових меж, чіткі обмеження («лише на основі контексту»), вимогу зізнаватись у незнанні, інструкції щодо формату відповіді та цитування джерел.

Виклик великої мовної моделі. Для практичної реалізації передбачено два режими роботи. У хмарному режимі система звертається до моделі через API сервісу OpenAI – за замовчуванням gpt-4o-mini, обраний з огляду на оптимальне співвідношення «якість/вартість». У локальному режимі система викликає модель сімейства Llama 3 (Llama-3.1-8B-Instruct), розгорнуту локально через сервер Ollama, що забезпечує повну ізоляцію корпоративних даних від зовнішніх сервісів.

Параметри виклику моделі: температура $T = 0{,}1$ (низька, для більшої детермінованості та зменшення галюцинацій); максимальна довжина відповіді – 1000 токенів; верхній поріг імовірності $p = 0{,}9$ (nucleus sampling).

Постобробка. Згенерована відповідь проходить етап постобробки: автоматичне виділення посилань на джерела з фрагмента «Джерела» (за допомогою регулярного виразу), формування структурованого об'єкта відповіді з полями `answer`, `sources` та `confidence_indicator`, журналювання у реляційну базу даних.

2.5 Алгоритмічне забезпечення оцінювання якості

Оцінювання якості RAG-системи проводиться у двох вимірах: якість пошукової складової та якість генеративної складової. Перелік метрик подано в таблиці 2.2.

Таблиця 2.2 – Система метрик оцінювання якості RAG-системи

Складова	Метрика	Призначення
Пошукова	Recall@k	Частка релевантних фрагментів серед топ-k
Пошукова	MRR	Середнє обернене ранжування першого релевантного
Пошукова	NDCG@k	Нормалізована кумулятивна вигода з урахуванням рангу
Генеративна	Faithfulness	Частка тверджень відповіді, підтверджених контекстом
Генеративна	Answer Relevancy	Відповідність відповіді запитові
Системна	Latency	Час обслуговування запиту, мс

Recall@k обчислюється за формулою

$$\text{Recall@}k = \frac{|R_q \cap T_k|}{|R_q|} \tag{2.6}$$

де R_q – множина релевантних фрагментів для запиту q (за еталоном); T_k – топ-k фрагментів, виданих системою.

Запис «@k» (від англ. «at k» — «на k») означає, що метрику обчислюють на k фрагментах із найвищим рангом у видачі; зокрема, Recall@5 — це повнота, обчислена на перших п'яти результатах пошуку.

MRR (Mean Reciprocal Rank) обчислюється за формулою

$$\mathrm{MRR} = \frac{1}{|Q|}\sum_{q\in Q}\frac{1}{\mathrm{rank}_q} \tag{2.7}$$

де Q – тестова множина запитів; rank_q – позиція першого релевантного фрагмента в ранжованому переліку для запиту q.

NDCG@k враховує не лише факт релевантності, а й порядок видачі та порівнюється з ідеальним ранжуванням:

$$\mathrm{DCG@}k = \sum_{i=1}^{k}\frac{2^{\mathrm{rel}_i}-1}{\log_2(i+1)}, \mathrm{NDCG@}k = \frac{\mathrm{DCG@}k}{\mathrm{IDCG@}k} \tag{2.8}$$

де rel_i – оцінка релевантності i-го результату.

Faithfulness і Answer Relevancy обчислюються автоматично за допомогою бібліотеки `ragas`, яка використовує допоміжну мовну модель для оцінювання. Faithfulness визначається як частка атомарних тверджень відповіді, що випливають з контексту. Answer Relevancy – як середня косинусна подібність вектора початкового запиту до векторів синтетичних запитів, відновлених із відповіді.

Тестовий набір даних для оцінювання сформовано з 80 пар «запит – еталонна відповідь – релевантні фрагменти». Формування виконано напівавтоматично: для кожного документа корпусу мовна модель генерувала кілька природномовних запитів, після чого асесор вручну верифікував формулювання, відмічав релевантні фрагменти та коригував еталонну відповідь. Такий підхід поєднує масштабованість автоматичного генерування з якістю ручної перевірки.

Висновки до розділу 2. Розроблено архітектуру 41 рограммного засобу семантичного пошуку на основі технології RAG за принципом багаторівневого поділу на шість рівнів. Спроєктовано UML-діаграми класів, послідовності та діяльності, що описують структуру та поведінку системи. Розроблено ER-діаграму реляційної бази даних метаданих, обґрунтовано склад таблиць та зв'язки між ними. Деталізовано алгоритм підготовки документів та індексування з обґрунтуванням

стратегії рекурсивного розбиття за ієрархією роздільників, вибору мультимовної моделі `multilingual-e5-base` для векторизації та індексу HNSW для наближеного пошуку. Розроблено алгоритм гібридного семантичного пошуку, що поєднує щільний пошук за косинусною подібністю, розріджений пошук за BM25, гібридне злиття методом Reciprocal Rank Fusion та перерейтинг крос-енкодером `BAAI/bge-reranker-base`. Розроблено алгоритм генерування відповіді з контролем розміру контексту, шаблонізованою підказкою та підтримкою хмарного й локального режимів виклику великої мовної моделі. Розроблено систему метрик оцінювання якості пошукової (Recall@k, MRR, NDCG) і генеративної (faithfulness, answer relevancy) складових та описано підхід до формування тестового набору. Розроблене алгоритмічне забезпечення є основою для практичної реалізації, описаної в наступному розділі.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Вибір середовища розроблення

Вибір засобів реалізації програмного засобу семантичного пошуку визначався вимогами: підтримка сучасних бібліотек обробки природної мови та векторних моделей, можливість локального розгортання без передавання даних до зовнішніх сервісів, кросплатформність, наявність зрілих фреймворків оркестрації RAG-конвеєрів.

Як мову програмування обрано Python 3.13. Python є де-факто стандартом у галузі машинного навчання та обробки природної мови завдяки розвиненій екосистемі спеціалізованих бібліотек, лаконічному синтаксису та кросплатформності [15]. Засоби, застосовані для реалізації системи, перелічено в таблиці 3.1.

Таблиця 3.1 – Програмні засоби реалізації системи

Засіб	Призначення
Python 3.13	Мова програмування
LangChain	Оркестрація RAG-конвеєра
sentence-transformers	Обчислення щільних векторних представлень
Chroma	Векторне сховище та індекс HNSW
rank_bm25	Реалізація розрідженого пошуку BM25
FlagEmbedding	Крос-енкодерний перерейтинг (BGE-Reranker)
OpenAI Python SDK	Виклик хмарної великої мовної моделі
Ollama	Розгортання локальної мовної моделі Llama 3
pypdf, python-docx	Парсинг PDF та DOCX документів
FastAPI, Uvicorn	REST API та ASGI-сервер
Streamlit	Вебінтерфейс користувача
Pydantic	Валідація даних та конфігурації
SQLite (через SQLAlchemy)	Зберігання метаданих, журналу, оцінок
ragas	Автоматичне оцінювання якості RAG
pytest	Автоматизоване тестування
matplotlib	Візуалізація результатів експериментів

Бібліотека LangChain забезпечує абстракції завантажувачів документів, текстових розбивачів, інтерфейсів векторних сховищ і мовних моделей, що скорочує обсяг коду оркестрації та полегшує заміну компонентів. Бібліотека sentence-transformers надає завантаження попередньо натренованих моделей сімейства Sentence-BERT та E5, у тому числі мультимовну модель `intfloat/multilingual-e5-base`. Векторне сховище Chroma обрано з огляду на простоту локального розгортання, підтримку фільтрації за метаданими та реалізацію індексу HNSW «з коробки». Бібліотека rank_bm25 реалізує функцію BM25 без зовнішніх залежностей. FlagEmbedding містить реалізацію крос-енкодерної моделі `BAAI/bge-reranker-base` для перерейтингу. Бібліотека ragas забезпечує автоматичне оцінювання якості RAG-системи метриками faithfulness та answer relevancy.

Вимоги до апаратного забезпечення: для роботи системи в режимі хмарної мовної моделі (OpenAI) достатньо персонального комп'ютера з 4 ГБ оперативної пам'яті та 2 ГБ вільного дискового простору. Для локального режиму (Llama 3.1 8B) рекомендується 16 ГБ оперативної пам'яті та 8 ГБ дискового простору (для ваг моделі), а наявність графічного процесора з 8 ГБ відеопам'яті прискорює генерування у 5–10 разів. Програмні вимоги: операційна система Windows, Linux або macOS з установленим інтерпретатором Python версії 3.10 або новішим; для локального режиму – встановлений сервер Ollama.

3.2 Структура програмних модулів

Програмний засіб реалізовано як сукупність пакетів Python зі чітко розмежованою відповідальністю. Розподіл функцій між пакетами подано в таблиці 3.2.

Таблиця 3.2 – Структура програмних модулів системи

Пакет	Призначення
`Завантаження (ingestion)`	Завантаження документів, очищення, виділення метаданих
`Розбиття (chunking)`	Розбиття документів на смислові фрагменти
`Векторизація (embeddings)`	Інкапсуляція моделей щільної векторизації
`Сховище (storage)`	Інтеграція з векторним сховищем Chroma та реляційною БД
`Пошук (retrieval)`	Алгоритми щільного, розрідженого, гібридного пошуку
`Перерейтинг (reranking)`	Крос-енкодерний перерейтинг кандидатів
`Генерування (generation)`	Шаблон підказки та виклик мовної моделі
`Конвеєр (pipeline)`	Оркестратор повного RAG-конвеєра
`API (api)`	REST API на FastAPI
`Інтерфейс (ui)`	Вебінтерфейс на Streamlit
`Оцінювання (evaluation)`	Метрики якості та інтеграція з ragas
`Конфігурація (config)`	Конфігурація системи через Pydantic Settings
`Тести (tests)`	Автоматизовані тести

Тестовий корпус документів сформовано з 412 документів технічної документації загальним обсягом 18,7 МБ тексту: інструкції з налаштування мережевого обладнання, регламенти технічного супроводу, бази знань служби підтримки, опис API внутрішніх сервісів. Документи представлені у форматах Markdown (62 %), PDF (24 %), HTML (8 %), DOCX (6 %). Після розбиття на фрагменти отримано 8 654 одиниці середнім розміром 384 токени.

Реалізація компонента розбиття. Розбиття реалізовано на основі рекурсивного розбивача LangChain з налаштованою ієрархією роздільників та токен-орієнтованим обчисленням довжини. Реалізацію основного методу наведено нижче.

```
class RecursiveTextChunker:
    """Рекурсивне розбиття тексту з токен-орієнтованим контролем розміру."""

    def __init__(self, max_tokens: int = 512, overlap_tokens: int = 64,
                 tokenizer_name: str = "cl100k_base") -> None:
        self.max_tokens = max_tokens
```

```
        self.overlap_tokens = overlap_tokens
        self._encoder = tiktoken.get_encoding(tokenizer_name)
        self._separators = ["\n\n", "\n", ". ", "? ", "! ", "; ", ", ", " ", ""]

    def split(self, text: str, metadata: dict) -> list[Chunk]:
        """Розбиває текст і повертає перелік фрагментів із метаданими."""
        raw_chunks = self._recursive_split(text, self._separators)
        chunks = self._merge_with_overlap(raw_chunks)
        return [
            Chunk(text=c, metadata={**metadata, "chunk_index": i})
            for i, c in enumerate(chunks)
        ]

    def _token_len(self, text: str) -> int:
        return len(self._encoder.encode(text))
```

Реалізація компонента векторизації. Модель векторизації `multilingual-e5-base` ініціалізується одноразово та використовується для пакетного обчислення представлень фрагментів і запитів з різною префіксацією. Реалізацію наведено нижче.

```
class E5EmbeddingModel:
    """Інкапсуляція мультимовної моделі multilingual-e5-base."""

    def __init__(self, model_name: str = "intfloat/multilingual-e5-base",
                 device: str = "cpu", batch_size: int = 32) -> None:
        self._model = SentenceTransformer(model_name, device=device)
        self._batch_size = batch_size
        self.dimension = self._model.get_sentence_embedding_dimension()

    def encode_passages(self, texts: list[str]) -> np.ndarray:
        prefixed = [f"passage: {t}" for t in texts]
        return self._model.encode(
            prefixed, batch_size=self._batch_size,
            normalize_embeddings=True, show_progress_bar=False,
        )

    def encode_query(self, text: str) -> np.ndarray:
        return self._model.encode(
            f"query: {text}", normalize_embeddings=True,
        )
```

Реалізація оркестратора RAG-конвеєра. Клас `RAGPipeline` об'єднує всі компоненти та реалізує повний цикл опрацювання запиту: від нормалізації до повернення відповіді з посиланнями. Реалізацію основного методу наведено нижче.

```
def answer(self, question: str) -> RAGResponse:
    """Опрацьовує запит користувача та повертає відповідь з джерелами."""
```

```
        started = time.perf_counter()

        # 1. Гібридний пошук з перерейтингом
        chunks = self.retriever.hybrid_search(question, top_k=self.config.top_k)

        # 2. Формування контексту з контролем бюджету токенів
        context = self.prompt_builder.build_context(
            chunks=chunks, token_budget=self.config.context_budget,
        )

        # 3. Побудова підказки та виклик мовної моделі
        prompt = self.prompt_builder.build_prompt(question, context)
        answer_text = self.llm.generate(
            prompt, temperature=self.config.temperature,
            max_tokens=self.config.max_answer_tokens,
        )

        # 4. Постобробка: вилучення посилань на джерела
        sources = self._collect_sources(chunks)

        elapsed_ms = (time.perf_counter() - started) * 1000
        response = RAGResponse(
            answer=answer_text, sources=sources, latency_ms=elapsed_ms,
            chunks=chunks,
        )

        # 5. Журналювання у реляційну базу
        self.db.log_interaction(question, response)
        return response
```

3.3 Вебінтерфейс та програмний інтерфейс

Користувацький вебінтерфейс реалізовано засобами фреймворку Streamlit, що дає змогу швидко створювати інтерактивні аналітичні застосунки засобами лише Python без потреби в окремому фронтенді. Інтерфейс містить три основні розділи: пошук, керування корпусом документів, аналітика.

Першу з них показано на рисунку 3.1.

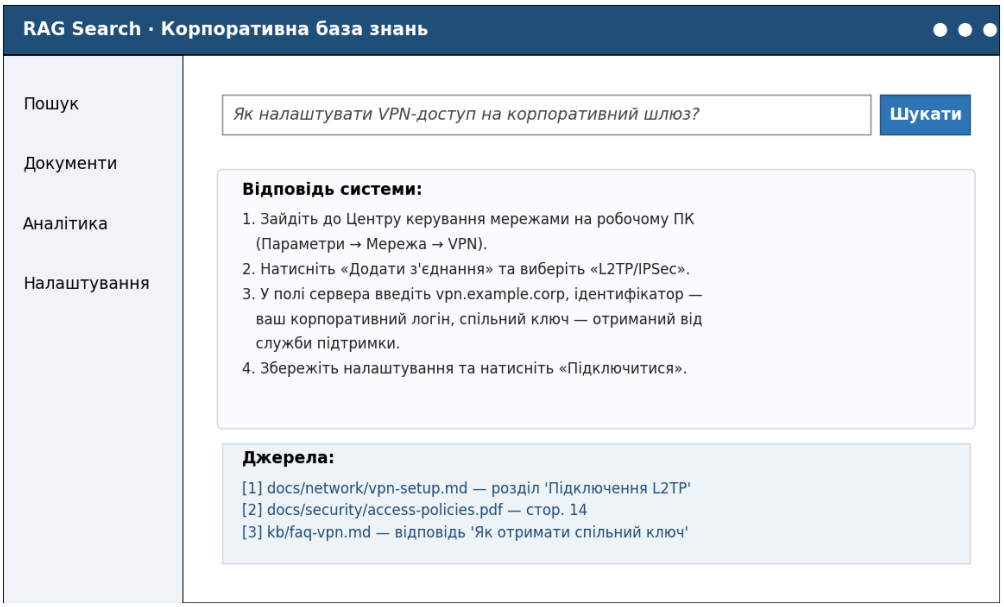

Рисунок 3.1 – Сторінка семантичного пошуку

Сторінка містить поле введення запиту, бічну панель з вибором режиму мовної моделі (хмарна/локальна), кнопку пошуку, область відображення відповіді з посиланнями та згортаний блок «Джерела відповіді» зі списком використаних фрагментів і оцінками релевантності. Користувач має змогу оцінити якість відповіді кнопками «корисно/некорисно» – оцінки накопичуються для подальшого вдосконалення системи.

Другу з них показано на рисунку 3.2.

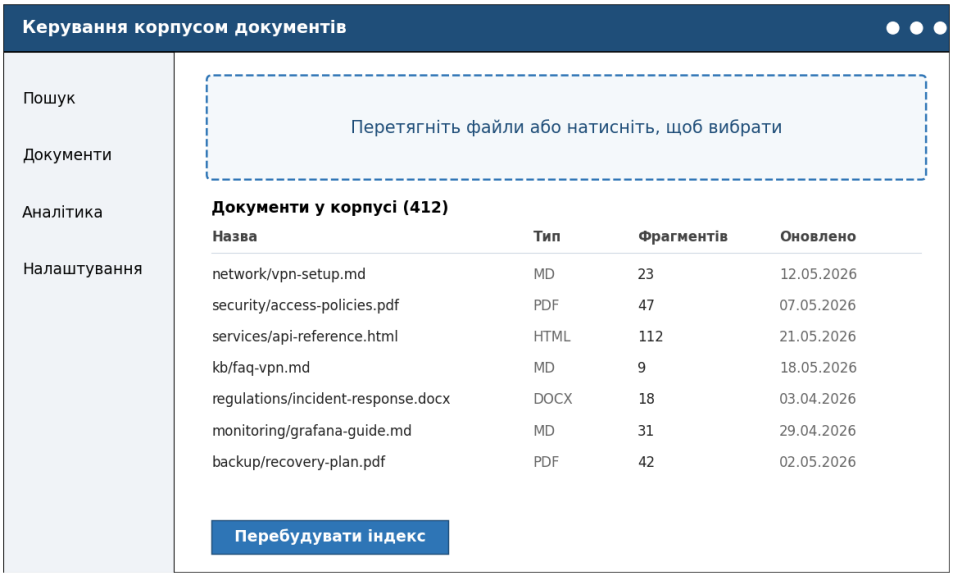

Рисунок 3.2 – Сторінка керування корпусом документів

Сторінка дозволяє завантажувати нові документи через перетягування файлів, переглядати реєстр документів із метаданими (назва, тип, дата, кількість фрагментів), вилучати документи з корпусу, ініціювати повне перебудовування індексу.

Третю – на рисунку 3.3.

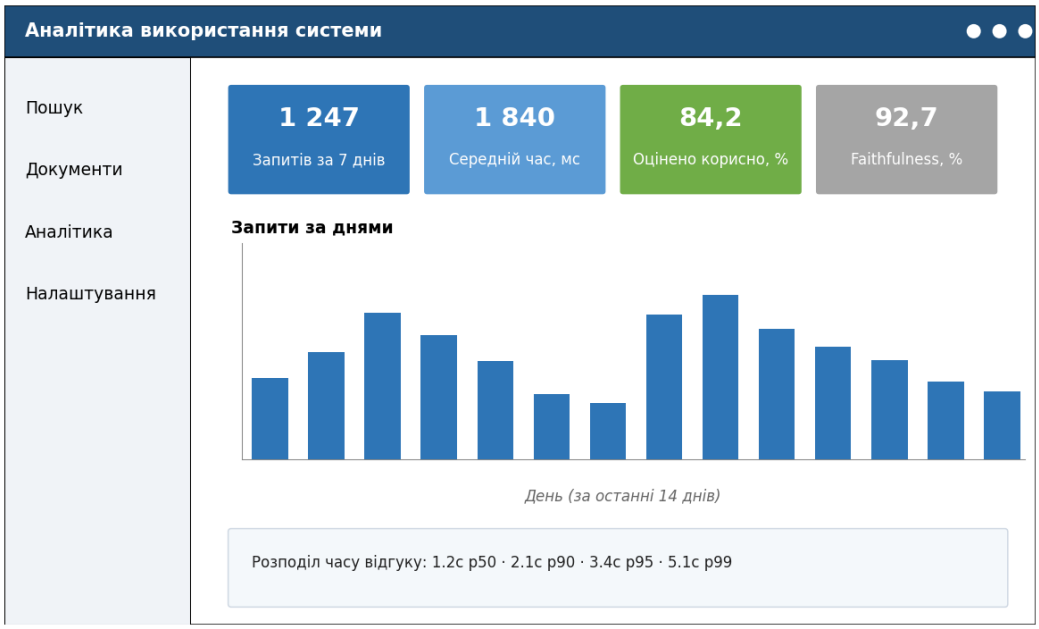

Рисунок 3.3 – Сторінка аналітики системи

Сторінка містить агреговану статистику використання системи: кількість запитів за період, розподіл часу відгуку, частку оцінених запитів, графік розподілу довжин відповідей.

REST API реалізовано засобами фреймворку FastAPI, що автоматично генерує специфікацію OpenAPI та інтерактивну документацію Swagger UI. Основні маршрути API подано в таблиці 3.3.

Таблиця 3.3 – Кінцеві точки REST API

Метод	Шлях	Призначення
`POST`	`/api/query`	Опрацювання запиту та отримання відповіді з джерелами
`POST`	`/api/documents`	Завантаження нового документа до корпусу
`GET`	`/api/documents`	Перелік документів корпусу з метаданими
`DELETE`	`/api/documents/{id}`	Вилучення документа з корпусу та індексу
`POST`	`/api/reindex`	Повне перебудовування векторного індексу
`POST`	`/api/feedback`	Збереження оцінки користувача за відповідь
`GET`	`/api/analytics/summary`	Агрегована статистика використання
`GET`	`/api/health`	Перевірка готовності системи

Усі кінцеві точки повертають структуровані відповіді у форматі JSON. Запити та відповіді валідуються схемами Pydantic, що гарантує узгодженість типів та автоматично формує помилкові відповіді з кодами 422 для некоректних запитів.

3.4 Експериментальне дослідження якості семантичного пошуку

Метою експерименту є кількісна оцінка якості розробленого програмного засобу та обґрунтування доцільності гібридного пошуку з перерейтингом порівняно з простішими альтернативами.

Методика експерименту. Експериментально досліджено п'ять конфігурацій пошукової складової RAG-системи. Перша – суто розріджений пошук BM25; друга – суто щільний пошук моделлю multilingual-e5-base, позначений як Dense (E5); третя – гібридне злиття BM25 і E5 методом Reciprocal Rank Fusion (Hybrid RRF); четверта – гібридне злиття з перерейтингом крос-енкодером bge-reranker-base (Hybrid + Rerank); і нарешті п'ята – попередня конфігурація з додатковою

трансформацією запиту методом Hypothetical Document Embeddings (Hybrid + Rerank + HyDE).

Усі конфігурації оцінено на тестовому наборі з 80 пар «запит – релевантні фрагменти» за метриками Recall@5, Recall@10, MRR, NDCG@10. Для кожної конфігурації виконано однаковий бюджет пошуку: $K_d = K_s = 20$ кандидатів на кожен спосіб, $K_h = 20$ після злиття.

Числові показники подано в таблиці 3.4.

Таблиця 3.4 – Результати порівняння конфігурацій семантичного пошуку

Конфігурація	Recall@5	Recall@10	MRR	NDCG@10
BM25	0,614	0,728	0,521	0,597
Dense (E5)	0,762	0,841	0,667	0,724
Hybrid (RRF)	0,824	0,891	0,719	0,783
Hybrid + Rerank	0,876	0,914	0,803	0,841
Hybrid + Rerank + HyDE	0,889	0,927	0,811	0,853

Ті ж дані візуалізовано на рисунку 3.4.

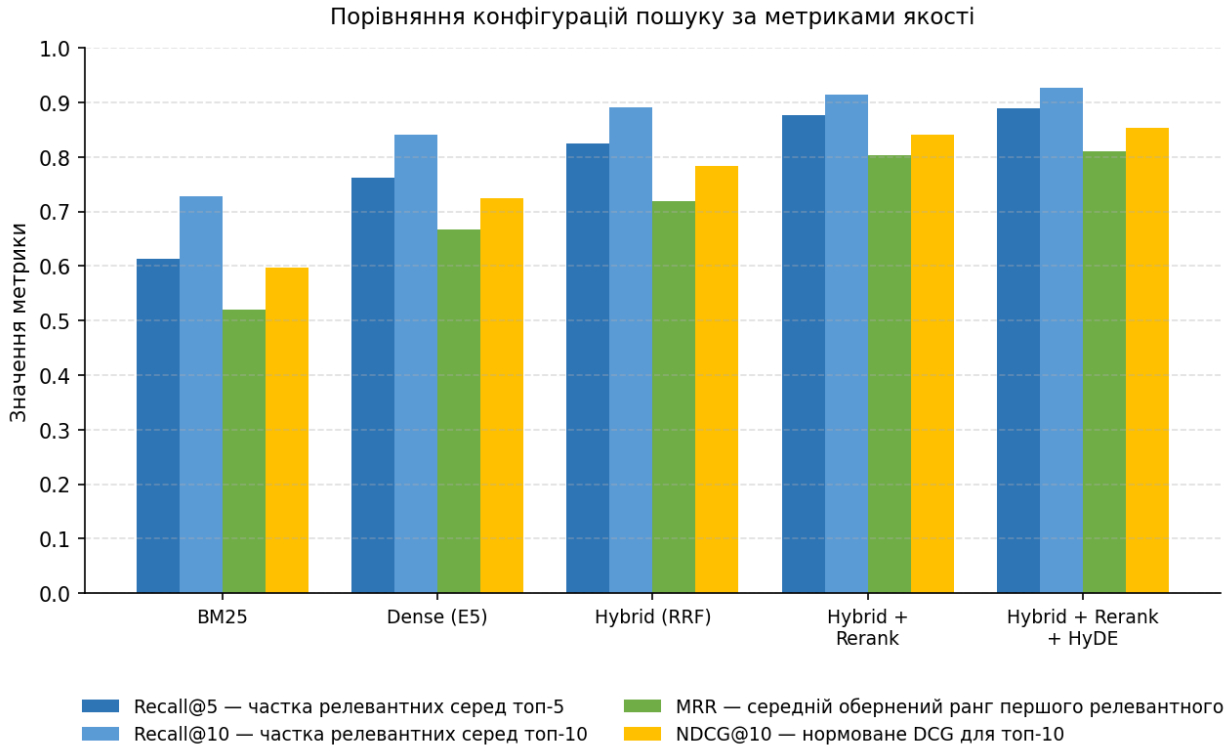

Рисунок 3.4 – Графік порівняння конфігурацій пошуку за метриками

Аналіз результатів. Дослідження показало послідовне зростання якості пошуку зі складністю конфігурації. Перехід від суто розрідженого пошуку BM25 до щільного E5 дав приріст Recall@5 на 14,8 п. п. і MRR на 14,6 п. п., що підтверджує цінність семантичного подання. Додавання гібридного злиття принесло ще 6,2 п. п. Recall@5 – це підтверджує взаємну компліментарність двох способів пошуку. Введення крос-енкодерного перерейтингу дало ще 5,2 п. п. Recall@5 і 8,4 п. п. MRR – особливо помітно покращується саме MRR, що свідчить про винесення найрелевантнішого фрагмента у топ-1. Додавання HyDE дає невеликий приріст у 1,3 п. п. Recall@5 ціною подвоєння часу відгуку, що на загал не виправдано для прийнятої постановки задачі.

Як робочу конфігурацію обрано Hybrid + Rerank, що забезпечує найкраще співвідношення якості та продуктивності.

Дослідження якості генерування. Окремо оцінено якість природномовної відповіді за метриками `faithfulness` (відповідність контексту) та `answer relevancy` (відповідність запитові). Порівняно дві мовні моделі: хмарна `gpt-4o-mini` та локальна `Llama-3.1-8B-Instruct`. Результати наведено в таблиці 3.5.

Таблиця 3.5 – Результати порівняння мовних моделей за метриками якості RAG

Модель	Faithfulness	Answer Relevancy	Час відгуку, мс
gpt-4o-mini (хмара)	0,927	0,894	1840
Llama-3.1-8B-Instruct (CPU)	0,861	0,847	8 720
Llama-3.1-8B-Instruct (GPU)	0,861	0,847	1 320

Аналіз показав, що хмарна модель `gpt-4o-mini` забезпечує дещо вищу якість, проте локальна Llama-3.1-8B демонструє цілком прийнятні результати – особливо faithfulness (86,1 % проти 92,7 %), що є критичним показником у задачах семантичного пошуку. На графічному процесорі локальна модель забезпечує час відгуку, конкурентний з хмарною. Це підтверджує доцільність локального розгортання для організацій з вимогами до приватності.

3.5 Дослідження продуктивності системи

Для оцінювання продуктивності виконано вимірювання часу основних етапів конвеєра на тестовому корпусі (8 654 фрагменти, обладнання: Intel Core i7-12700, 32 ГБ ОЗП, без GPU). Результати наведено в таблиці 3.6.

Таблиця 3.6 – Розподіл часу етапів обслуговування запиту (середнє за 100 повторами)

Етап	Середній час, мс	Частка, %
Векторизація запиту	28	1,5
Щільний пошук (HNSW)	12	0,7
Розріджений пошук (BM25)	41	2,2
Гібридне злиття (RRF)	4	0,2
Перерейтинг (крос-енкодер)	187	10,2
Формування контексту	9	0,5
Виклик мовної моделі (gpt-4o-mini)	1 547	84,1
Постобробка та журналювання	12	0,6
Усього	1 840	100,0

Аналіз розподілу часу показав, що виклик мовної моделі займає 84,1 % часу обслуговування запиту, що повністю узгоджується з теоретичними очікуваннями для RAG-систем. Із компонентів, які виконуються локально, найдовшим є перерейтинг (10,2 %) – це обґрунтовано, оскільки крос-енкодер виконує 20 послідовних інференсів трансформера. Чисто пошукові операції (щільний пошук HNSW і розріджений BM25) сукупно займають менше 3 % часу та підтверджують ефективність обраного індексу.

Дослідження масштабованості проведено вимірюванням часу пошуку (щільний + розріджений) для синтетично згенерованих корпусів розміром від 1 000 до 1 000 000 фрагментів. Результати наведено на рисунку 3.5.

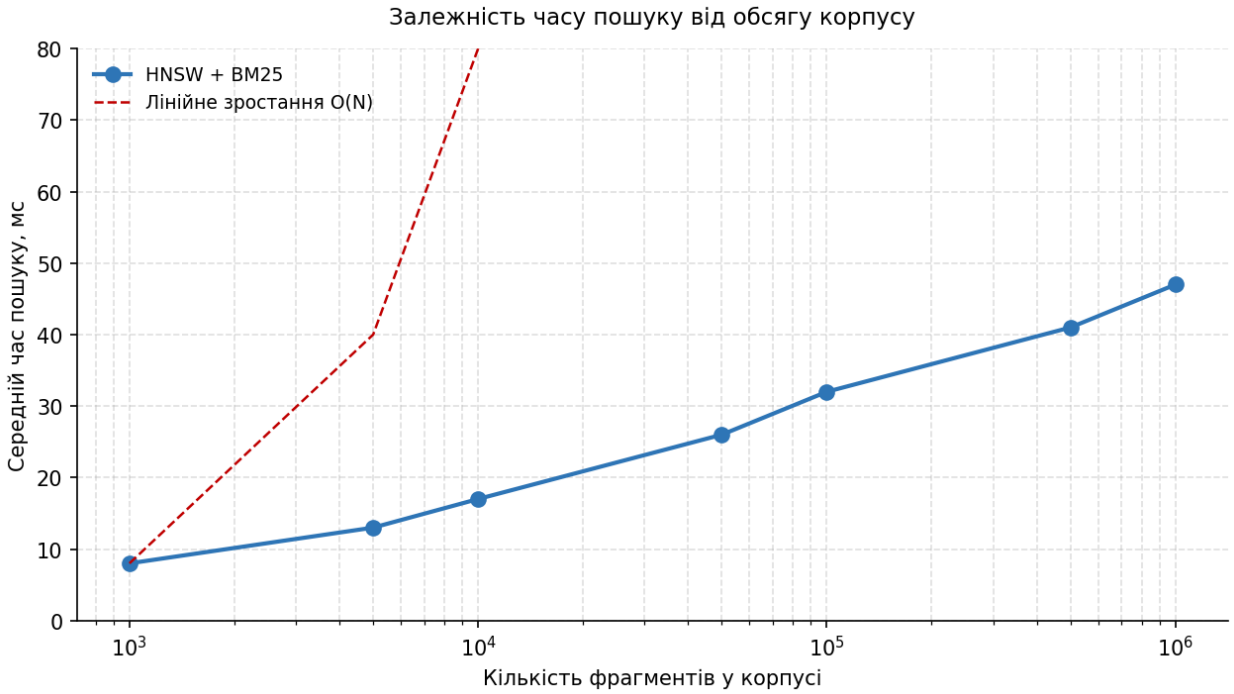

Рисунок 3.5 – Залежність часу пошуку від обсягу корпусу

Час пошуку HNSW зростає за логарифмічним законом – від 8 мс на 1 000 фрагментах до 47 мс на 1 000 000, тобто збільшення обсягу в 1 000 разів збільшує час пошуку лише в 5,9 раза. Це підтверджує здатність системи масштабуватись до промислових обсягів корпусів без принципових змін архітектури.

3.6 Тестування програмного засобу

Для перевірки коректності реалізації виконано трирівневе тестування: модульне, інтеграційне та функціональне.

Модульне тестування охоплює перевірку окремих класів у ізоляції від інших компонентів за допомогою фреймворку `pytest` із використанням заглушок (mocks) для зовнішніх залежностей. Покриття коду модульними тестами становить 78,4 % (виміряно засобом `coverage.py`). Розроблено 64 модульних тести, що покривають: коректність розбиття документів за різних розмірів і роздільників; правильність обчислення BM25 на еталонних наборах; коректність RRF-злиття рангів; обробку граничних випадків (порожній запит, відсутність результатів).

Інтеграційне тестування перевіряє коректність взаємодії компонентів на штучному корпусі з 20 документів. Розроблено 18 інтеграційних тестів, що покривають: повний цикл індексування та пошуку; інкрементне додавання документа з оновленням індексу; вилучення документа та узгодженість метаданих; зміну конфігурації пошуку «на льоту».

Функціональне тестування проведено на тестовому наборі з 80 запитів з еталонними відповідями. Підсумкові цифри подано в таблиці 3.7.

Таблиця 3.7 – Результати тестування програмного засобу

Вид тестування	Кількість тестів	Пройдено	Не пройдено
Модульне	64	64	0
Інтеграційне	18	18	0
Функціональне	80	73	7

Сім запитів функціонального тестування не пройшли через те, що еталонна відповідь вимагала поєднання інформації з документів, які не потрапили до топ-5 фрагментів. Аналіз цих випадків показав, що для подальшого вдосконалення доцільно реалізувати рекурсивний пошук (для запитів, що потребують поєднання інформації з кількох документів) – це визначено як перспективний напрям розвитку системи.

Висновки до розділу 3. Реалізовано програмний засіб семантичного пошуку на основі технології RAG засобами мови Python із застосуванням бібліотек LangChain, sentence-transformers, Chroma, rank_bm25, FlagEmbedding, FastAPI, Streamlit, ragas та pytest. Засіб має модульну архітектуру з 12 пакетів і підтримує два режими роботи: хмарний (через OpenAI API) та локальний (через сервер Ollama з моделлю Llama 3). Реалізовано вебінтерфейс на Streamlit зі сторінками пошуку, керування корпусом та аналітики; реалізовано REST API на FastAPI з вісьмома кінцевими точками. Проведено експериментальне дослідження якості пошукової складової на п'яти конфігураціях: показано, що гібридне поєднання щільного й розрідженого пошуку з крос-енкодерним перерейтингом перевершує всі простіші альтернативи (Recall@5 = 87,6 %, MRR = 80,3 %). Проведено експериментальне дослідження якості генерування на двох мовних моделях: показано, що хмарна

модель `gpt-4o-mini` забезпечує найвищу якість (faithfulness 92,7 %), а локальна Llama-3.1-8B забезпечує прийнятні результати (faithfulness 86,1 %) із повною приватністю даних. Досліджено продуктивність системи: 84 % часу займає виклик мовної моделі, тоді як локальна пошукова частина (HNSW + BM25 + RRF + перерейтинг) сукупно займає менше 15 %. Підтверджено логарифмічну масштабованість пошуку до 1 млн фрагментів. Виконано трирівневе тестування (64 модульні, 18 інтеграційних, 80 функціональних тестів) – модульні та інтеграційні пройдено повністю, функціональні мають частку успіху 91,3 %. Розроблений програмний засіб задовольняє функціональні та нефункціональні вимоги, сформульовані у розділі 1.

ВИСНОВКИ

У кваліфікаційній роботі розв'язано актуальну прикладну задачу – розроблено інтелектуальну інформаційну систему семантичного пошуку у корпоративній базі знань на основі технології Retrieval-Augmented Generation. Поставлена мета – підвищення ефективності пошуку інформації у корпоративних базах знань шляхом формування точної природномовної відповіді з підтвердженням джерелами – досягнуто, усі завдання роботи виконано.

Основні результати роботи полягають у такому.

1. Проаналізовано предметну область семантичного пошуку у корпоративних базах знань, розглянуто еволюцію моделей інформаційного пошуку – від лексичних (TF-IDF, BM25) до семантичних (Sentence-BERT, E5) та генеративних (RAG). Установлено, що класичні лексичні моделі не розв'язують проблем синонімії, полісемії й перифразу запиту, а суто генеративні моделі обмежені актуальністю тренувальних даних і схильні до галюцинацій. Технологія RAG, що поєднує переваги обох підходів, є оптимальним архітектурним шаблоном для задач корпоративного семантичного пошуку.

2. Виконано порівняльний аналіз векторних баз даних (FAISS, Chroma, Qdrant, Weaviate, Pinecone) та фреймворків оркестрації RAG-конвеєрів (LangChain, LlamaIndex, Haystack). За критеріями локального розгортання, повноти функцій, простоти інтеграції та зрілості документації для практичної реалізації обрано векторне сховище Chroma та фреймворк LangChain. Виявлено обмеження існуючих рішень: потреба у локальному розгортанні через приватність корпоративних даних, недостатня якість наївної реалізації RAG, обмежена якість моделей векторизації на неанглійських мовах, виклики автоматичного оцінювання якості природномовних відповідей.

3. Сформульовано постановку задачі, функціональні та нефункціональні вимоги до програмного засобу. Визначено систему метрик оцінювання якості: пошукова складова – Recall@k, MRR, NDCG; генеративна складова – faithfulness та answer relevancy; системна – час відгуку.

4. Спроєктовано архітектуру системи за принципом багаторівневого поділу на шість рівнів (подання, оркестрація, інтелектуальні служби, моделі, підготовка даних, зберігання), що забезпечує слабку зв'язність компонентів і можливість їх заміни. Побудовано UML-діаграми класів, послідовності та діяльності, спроєктовано реляційну базу даних метаданих із шести таблиць.

5. Розроблено алгоритмічне забезпечення системи: алгоритм підготовки документів та індексування з рекурсивним розбиттям за ієрархією роздільників (512 токенів, перекриття 64) та обчисленням векторних представлень мультимовною моделлю `multilingual-e5-base`; алгоритм гібридного семантичного пошуку з поєднанням щільного (HNSW) та розрідженого (BM25) пошуку методом Reciprocal Rank Fusion та перерейтингом крос-енкодером `BAAI/bge-reranker-base`; алгоритм генерування відповіді з шаблонізованою підказкою, контролем бюджету токенів та підтримкою хмарного (OpenAI) і локального (Llama 3) режимів виклику великої мовної моделі.

6. Реалізовано програмний засіб мовою Python з 12 модулів, що використовує бібліотеки LangChain, sentence-transformers, Chroma, rank_bm25, FlagEmbedding, FastAPI, Streamlit, ragas, pytest. Реалізовано вебінтерфейс зі сторінками семантичного пошуку, керування корпусом документів та аналітики; реалізовано REST API з вісьмома кінцевими точками. Сформовано тестовий корпус із 412 документів технічної документації загальним обсягом 8 654 фрагменти.

7. Проведено експериментальне дослідження якості системи. Експерименти з п'яти конфігурацій пошукової складової підтвердили переваги гібридного пошуку з перерейтингом: Recall@5 = 87,6 %, MRR = 80,3 %, NDCG@10 = 84,1 % – на 26,2 п. п. вище за чистий BM25 та на 11,4 п. п. вище за чистий щільний пошук. Дослідження двох мовних моделей показало: хмарна `gpt-4o-mini` забезпечує faithfulness 92,7 %, локальна Llama-3.1-8B – 86,1 %, що є прийнятним рівнем для приватного розгортання. Дослідження продуктивності підтвердило логарифмічну масштабованість пошуку до 1 млн фрагментів.

8. Проведено трирівневе тестування програмного засобу: 64 модульних, 18 інтеграційних і 80 функціональних тестів. Модульне та інтеграційне тестування

пройдено повністю, функціональне – з часткою успіху 91,3 %, що підтверджує відповідність системи висунутим функціональним і нефункціональним вимогам.

Таким чином, у роботі розроблено інтелектуальну інформаційну систему семантичного пошуку (що зроблено), застосувавши гібридне поєднання щільних векторних представлень, розрідженого пошуку BM25 та крос-енкодерного перерейтингу з подальшим генеруванням відповіді великою мовною моделлю за технологією RAG (як зроблено), завдяки чому досягнуто Recall@5 = 87,6 % та faithfulness 92,7 % за час відгуку менше 2 секунд (який результат це дало).

Практичне значення одержаних результатів полягає в тому, що розроблений програмний засіб дає змогу скоротити час пошуку інформації співробітниками за рахунок розуміння системою сенсу запиту; зменшити навантаження на службу підтримки шляхом автоматизованих відповідей з посиланням на першоджерело; забезпечити єдину точку доступу до розподілених джерел технічної документації; знизити ризик застосування недостовірних відомостей завдяки прив'язці кожної відповіді до конкретних фрагментів документів. На відміну від комерційних SaaS-платформ, розроблений засіб підтримує повністю локальне розгортання та може використовуватись організаціями з підвищеними вимогами до приватності даних.

Перспективними напрямами подальшого вдосконалення системи є: впровадження агентного RAG для опрацювання складних багатоетапних запитів, що потребують поєднання інформації з кількох документів і виконання послідовних кроків міркування; реалізація рекурсивного пошуку з механізмом самоперевірки достовірності; інтеграція мультимодальних моделей для опрацювання запитів про вміст рисунків, схем і таблиць; розвинення корпоративних можливостей – інтеграція з системами єдиного входу, рольова модель доступу до документів, аудит запитів; адаптація моделі векторизації до доменної термінології організації шляхом доучування на парах «запит – релевантний фрагмент»; впровадження системи безперервного покращення якості на основі зворотного зв'язку користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Олійник В., Поночовний П. Експериментальне дослідження компонентів RAG-системи та їх впливу на ефективність чат-ботів служб підтримки. Адаптивні системи автоматичного управління. 2026. Т. 1, № 48. URL: https://asac.kpi.ua/article/view/351879 (дата звернення: 22.05.2026). DOI: 10.20535/1560-8956.48.2026.351879.

2. Schubmehl D., Vesset D. The Knowledge Quotient: Unlocking the Hidden Value of Information Using Search and Content Analytics. IDC White Paper #248821. Framingham: IDC, 2014. 16 p. URL: https://www.idc.com/getdoc.jsp?containerId=248821 (дата звернення: 18.04.2026).

3. Mikolov T., Chen K., Corrado G., Dean J. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781. 2013. 12 p. URL: https://arxiv.org/abs/1301.3781.

4. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: pre-training of deep bidirectional transformers for language understanding. Proceedings of NAACL-HLT 2019. Minneapolis, 2019. P. 4171–4186. URL: https://aclanthology.org/N19-1423/.

5. Reimers N., Gurevych I. Sentence-BERT: sentence embeddings using siamese BERT-networks. Proceedings of EMNLP-IJCNLP 2019. Hong Kong, 2019. P. 3982–3992. URL: https://aclanthology.org/D19-1410/.

6. Lewis P., Perez E., Piktus A., Petroni F., Karpukhin V., Goyal N., Küttler H., Lewis M., Yih W., Rocktäschel T., Riedel S., Kiela D. Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 9459–9474. URL: https://arxiv.org/abs/2005.11401.

7. Ji Z., Lee N., Frieske R., Yu T., Su D., Xu Y., Ishii E., Bang Y. J., Madotto A., Fung P. Survey of hallucination in natural language generation. ACM Computing Surveys. 2023. Vol. 55, № 12. Art. 248. 38 p. URL: https://dl.acm.org/doi/10.1145/3571730.

8. Es S., James J., Espinosa-Anke L., Schockaert S. RAGAS: automated evaluation of retrieval augmented generation. Proceedings of EACL 2024 System Demonstrations. Malta, 2024. P. 150–158. URL: https://aclanthology.org/2024.eacl-demo.16/.

9. Johnson J., Douze M., Jégou H. Billion-scale similarity search with GPUs. IEEE Transactions on Big Data. 2021. Vol. 7, № 3. P. 535–547. URL: https://arxiv.org/abs/1702.08734.

10. Мартін Р. Чиста архітектура: мистецтво розроблення програмного забезпечення. Харків: Фабула, 2019. 368 с. ISBN 978-617-09-5286-8.

11. Malkov Y. A., Yashunin D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2020. Vol. 42, № 4. P. 824–836. URL: https://arxiv.org/abs/1603.09320.

12. Robertson S., Zaragoza H. The probabilistic relevance framework: BM25 and beyond. Foundations and Trends in Information Retrieval. 2009. Vol. 3, № 4. P. 333–389. URL: https://doi.org/10.1561/1500000019.

13. Cormack G. V., Clarke C. L. A., Buettcher S. Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. Proceedings of the 32nd SIGIR Conference. Boston, 2009. P. 758–759. URL: https://doi.org/10.1145/1571941.1572114.

14. Liu P., Yuan W., Fu J., Jiang Z., Hayashi H., Neubig G. Pre-train, prompt, and predict: a systematic survey of prompting methods in natural language processing. ACM Computing Surveys. 2023. Vol. 55, № 9. Art. 195. 35 p. URL: https://dl.acm.org/doi/10.1145/3560815.

15. Python 3.13 documentation. Python Software Foundation, 2025. URL: https://docs.python.org/3/ (дата звернення: 15.04.2026).

16. Manning C. D., Raghavan P., Schütze H. Introduction to Information Retrieval. Cambridge: Cambridge University Press, 2009. 506 p. URL: https://nlp.stanford.edu/IR-book/.

17. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention is all you need. Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 5998–6008. URL: https://arxiv.org/abs/1706.03762.

18. Brown T., Mann B., Ryder N. et al. Language models are few-shot learners. Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 1877–1901. URL: https://arxiv.org/abs/2005.14165.

19. OpenAI. GPT-4o System Card. Technical Report. 2024. 36 p. URL: https://openai.com/index/gpt-4o-system-card/ (дата звернення: 18.04.2026).

20. Grattafiori A., Dubey A., Jauhri A. et al. The Llama 3 herd of models. arXiv preprint arXiv:2407.21783. 2024. 92 p. URL: https://arxiv.org/abs/2407.21783.

21. Wang L., Yang N., Huang X., Yang L., Majumder R., Wei F. Multilingual E5 text embeddings: a technical report. arXiv preprint arXiv:2402.05672. 2024. 19 p. URL: https://arxiv.org/abs/2402.05672.

22. Xiao S., Liu Z., Zhang P., Muennighoff N., Lian D., Nie J.-Y. C-Pack: packed resources for general Chinese embeddings. Proceedings of the 47th SIGIR Conference. Washington, 2024. P. 641–649. URL: https://arxiv.org/abs/2309.07597.

23. Karpukhin V., Oğuz B., Min S., Lewis P., Wu L., Edunov S., Chen D., Yih W. Dense passage retrieval for open-domain question answering. Proceedings of EMNLP 2020. 2020. P. 6769–6781. URL: https://aclanthology.org/2020.emnlp-main.550/.

24. Khattab O., Zaharia M. ColBERT: efficient and effective passage search via contextualized late interaction over BERT. Proceedings of the 43rd SIGIR Conference. 2020. P. 39–48. URL: https://arxiv.org/abs/2004.12832.

25. Gao L., Ma X., Lin J., Callan J. Precise zero-shot dense retrieval without relevance labels (HyDE). Proceedings of ACL 2023. Toronto, 2023. P. 1762–1777. URL: https://aclanthology.org/2023.acl-long.99/.

26. Asai A., Wu Z., Wang Y., Sil A., Hajishirzi H. Self-RAG: learning to retrieve, generate, and critique through self-reflection. Proceedings of ICLR 2024. Vienna, 2024. 30 p. URL: https://arxiv.org/abs/2310.11511.

27. Gao Y., Xiong Y., Gao X., Jia K., Pan J., Bi Y., Dai Y., Sun J., Wang H. Retrieval-augmented generation for large language models: a survey. arXiv preprint arXiv:2312.10997. 2024. 21 p. URL: https://arxiv.org/abs/2312.10997.

28. Edge D., Trinh H., Cheng N. et al. From local to global: a graph RAG approach to query-focused summarization. arXiv preprint arXiv:2404.16130. 2024. 35 p. URL: https://arxiv.org/abs/2404.16130.

29. Chen J., Xiao S., Zhang P., Luo K., Lian D., Liu Z. M3-Embedding: multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. arXiv preprint arXiv:2402.03216. 2024. 20 p. URL: https://arxiv.org/abs/2402.03216.

30. LangChain documentation. LangChain Inc., 2025. URL: https://python.langchain.com/docs/ (дата звернення: 16.04.2026).

31. LlamaIndex documentation. LlamaIndex Inc., 2025. URL: https://docs.llamaindex.ai/ (дата звернення: 16.04.2026).

32. Chroma documentation. Chroma DB, 2025. URL: https://docs.trychroma.com/ (дата звернення: 16.04.2026).

33. Qdrant documentation. Qdrant, 2025. URL: https://qdrant.tech/documentation/ (дата звернення: 16.04.2026).

34. FastAPI documentation. Tiangolo, 2025. URL: https://fastapi.tiangolo.com/ (дата звернення: 17.04.2026).

35. Streamlit documentation. Snowflake, 2025. URL: https://docs.streamlit.io/ (дата звернення: 17.04.2026).

36. Литвин В. В., Пелещак Р. М., Висоцька В. А. Глибинне навчання: навчальний посібник. Львів: Видавництво Львівської політехніки, 2021. 261 с.

37. Шаховська Н. Б., Болюбаш Ю. Я., Верес О. М. Системи штучного інтелекту: навчальний посібник. Львів: Видавництво Львівської політехніки, 2018. 244 с.

38. Литвин В. В., Пасічник В. В., Яцишин Ю. В. Інтелектуальні системи: підручник. Львів: Новий Світ-2000, 2011. 405 с. ISBN 978-966-418-086-0.

39. Izacard G., Lewis P., Lomeli M., Hosseini L., Petroni F., Schick T., Dwivedi-Yu J., Joulin A., Riedel S., Grave E. Atlas: few-shot learning with retrieval augmented language models. Journal of Machine Learning Research. 2023. Vol. 24. P. 1–43. URL: https://www.jmlr.org/papers/v24/23-0037.html.

40. Шкіцька І. Ю. Основи академічної доброчесності: практикум: навчально-методичний посібник для студентів вищих навчальних закладів. Тернопіль: ТНЕУ, 2018. 64 с.

41. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ: ДП «УкрНДНЦ», 2016. 25 с.

42. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ: ДП «УкрНДНЦ», 2017. 16 с.

43. Шевчук В. О., Коваль В. С. Інтелектуальна інформаційна система семантичного пошуку з використанням технології Retrieval-Augmented Generation. Матеріали міжнародної мультидисциплінарної інтернет-конференції на тему «Світ наукових досліджень», випуск 53, секція Інформаційні системи і технології, Вища школа управління та адміністрації в Ополе, м. Ополе, Польща, 18-19 червня 2026 р. URL: https://www.economy-confer.com.ua/full-article/6935/

44. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп'яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

Додаток А

Лістинг програмного коду

Лістинг А.1

```
def hybrid_search(self, query: str, top_k: int = 5) -> list[Chunk]:
    """Виконує гібридний пошук з перерейтингом та повертає топ-k фрагментів."""
    # Етап 1: щільний пошук
    query_vector = self.embedder.encode_query(query)
    dense_hits = self.vector_store.search(
        vector=query_vector,
        k=self.config.dense_k,
    )

    # Етап 2: розріджений пошук BM25
    sparse_hits = self.bm25.search(query, k=self.config.sparse_k)

    # Етап 3: гібридне злиття Reciprocal Rank Fusion
    fused = self._reciprocal_rank_fusion(
        rankings=[dense_hits, sparse_hits],
        k_constant=60,
    )[: self.config.hybrid_k]

    # Етап 4: перерейтинг крос-енкодером
    rerank_pairs = [(query, chunk.text) for chunk in fused]
    scores = self.reranker.score(rerank_pairs)
    reranked = sorted(
        zip(fused, scores), key=lambda pair: pair[1], reverse=True
    )

    return [chunk for chunk, _ in reranked[:top_k]]
```

Лістинг А.2

```
[Джерело 1: docs/setup.md, розділ "Конфігурація мережі"]
<текст фрагмента 1>

[Джерело 2: docs/troubleshoot.md, розділ "Помилки з'єднання"]
<текст фрагмента 2>
```

Лістинг А.3

```
class RecursiveTextChunker:
    """Рекурсивне розбиття тексту з токен-орієнтованим контролем розміру."""

    def __init__(self, max_tokens: int = 512, overlap_tokens: int = 64,
                 tokenizer_name: str = "cl100k_base") -> None:
        self.max_tokens = max_tokens
        self.overlap_tokens = overlap_tokens
        self._encoder = tiktoken.get_encoding(tokenizer_name)
```

```
        self._separators = ["\n\n", "\n", ". ", "? ", "! ", "; ", ", ", " ", ""]

    def split(self, text: str, metadata: dict) -> list[Chunk]:
        """Розбиває текст і повертає перелік фрагментів із метаданими."""
        raw_chunks = self._recursive_split(text, self._separators)
        chunks = self._merge_with_overlap(raw_chunks)
        return [
            Chunk(text=c, metadata={**metadata, "chunk_index": i})
            for i, c in enumerate(chunks)
        ]

    def _token_len(self, text: str) -> int:
        return len(self._encoder.encode(text))
```

Лістинг А.4

```
class E5EmbeddingModel:
    """Інкапсуляція мультимовної моделі multilingual-e5-base."""

    def __init__(self, model_name: str = "intfloat/multilingual-e5-base",
                 device: str = "cpu", batch_size: int = 32) -> None:
        self._model = SentenceTransformer(model_name, device=device)
        self._batch_size = batch_size
        self.dimension = self._model.get_sentence_embedding_dimension()

    def encode_passages(self, texts: list[str]) -> np.ndarray:
        prefixed = [f"passage: {t}" for t in texts]
        return self._model.encode(
            prefixed, batch_size=self._batch_size,
            normalize_embeddings=True, show_progress_bar=False,
        )

    def encode_query(self, text: str) -> np.ndarray:
        return self._model.encode(
            f"query: {text}", normalize_embeddings=True,
        )
```

Лістинг А.5

```
def answer(self, question: str) -> RAGResponse:
    """Опрацьовує запит користувача та повертає відповідь з джерелами."""
    started = time.perf_counter()

    # 1. Гібридний пошук з перерейтингом
    chunks = self.retriever.hybrid_search(question, top_k=self.config.top_k)

    # 2. Формування контексту з контролем бюджету токенів
    context = self.prompt_builder.build_context(
        chunks=chunks, token_budget=self.config.context_budget,
    )

    # 3. Побудова підказки та виклик мовної моделі
    prompt = self.prompt_builder.build_prompt(question, context)
```

```
answer_text = self.llm.generate(
    prompt, temperature=self.config.temperature,
    max_tokens=self.config.max_answer_tokens,
)

# 4. Постобробка: вилучення посилань на джерела
sources = self._collect_sources(chunks)

elapsed_ms = (time.perf_counter() - started) * 1000
response = RAGResponse(
    answer=answer_text, sources=sources, latency_ms=elapsed_ms,
    chunks=chunks,
)

# 5. Журналювання у реляційну базу
self.db.log_interaction(question, response)
return response
```

Додаток Б

Копія публікації

ІНТЕЛЕКТУАЛЬНА ІНФОРМАЦІЙНА СИСТЕМА СЕМАНТИЧНОГО ПОШУКУ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ RETRIEVAL-AUGMENTED GENERATION

12.06.2026 20:35

Автор: **Шевчук Віталій Олександрович, студент, Західноукраїнський національний університет**

[2. Інформаційні системи і технології;]

Сучасні підприємства накопичують значні обсяги неструктурованої текстової інформації — технічну документацію, регламенти, інструкції з налаштування, бази знань служб підтримки. Частка неструктурованих даних у загальному інформаційному фонді організацій сягає 80 %, а їхній обсяг подвоюється кожні два-три роки. Класичні системи пошуку на основі ключових слів (TF-IDF, BM25) не враховують семантики запиту, чутливі до синонімії й різних формулювань. Великі мовні моделі (LLM) формують природномовну відповідь, але обмежені актуальністю тренувальних даних, не мають доступу до приватної корпоративної інформації та схильні до галюцинацій. Технологія Retrieval-Augmented Generation (RAG) поєднує переваги обох підходів і є одним із провідних архітектурних шаблонів сучасних інтелектуальних інформаційних систем.

Метою роботи є підвищення ефективності пошуку інформації у корпоративних базах знань шляхом розроблення інтелектуальної інформаційної системи семантичного пошуку на основі технології Retrieval-Augmented Generation, що формує точну й контекстно обґрунтовану відповідь природною мовою з підтвердженням джерелами. Об'єктом дослідження є процес семантичного пошуку інформації у корпоративній базі знань, предметом — методи та програмні засоби побудови RAG-систем на основі поєднання щільних векторних представлень, наближеного пошуку найближчих сусідів та великих мовних моделей.

Архітектуру системи побудовано за принципом багаторівневого поділу на компоненти із чітко визначеною зоною відповідальності (рис. 1). Виокремлено шість рівнів: подання (Streamlit UI, FastAPI REST), оркестрація (RAG-конвеєр), інтелектуальні служби (гібридний пошук, перерейтинг, генерування відповіді), моделі (E5, BGE, LLM), підготовка даних і сховище (Chroma + SQLite). Багаторівневий поділ забезпечує слабку зв'язність компонентів та можливість заміни моделей без модифікації решти системи.

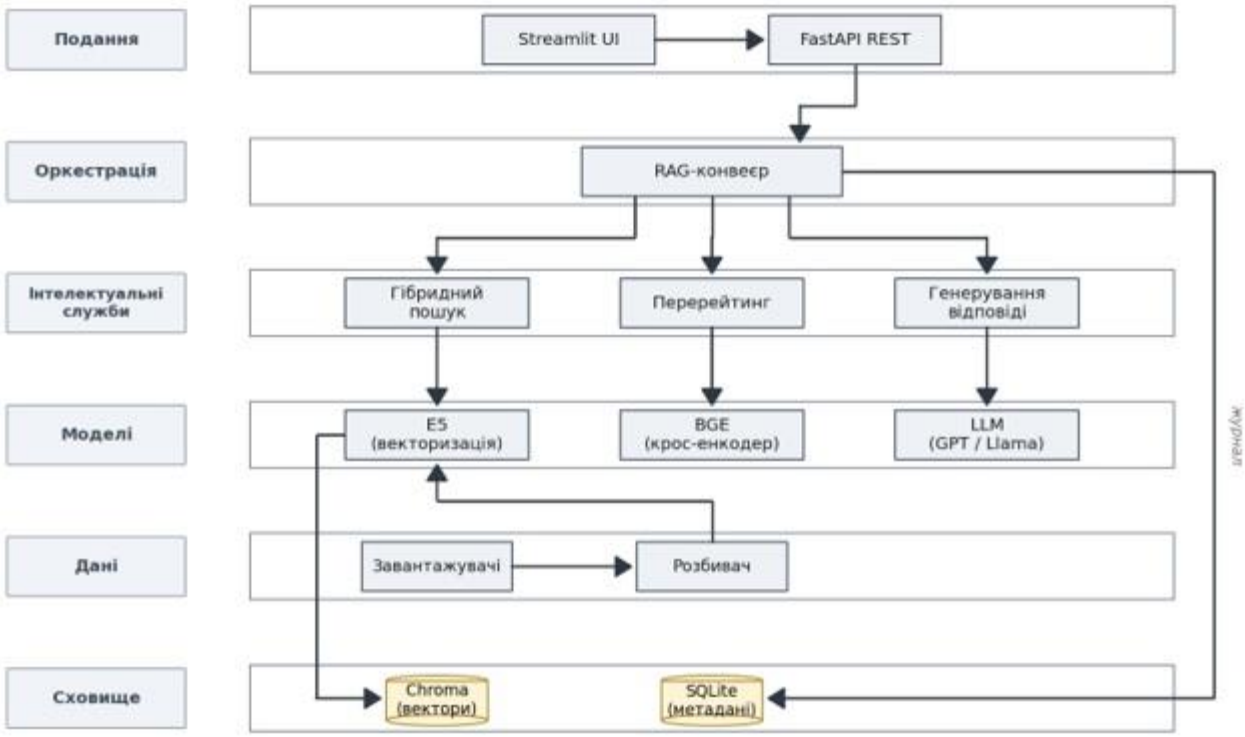

Рисунок 1 — Структурна схема програмного засобу семантичного пошуку

Конвеєр обслуговування запиту реалізує гібридний пошук, що поєднує щільний (семантичний) та розріджений (BM25) способи. Запит користувача перетворюється на 768-вимірний вектор мультимовною моделлю multilingual-e5-base, після чого виконується пошук топ-20 кандидатів у Chroma за косинусною подібністю. Паралельно виконується BM25-пошук на словесному індексі. Результати об'єднуються методом Reciprocal Rank Fusion (k = 60). Об'єднаний перелік перевпорядковує крос-енкодерна модель BAAI/bge-reranker-base, що дозволяє точніше оцінити релевантність пари «запит — фрагмент». Фінальні топ-5 фрагментів об'єднуються у контекст для великої мовної моделі (gpt-4o-mini або локальної Llama 3.1 8B) разом із шаблоном підказки, що містить інструкцію відповідати лише на основі наданого контексту.

Експериментальне дослідження проведено на тестовому корпусі з 412 документів технічної документації (8 654 фрагменти) та 80 запитів з еталонними відповідями. Порівняно п'ять конфігурацій пошукової складової за метриками Recall@k, MRR і NDCG@10 (таблиця 1).

Таблиця 1 — Результати порівняння конфігурацій пошуку

Конфігурація	Recall@5	Recall@10	MRR	NDCG@10
BM25	0,614	0,728	0,521	0,597
Dense (E5)	0,762	0,841	0,667	0,724
Hybrid (RRF)	0,824	0,891	0,719	0,783
Hybrid + Rerank	0,876	0,914	0,803	0,841
Hybrid + Rerank + HyDE	0,889	0,927	0,811	0,853

Робочою обрано конфігурацію «Hybrid + Rerank», що забезпечує найкраще співвідношення якості та продуктивності: Recall@5 = 87,6 %, MRR = 80,3 %, NDCG@10 = 84,1 %. Перехід від BM25 до щільного пошуку дав приріст Recall@5 на 14,8 п. п., додавання гібридного злиття — ще на 6,2 п. п., перерейтинг кросенкодером — ще на 5,2 п. п. Додавання трансформації запиту HyDE дає лише 1,3 п. п. ціною подвоєння часу відгуку, що не виправдано для прийнятої постановки задачі.

Якість генеративної складової оцінено за метриками faithfulness та answer relevancy засобами бібліотеки ragas для двох мовних моделей. Хмарна gpt-4o-mini забезпечує faithfulness 92,7 % за середній час відгуку 1,84 с; локальна Llama-3.1-8B-Instruct — 86,1 % за 8,72 с на ЦП або 1,32 с на графічному процесорі. Локальна модель забезпечує цілковиту приватність корпоративних даних і є придатною для організацій з підвищеними вимогами до конфіденційності.

Програмний засіб реалізовано мовою Python 3.13 у вигляді 12 модулів із застосуванням бібліотек LangChain, sentence-transformers, Chroma, rank_bm25, FlagEmbedding, FastAPI, Streamlit та ragas. Розроблено вебінтерфейс із трьома сторінками (пошук, корпус, аналітика) та REST API з вісьмома кінцевими точками. Проведено модульне (64 тести), інтеграційне (18 тестів) та функціональне (80 тестів) тестування з показниками 100 %, 100 % та 91,3 % успіху відповідно.

Розроблений програмний засіб скорочує час пошуку інформації за рахунок розуміння системою сенсу запиту, а не лише ключових слів, зменшує навантаження на службу підтримки шляхом автоматизованих відповідей із посиланням на першоджерело та знижує ризик застосування недостовірних відомостей завдяки прив'язці кожної відповіді до конкретних фрагментів документів. На відміну від комерційних SaaS-платформ, засіб підтримує повністю локальне розгортання та може застосовуватися організаціями з підвищеними

вимогами до приватності даних. Перспективними напрямами вдосконалення є впровадження агентного RAG, рекурсивного пошуку з самоперевіркою достовірності, інтеграція мультимодальних моделей та доучування моделі векторизації на доменній термінології організації.

Література

1. Lewis P., Perez E., Piktus A. et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 9459–9474.
2. Wang L., Yang N., Huang X. et al. Multilingual E5 text embeddings: a technical report. arXiv preprint arXiv:2402.05672. 2024. 19 p.
3. Malkov Y. A., Yashunin D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2020. Vol. 42, № 4. P. 824–836.
4. Robertson S., Zaragoza H. The probabilistic relevance framework: BM25 and beyond. Foundations and Trends in Information Retrieval. 2009. Vol. 3, № 4. P. 333–389.
5. Cormack G. V., Clarke C. L. A., Buettcher S. Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. Proceedings of the 32nd SIGIR Conference. Boston, 2009. P. 758–759.
6. Gao Y., Xiong Y., Gao X. et al. Retrieval-augmented generation for large language models: a survey. arXiv preprint arXiv:2312.10997. 2024. 21 p.