Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

До захисту допущено
Кафедрою комп'ютерних систем та робототехніки
протокол № __ від __ грудня 2025р.

завідувач кафедри _____ Максим ХРУСЛОВ
(підпис)

«___»_____ 2025 р.

Кваліфікаційна робота

здобувача другого (магістерського) рівня вищої освіти

«Мультимодальна платформа для аналізу та пошуку даних на основі RAG-архітектури»

Спеціальність ***123 - Комп'ютерна інженерія.***
Освітня програма ***Комп'ютерна інженерія***

Виконавець  ***Євген Філіп'єв***
(підпис)

Науковий керівник  ***Ніна БАКУМЕНКО***
(підпис)

Харків - 2025

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи магістр складається зі вступу, трьох розділів, висновків, списку використаних джерел і трьох додатків. Загальний обсяг роботи складає 98 сторінки, із яких 71 сторінок основної частини з 19 рисунками, 21 таблицею, 45 найменувань списку використаних джерел та трьома додатками.

Метою магістерської роботи - підвищення якості інформаційного пошуку в неструктурованих текстових масивах шляхом застосування як готових, так розробки нових методів інформаційного пошуку, які допоможуть подолати лексичний розрив між запитом та релевантними документами.

Об'єкт дослідження - процеси семантичного пошуку та генерації відповідей у системах доповненої генерації на основі великих мовних моделей.

Предмет магістерської роботи - методи та алгоритми гібридного семантичного пошуку документів із застосуванням генерації гіпотетичних документів та механізму пізньої взаємодії ColBERT.

Область застосування - інтелектуальні системи пошуку та аналізу текстових даних. Розроблений підхід може бути інтегрований у корпоративні системи управління знаннями, платформи технічної підтримки з автоматизованими відповідями, освітні середовища з адаптивним контентом, а також використаний у дослідницьких проектах з обробки природної мови.

Ключові слова: RAG, Retrieval-Augmented Generation, генеративні мовні моделі, семантичний пошук, BM25, Dense, ColBERT, HyDE, гіпотетичні документи, мікросервісна архітектура

ABSTRACT

The explanatory note to the master's thesis consists of an introduction, three chapters, conclusions, a list of references, and three appendices. The total volume of the work is 98 pages, of which 71 pages are the main part with 19 figures, 21 tables, 45 references, and three appendices.

The purpose of the qualification work is to develop a multimodal platform for data analysis and retrieval based on RAG architecture.

The object of the research is the processes of semantic search and response generation in retrieval-augmented generation systems based on large language models.

The subject of the research is the methods and algorithms of hybrid semantic document retrieval that combine Dense and Sparse approaches with HyDE hypothetical document generation and ColBERT late interaction mechanism to improve result relevance in RAG systems.

The problem addressed in the thesis is to improve information retrieval quality by transitioning from homogeneous methods to hybrid approaches using hypothetical document generation and multi-level ranking. This makes it possible to bridge the semantic gap between user queries and relevant documents, reduce lexical mismatch impact, and improve search precision and recall.

Scope - intelligent text data search and analysis systems. The developed approach can be integrated into corporate knowledge management systems, technical support platforms with automated responses, educational environments with adaptive content, and used in natural language processing projects.

Keywords: RAG, Retrieval-Augmented Generation, generative language models, semantic search, ColBERT, late interaction, BM25, Dense, HyDE, hypothetical documents, microservice architecture.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

API - програмний інтерфейс застосунків (Application Programming Interface);

ASGI - інтерфейс шлюзу асинхронного сервера (Asynchronous Server Gateway Interface);

BEIR - бенчмарк для оцінювання інформаційного пошуку (Benchmarking IR);

BGE - двонаправлені вбудовування для узагальненого пошуку (Bidirectional Generative Embeddings);

BM25 - алгоритм ранжування Best Matching 25;

CLI - інтерфейс командного рядка (Command Line Interface);

ColBERT - контекстуалізована пізня взаємодія для BERT (Contextualized Late Interaction over BERT);

CPU - центральний процесор (Central Processing Unit);

DCG - дисконтований кумулятивний приріст (Discounted Cumulative Gain);

Docker - платформа контейнеризації програмного забезпечення;

GPU - графічний процесор (Graphics Processing Unit);

HyDE - гіпотетичні вбудовування документів (Hypothetical Document Embeddings);

IDF - обернена частота документа (Inverse Document Frequency);

JSON - об’єктна нотація JavaScript (JavaScript Object Notation);

JWT - JSON веб-токен (JSON Web Token);

LLM - велика мовна модель (Large Language Model);

MAP - середня точність (Mean Average Precision);

MaxSim - операція максимальної подібності (Maximum Similarity);

MinIO - об’єктне сховище з відкритим кодом;

MRR - середній обернений ранг (Mean Reciprocal Rank);

NDCG - нормалізований дисконтований кумулятивний приріст (Normalized Discounted Cumulative Gain);

NLP - обробка природної мови (Natural Language Processing);

PQ - квантування продукту (Product Quantization);

RAG - генерація з доповненням на основі пошуку (Retrieval-Augmented Generation);

REST - передача репрезентативного стану (Representational State Transfer);

RRF - взаємне ранжування злиття (Reciprocal Rank Fusion);

TF - частота терміна (Term Frequency);

URL - уніфікований локатор ресурсу (Uniform Resource Locator);

ВСТУП

Стрімка цифровізація усіх сфер діяльності супроводжується зростанням обсягів неструктурованої текстової інформації у геометричній прогресії. Традиційні пошукові системи демонструють обмежені можливості та загалом не ефективними в наданні релевантної інформації, адже вони не здатні інтерпретувати семантичний контекст запитів користувачів і надати найбільш актуальну інформацію.

Актуальність роботи. Великі мовні моделі (LLM) є однією з технологій, покликаних вирішити дану проблему. Проте LLM страждають від фундаментальної проблеми галюцинацій - генерації правдоподібних, але фактично некоректних відповідей, необхідності у постійному перенавчанні на нових масивах даних.

Для вирішення цієї проблеми була запропонована архітектура Retrieval-Augmented Generation (RAG). Даний архітектурний патерн усуває зазначені недоліки шляхом інтеграції етапу пошуку релевантних документів перед генерацією відповіді, що забезпечує верифікованість результатів через посилання на першоджерела.

Метою магістерської роботи є підвищення якості інформаційного пошуку в неструктурованих текстових масивах шляхом застосування як готових, так розробки нових методів інформаційного пошуку, які допоможуть подолати лексичний розрив між запитом та релевантними документами.

Об'єкт магістерської роботи - процеси семантичного пошуку та генерації відповідей у системах доповненої генерації на основі великих мовних моделей, їх взаємозв'язок з якістю кінцевих результатів.

Методи роботи: методи та алгоритми інформаційного семантичного пошуку документів, методи глибокого навчання, методи статистичного аналізу якості пошуку.

Предмет магістерської роботи - методи та алгоритми гібридного семантичного пошуку документів із застосуванням генерації гіпотетичних документів та механізму пізньої взаємодії ColBERT.

Завдання магістерської роботи:

1. Виконати аналіз існуючих RAG систем та методів семантичного пошук.
2. Обґрунтувати та спроектувати архітектуру мультимодальної системи.
3. Розробити метод інформаційного пошуку для подолання лексичного розриву між запитом та документами.
4. Реалізувати програмні компоненти RAG системи з запропонованими підтримкою методів BM25, Dense, Sparse та HyDE-ColBERT.
5. Провести експериментальне тестування на бенчмарках BEIR.
6. Виконати оцінку якості інформаційного пошуку на датасетах з різних предметних областей.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНИХ МЕТОДІВ RAG-СИСТЕМ

1.1. Архітектура Retrieval-Augmented Generation

1.1.1. Еволюція підходів до інформаційного пошуку

Історія автоматизованого пошуку інформації налічує понад п'ятдесят років, протягом яких підходи до вилучення релевантних документів зазнали кардинальних змін. Перші системи 1960-х років працювали на засадах булевої логіки: текст або повністю відповідає критеріям пошуку, або відкидався - жодних проміжних варіантів не передбачалось. Оператори AND, OR, NOT давали змогу будувати складні умови відбору, однак такий бінарний підхід виявився надто жорстким для практичних потреб - документи із частковим співпадінням просто губились у масиві результатів [7].

Ситуацію змінила векторно-просторова модель, сформульована Джерардом Солтоном у рамках проєкту SMART у 1970-х роках [7]. Згідно з цією концепцією і документи, і запити проектуються як точки багатовимірного простору, де кожен вимір відповідає окремому терміну словника. Відстань між точками визначає ступінь релевантності - чим ближче документ до запиту, тим вищою є його відповідність. Для зважування термінів було запропоновано функцію TF-IDF, що враховує як локальну частоту слова в документі, так і його розрізнювальну здатність у межах колекції:

$$TF - IDF(t,d) = TF(t,d) \times log\left(\frac{N}{DF(t)}\right) \quad (1.1)$$

де $TF(t,d)$ - частота терміну t у документі d; N - загальна кількість документів у колекції; $DF(t)$ - кількість документів, що містять термін t.

Стохастична парадигма 1970–80-х років запропонувала радикально інший кут зору: замість геометричних метрик почали обчислювати

імовірність релевантності документа заданому запиту. Ключовою ідеєю стало припущення, що переважна більшість документів у будь-якій колекції не відповідає конкретному запиту, а отже, під час обчислень цими випадками можна знехтувати - саме це спрощення дозволило перейти від складної теоретичної формули до практично застосовного алгоритму.

Найвідомішим представником цієї родини стала функція ранжування BM25 (Best Match 25) [7], вперше запроваджена у пошуковій системі Okapi Лондонського міського університету, звідки й походить альтернативна назва Okapi BM25. Алгоритм враховує наскільки часто з'являється слово у документі, щоб уникнути несправедливої переваги довгих текстів проводиться нормалізація оцінки за довжиною документа та де-факто стандартом лексичного пошуку й сьогодні.

Справжній переворот у галузі інформаційного пошуку стався з появою нейромережевого кодування. Word2Vec, запропонований Томасом Міколовим та командою Google у 2013 році, який продемонстрував, що слова можна представляти щільними векторами фіксованої розмірності, здатними відображати семантичні зв'язки [8]. На відміну від попередніх підходів, де кожне слово кодувалося ізольованим one-hot вектором, Word2Vec навчався на великих корпусах тексту та має можливість зберігали інформацію про контекст в якому вживаються слова.

Фундамент сучасних систем обробки природної мови було закладено у 2017 році, коли була опублікована стаття «Attention Is All You Need» [9]. Запропонована архітектура Transformer відмовилася від рекурентних та згорткових нейронних мереж, які домінували в NLP протягом попереднього десятиліття, на користь механізму самоуваги (self-attention). Ключова інновація полягала у здатності моделі одночасно аналізувати всі позиції вхідної послідовності, встановлюючи зв'язки між словами незалежно від відстані між ними. Це усунуло проблему затухання градієнтів при обробці

довгих текстів та забезпечило ефективне паралельне обчислення на графічних процесорах.

Архітектура Transformer стала основою для нового покоління мовних моделей. Паралельно Google представив BERT, який використовував двонаправлене кодування для глибшого розуміння контексту.

Водночас практичне застосування великих мовних моделей (LLM) виявило серйозну проблему, що отримала назву «галюцинації». Моделі впевнено видають правдоподібну, але фактично неправильну інформацію - вигадані цитати, неіснуючі факти, хибні комбінації реальних даних. Природа галюцинацій криється в самій архітектурі LLM: моделі оптимізовані на передбачення наступного токена, а не на перевірку фактів. Вони не здатні відрізнити власні знання від вигадок і демонструють однакову впевненість як у правильних, так і в помилкових відповідях.

Додатковим обмеженням виступає статичність знань моделі, закладених на етапі навчання. Великі мовні моделі не мають доступу до актуальної інформації, оскільки їхні параметри не змінюються після завершення тренування, а перенавчання на свіжих даних потребує великих обчислювальних ресурсів.

У відповідь на зазначені виклики команда Meta AI у 2020 році запропонувала архітектурний патерн Retrieval-Augmented Generation (RAG) [1]. Ключова ідея підходу полягає в інтеграції механізму пошуку релевантної інформації з процесом генерації відповіді. Замість покладання виключно на внутрішні параметри моделі, система спочатку ідентифікує документи з зовнішньої бази знань, які містять відомості, необхідні для формування коректної відповіді. Це дозволяє суттєво знизити рівень галюцинацій та забезпечити доступ до актуальної інформації без перенавчання моделі.

Рисунок 1.1 - Хронологія розвитку методів інформаційного пошуку.

Кожна епоха розвитку інформаційного пошуку принесла свій внесок у формування сучасної парадигми. Булева модель заклала формальну основу зіставлення запитів із документами через логічні оператори. Векторна модель увела градуйоване ранжування, відійшовши від бінарної релевантності до числових оцінок схожості. Імовірнісна модель забезпечила статистичну обґрунтованість ранжування на базі теорії ймовірностей. Нейромережева парадигма вивела пошук на принципово новий рівень - семантичне розуміння на рівні смислу, а не лише поверхневого збігу слів [8].

1.1.2. Концептуальна модель RAG системи

Архітектура RAG реалізує послідовність із трьох взаємопов'язаних етапів обробки запитів, де якість кожної стадії безпосередньо впливає на кінцевий результат. Принципова відмінність від класичних генеративних систем полягає у динамічному доповненні контекстного вікна моделі релевантною інформацією безпосередньо в момент формування відповіді. Концептуальна модель системи, що ілюструє взаємодію основних компонентів, представлена на (рис. 1.2).

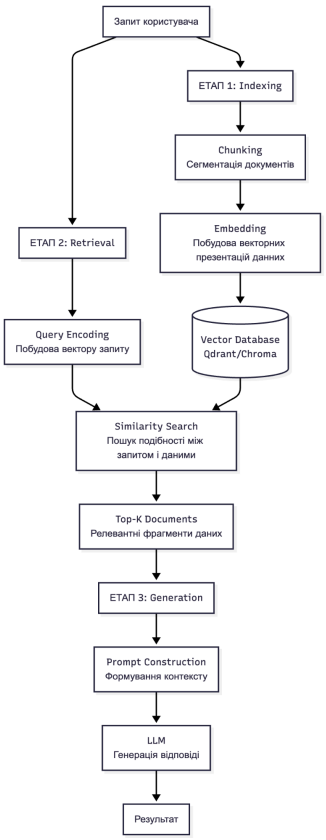

Рисунок 1.2 - Концептуальна модель архітектури RAG.

Перший етап - індексування (Indexing) - передбачає попередню підготовку корпусу документів для забезпечення ефективного семантичного пошуку. Вхідні дані ділиться на семантично завершені фрагменти (chunks) оптимального розміру. Вибір стратегії сегментації має критичне значення: розмір фрагмента суттєво впливає на якість пошуку [2]. Занадто малі фрагменти втрачають контекст, тоді як завеликі знижують точність, оскільки можуть містити інформацію з різних тематичних областей одночасно.

Оптимальний розмір фрагмента визначається формулою:

$$chunk\_size_{opt} = arg \max_{s} (\alpha \cdot Recall(s) + \beta \cdot Precision(s)) \quad (1.2)$$

де s - розмір фрагмента в токенах; α, β - вагові коефіцієнти (типово $\alpha = \beta = 0.5$); $Recall(s)$ - повнота пошуку при розмірі s; $Precision(s)$ - точність пошуку при розмірі s.

Після сегментації кожен фрагмент даних трансформується у векторне представлення (embedding) фіксованої розмірності за допомогою спеціалізованих нейромережевих моделей. Отримані вектори індексуються у векторній базі даних для швидкого пошуку релевантної інформації.

Другий етап - пошук (Retrieval) - активується при надходженні запиту користувача. Запит користувача кодується тим самим енкодером, що використовувався для індексування документів, що дозволяє забезпечити порівняння векторів у єдиному семантичному просторі, яке відбувається за рахунок розрахунку косинусної подібності:

$$sim(q, d) = \frac{\vec{E}_q \cdot \vec{E}_d}{\|\vec{E}_q\| \cdot \|\vec{E}_d\|} \quad (1.3)$$

де $\vec{E}_q$ - векторне представлення запиту; $\vec{E}_d$ - векторне представлення документа; $sim(q, d)$ - міра семантичної близькості в діапазоні $[-1, 1]$.

Значення косинусної подібності, близьке до 1, свідчить про високу семантичну подібність між запитом користувача та документом, тоді як значення близьке до 0 вказує на відсутність смислового зв'язку.

Результатом пошуку є впорядкований за релевантністю перелік з k найбільш відповідних фрагментів документів ($Top\text{-}K\ results$).

Третій етап - генерація (Generation) - передбачає формування розгорнутої відповіді на основі отриманого контексту. Витягнуті документи конкатенуються з оригінальним запитом у структурований промпт, який подається на вхід великої мовної моделі. Процес формування контексту (Prompt Construction) є одним з найвідповідальніших кроків, від якості якого

залежить здатність LLM коректно інтерпретувати надану інформацію та згенерувати релевантну відповідь.

Генеративна модель синтезує відповідь, спираючись виключно на надану в промпті інформацію, що може суттєво мінімізувати ризик галюцинацій. Додатковою перевагою є можливість верифікації згенерованого контенту через посилання на першоджерело.

1.1.3. Переваги RAG над класичними LLM

Порівняльний аналіз архітектурних підходів демонструє низку суттєвих переваг RAG-систем перед автономними генеративними моделями (табл. 1.1).

Таблиця 1.1

Порівняння RAG-підходу та класичних LLM

Критерій	Класичні LLM	RAG системи
Актуальність даних	Обмежена датою навчання	Динамічне оновлення корпусу
Фактологічна точність	85-95%	95-99% з верифікацією
Прозорість	«Чорний ящик»	Посилання на джерела
Масштабованість знань	Потребує перенавчання	Додавання документів
Обчислювальні ресурси	Значні	Помірні
Контроль доступу	Відсутній	На рівні документів

Ключовою перевагою RAG підходу є можливість оперативного оновлення бази знань без необхідності повторного навчання моделі. Класична LLM потребує тижнів обчислень та значних фінансових витрат для інтеграції нової інформації через процедуру fine-tuning. RAG натомість дозволяє додавати нові дані або видаляти застарілі через просту модифікацію векторного індексу. Для корпоративного середовища, де інформація постійно змінюється, ця властивість набуває критичного значення.

Фактологічна точність RAG-систем суттєво перевищує показники автономних LLM. Класичні моделі демонструють точність на рівні 85-95%,

що пояснюється феноменом «галюцинацій» - генерації правдоподібного, але фактично некоректного контенту. RAG із верифікацією через першоджерела досягає показників 95-99%, оскільки генеративна модель не покладається виключно на параметричну пам'ять, а отримує релевантний контекст безпосередньо з документів. Це суттєво знижує ризик генерації недостовірної інформації.

Можливість самостійно керувати процесом донавчання забезпечує прозорість RAG підходу, де натомість LLM функціонують більше як «чорний ящик» - користувач отримує відповідь без будь-якого розуміння їх походження чи обґрунтування. RAG системи зберігають явний зв'язок між наданою відповіддю та конкретними фрагментами даних. Внаслідок цього користувач бачить не лише кінцевий результат, а й перелік першоджерел із зазначенням релевантних пасажів для самостійної верифікації інформації.

Контроль доступу в RAG системах реалізується на рівні документів: архітектура дозволяє обмежувати доступ до конфіденційної інформації на основі ролей користувачів, їхніх повноважень чи організаційної належності. Це принципово неможливо при використанні монолітних LLM, де всі знання знаходяться по замовчуванню як параметри в моделі без можливості їх сегрегації чи вибіркового приховування. Для бізнесу, що оперують чутливими даними, ця властивість визначає саму можливість впровадження інтелектуальних систем.

Обчислювальна ефективність RAG-систем також заслуговує на увагу. Хоча архітектура передбачає додатковий етап пошуку, загальні витрати ресурсів виявляються меншими порівняно з використанням надвеликих LLM. Компактна генеративна модель у поєднанні з ефективним векторним пошуком забезпечує кращий час відгуку та нижчу вартість.

Зазначені переваги роблять RAG-архітектуру оптимальним вибором для побудови корпоративних інформаційно-пошукових систем, де

критичними є актуальність даних, їх достовірність, прозорість прийняття рішень та контрольований доступ до інформації різного рівня конфіденційності.

1.2. Методи семантичного пошуку документів

Ефективність RAG систем безпосередньо визначається якістю етапу пошуку релевантних даних. Сучасні підходи до семантичного пошуку прийнято класифікувати за трьома основними категоріями. Перша - dense retrieval, або щільний пошук, що базується на нейромережевих ембедінгах. Друга - sparse retrieval, розріджений пошук із використанням лексичного зіставлення. Третя категорія охоплює різноманітні гібридні підході, які поєднують в собі найкраще від обох попередніх підходів [11].

Кожна з цих підходів має власну специфіку, а саме відмінні механізми представлення тексту у векторному просторі, різні алгоритми ранжування кандидатів та унікальний профіль компромісів між точністю результатів і швидкодією системи. Вибір конкретного методу залежить від особливостей предметної області, обсягу корпусу даних та вимог до часу відгуку.

1.2.1. Лексичний пошук BM25

Попри значний прогрес dense-методів на основі нейронних мереж, лексичний пошук за точними збігами слів не втрачає своєї практичної цінності. BM25 (Best Matching 25) - це класичний алгоритм розріджених представлень (sparse retrieval), який за все ще залишається стандартом для більшості пошукових систем. Ключовою особливістю алгоритму є те, що він враховує статистичних характеристик входження термінів, а саме наскільки часто він з’являється в документі та розподілу по колекції загалом.

Формула оцінки релевантності документа d для запиту q має вигляд:

$$BM25(q,d) = \sum_{t \in q} IDF(t) \cdot \frac{tf(t,d) \cdot (k_1 + 1)}{tf(t,d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{avgdl}\right)} \quad (1.4)$$

де $tf(t,d)$ - частота терміна t у документі d; $|d|$ - довжина документа в словах; $avgdl$ - середня довжина документа в колекції; k_1, b - параметри налаштування (типово $k_1 = 1.5, b = 0.75$).

Формула складається з двох ключових компонентів. IDF (Inverse Document Frequency) - надає більшу вагу рідкісним термінам, які краще розрізняють документи між собою. Обчислюється за формулою:

$$IDF(t) = log \frac{N - n(t) + 0.5}{n(t) + 0.5} \quad (1.5)$$

де N - загальна кількість документів у колекції; $n(t)$ - кількість документів, що містять терм t.

Другий компонент відповідає за насичення частоти терміна: кожне наступне входження слова додає дедалі меншу вагу до загальної оцінки. Це запобігає надмірному впливу документів із штучно завищеною частотою окремих слів.

Завдяки використанню інвертованого індексу, BM25 забезпечує високу швидкість точності для запитів за ключовими словами. В той же час алгоритм має суттєві обмеження. BM25 не здатний враховувати семантичну подібність між різними частинами даних, спостерігається висока чутливість до формулювання запиту та проблема vocabulary mismatch, коли релевантні документи не знаходяться через розбіжність у термінології.

1.2.2. Bi-encoder підхід до dense retrieval

Dense retrieval принципово відрізняється від класичних методів пошуку тим, що представляє тексти у вигляді щільних векторів (sparse retrieval) фіксованої розмірності. На відміну від розріджених (dense) представлень

типу TF-IDF, де переважна більшість елементів дорівнює нулю, dense ембедінги заповнені ненульовими значеннями. Це дозволяє компактно кодувати складні семантичні відношення між частинами тексту.

Bi-encoder архітектура передбачає використання двох ідентичних або подібних нейромережевих енкодерів для паралельного кодування запитів і документів (рис. 1.6).

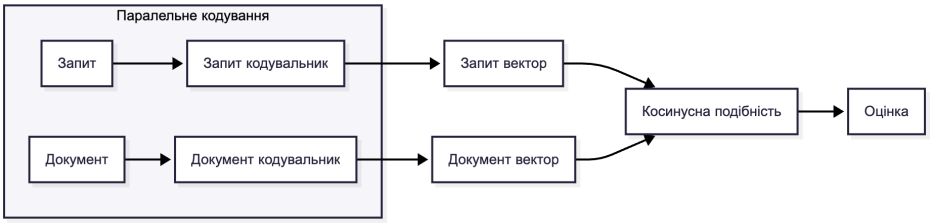

Рисунок 1.3 - Архітектура bi-encoder для dense retrieval.

Формально, bi-encoder реалізує відображення текстових послідовностей у простір фіксованої розмірності:

$$E_\theta : T \to R^d \tag{1.6}$$

де T - множина текстових послідовностей; θ - параметри енкодера; d - розмірність ембедінгу (типово 768 або 1024).

Енкодування запиту та документа виконується незалежно:

$$\vec{q} = E_\theta^{(q)}(query), \quad \vec{d} = E_\theta^{(d)}(document) \tag{1.7}$$

Оцінка релевантності обчислюється як скалярний добуток або косинусна подібність:

$$score(q, d) = \vec{q}^{\,T}\vec{d} = \sum_{i=1}^{d} q_i \cdot d_i \tag{1.8}$$

Саме незалежність кодування забезпечує головну перевагу bi-encoder - можливість офлайн-індексації. На етапі підготовки системи всі документи колекції пропускаються через енкодер, а отримані вектори зберігаються у спеціалізованому векторному індексі. Ця операція виконується одноразово під час розгортання системи, тоді як оновлення індексу потрібне лише при додаванні нових документів. Під час обробки запиту користувача кодується виключно сам запит - один прохід через нейронну мережу замість повторного кодування мільйонів документів. Внаслідок цього час відповіді системи не залежить від розміру колекції на етапі кодування.

Пошук найближчих сусідів у високорозмірному векторному просторі становить окрему алгоритмічну задачу. Точний (exact) пошук найближчих векторів передбачає обчислення відстані до кожного документа в індексі, що дає лінійну складність O(n). Для колекції з мільйона документів це означає мільйон операцій на кожен запит - неприйнятна латентність для інтерактивних систем.

Таблиця 1.2

Порівняння embedding моделей для dense retrieval

Модель	Розмірність	NDCG@10 (BEIR)	Параметри
BGE-base-en-v1.5 [12]	768	64.9%	110M
E5-large-v2 [36]	1024	66.2%	335M
GTR-T5-XXL	768	67.8%	4.8B
text-embedding-3-large	3072	68.5%	-

Модель BGE (BAAI General Embedding) демонструє оптимальне співвідношення якості та обчислювальної ефективності, що робить її оптимальним кандидатом для систем з обмеженими ресурсами [12].

1.2.3. Гібридний пошук Dense + Sparse

Усвідомлення взаємодоповнюваності dense та sparse методів привело до розробки гібридних підходів, що поєднують переваги обох категорій. Dense

retrieval ефективно захоплює семантичну подібність та синоніми, забезпечуючи розуміння контексту навіть за відсутності точних лексичних збігів. Водночас BM25 точно ідентифікує документи зі специфічними ключовими словами, абревіатурами та технічними термінами, що критично важливо для доменно-специфічних запитів. Гібридний пошук забезпечує робастність системи як до семантичних, так і до лексичних запитів, компенсуючи слабкі сторони кожного окремого підходу (рис. 1.8).

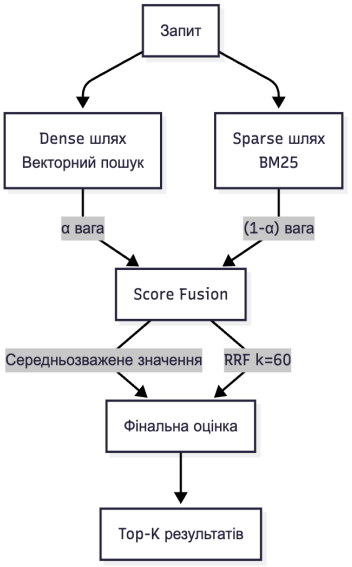

Рисунок 1.4 - Архітектура гібридного пошуку.

На практиці гібридна архітектура передбачає паралельне виконання двох пошукових шляхів. Запит одночасно обробляється векторним пошуком із заданою вагою α та алгоритмом BM25 із комплементарною вагою (1-α). Результати обох шляхів надходять до модуля Score Fusion, де відбувається їх об'єднання для формування фінальної оцінки кожного документа.

Для злиття результатів у системі реалізовано два методи. Перший - Weighted Average Fusion - обчислює зважену суму нормалізованих оцінок dense та sparse компонентів:

$$score_{hybrid} = \alpha \cdot score_{dense} + (1 - \alpha) \cdot score_{sparse} \quad (1.9)$$

де $\alpha \in [0, 1]$ - ваговий коефіцієнт dense компонента, що визначає баланс між семантичним та лексичним пошуком. Значення α близьке до 1 надає пріоритет семантичній подібності, тоді як значення близьке до 0 акцентує увагу на точних лексичних збігах. Оптимальне значення α зазвичай визначається експериментально для конкретного домену та типу запитів.

Другий метод - Reciprocal Rank Fusion (RRF) - використовує позиційну інформацію замість абсолютних оцінок:

$$RRF(d) = \sum_{r \in R} \frac{1}{k + rank_r(d)} \tag{1.10}$$

де R - множина списків ранжування; $rank_r(d)$ - позиція документа d у списку r; k - параметр згладжування (типово $k = 60$).

RRF не потребує нормалізації оцінок та демонструє робастність до різних розподілів scores між системами. Це суттєво спрощує інтеграцію гетерогенних пошукових компонентів, оскільки кожна система може використовувати власну шкалу оцінювання без додаткової калібрації. Крім того, RRF менш чутливий до викидів - документи з аномально високими оцінками в одній системі не домінуватимуть у фінальному ранжуванні.

Після злиття результатів система відбирає Top-K документів із найвищими фінальними оцінками для передачі на наступні етапи обробки. Вибір між Weighted Average Fusion та RRF залежить від специфіки застосування: перший метод забезпечує більш тонкий контроль через налаштування ваг, тоді як другий є більш універсальним рішенням, що не потребує попередньої калібрації систем.

1.2.4. Механізм пізньої взаємодії ColBERT

ColBERT (Contextualized Late Interaction over BERT) реалізує принципово інший підхід до dense retrieval, зберігаючи токен-рівневі

ембедінги замість їх агрегації в єдиний вектор [3]. Документ представляється не точкою у просторі, а матрицею ембедінгів, де кожен рядок відповідає окремому токену тексту. Аналогічно представляється і запит, що уможливлює дрібнозернисте порівняння на рівні окремих слів та фраз (рис. 1.7).

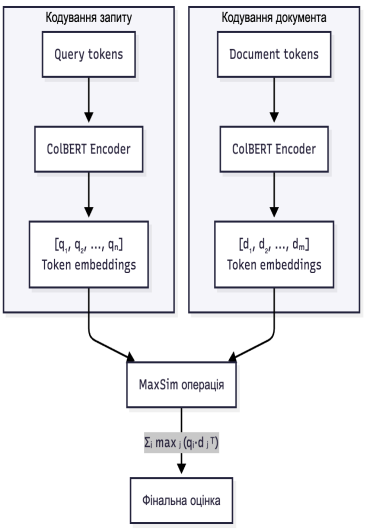

Рисунок 1.5 - Механізм пізньої взаємодії ColBERT.

Архітектура ColBERT складається з двох ідентичних енкодерів на базі BERT, які незалежно обробляють запит та документ. На виході формуються послідовності контекстуалізованих векторів: для запиту - набір [q_1, q_2, ..., $q_{\square}$], для документа - набір [d_1, d_2, ..., $d_{\square}$]. Ключова особливість полягає у тому, що взаємодія між цими представленнями відбувається лише на етапі обчислення фінальної оцінки релевантності, звідси й назва «пізня взаємодія».

Ключовою операцією є MaxSim - обчислення максимальної подібності кожного токена запиту з усіма токенами документа:

$$S_{ColBERT}(q,d) = \sum_{i=1}^{|q|} max^{|d|}{}_{j=1}\left(\vec{q}_i \cdot \vec{d}_j^{\,T}\right) \tag{1.11}$$

де $\vec{q}_i$ - ембединг i-го токена запиту; $\vec{d}_j$ - ембединг j-го токена документа; $|q|$, $|d|$ - кількість токенів у запиті та документі відповідно.

Для реалізації цього механізму застосовується наступний алгоритм: спочатку обчислюється матриця подібностей між усіма парами токенів запиту та документа через матричне множення, потім для кожного токена запиту вибирається максимальне значення подібності серед усіх токенів документа, і нарешті ці максимуми усереднюються для отримання фінальної оцінки релевантності. Такий підхід дозволяє захоплювати часткові співпадіння: навіть якщо лише один токен документа семантично близький до токена запиту, це враховується при ранжуванні.

Таблиця 1.3

Порівняння ColBERT та bi-encoder підходів

Критерій	Bi-encoder	ColBERT
Гранулярність	Документ → вектор	Документ → набір векторів
Розмір індексу	~3 КБ/документ	~30-100 КБ/документ
Латентність пошуку	<10 мс	50-200 мс
NDCG@10 (SciFact)	64.9%	67.1%
Інтерпретованість	Низька	Висока (токен-матчинг)

Порівняно з bi-encoder, ColBERT забезпечує вищу точність за рахунок збереження локальної семантичної структури. Одним з головних недоліків ColBERT є значно більший розмір індексу - зберігання окремих векторів для кожного токена потребує у 10-30 разів більше пам'яті порівняно з класичним bi-encoder. Латентність пошуку також зростає через необхідність обчислення MaxSim операції. Компроміс між якістю та ефективністю досягається через техніки стисненн.

Додатковою перевагою ColBERT є висока інтерпретованість результатів. На відміну від bi-encoder, де складно пояснити, чому документ отримав певну оцінку, у ColBERT можна візуалізувати, які саме токени запиту знайшли найкраще співпадіння з токенами документа.

1.3. Методологія HyDE для покращення пошуку

1.3.1. Теоретичні засади гіпотетичних документів

Hypothetical Document Embeddings (HyDE) - інноваційний підхід до подолання лексичного розриву між запитом користувача та релевантними документами у корпусі [5]. Ключова ідея полягає у використанні генеративної моделі для синтезу гіпотетичного документа, який би потенційно відповідав на запит користувача. Після генерації, на базі сформованих гіпотетичних відповідей виконується пошук реальних документів, семантично близьких до штучно створеного тексту.

Як правило, користувацькі запити складаються з мало змістовних запитань, тоді як для отримання найбільш оптимальної відповіді критично мати гарно сформований за контекстом запит. На кожне питання відбувається побудова ембедінгів і пряме їх порівняння та твердження часто призводить до субоптимальних результатів через стилістичну та структурну невідповідність. HyDE вирішує цю проблему елегантно за допомогою трансформації запиту у формат документа, який найбільш схожий за стилем на текст в індексованому матеріали. Внаслідок цього суттєво зменшується семантична відстань між пошуковим вектором та релевантними документами (рис. 1.6).

Рисунок 1.6 - Pipeline методу HyDE.

1.3.2. Формалізація методу

Нехай q - вхідний запит користувача, G - генеративна модель, E - енкодер для отримання векторних представлень. В такому випадку HyDE можна представити як послідовність із трьох кроків.

1. **Генерація гіпотетичного документа:**

$$h = G\left(q; \theta_g\right) \tag{1.12}$$

де θ_g - параметри генеративної моделі, включаючи системний промпт та температура.

2. **Кодування гіпотетичного документа:**

$$\vec{h} = E(h)$$

3. **Пошук реальних документів:**

$$D_{rel} = Top - K\left(sim\left(\vec{h}, \vec{d}_i\right)\right), \quad d_i \in D$$

де D - корпус документів; D_{rel} - множина знайдених релевантних документів.

1.3.3. Оптимізація параметрів генерації

Якість гіпотетичних документів критично впливає на ефективність пошуку. Емпірично встановлено оптимальні параметри генерації [5]. Кількість гіпотез (n) у деякому діапазоні забезпечує баланс між якістю покриття семантичного простору та обчислювальними витратами. Для забезпечення різноманітності та когерентності застосовується температура. Поріг якості (threshold) відсіює дегенеративні або надто короткі відповіді.

Усереднення ембедінгів множини гіпотетичних документів підвищує робастність фінального вектора:

$$\vec{h}_{avg} = \frac{1}{n}\sum_{i=1}^{n} E\left(h_i\right) \tag{1.13}$$

1.3.4. Комбінування HyDE з ColBERT

Гібридний підхід HyDE-ColBERT поєднує генерацію гіпотетичних документів з механізмом пізньої взаємодії (рис. 1.10). Це дозволяє використати семантичне збагачення запиту разом з дрібнозернистим токен-рівневим порівнянням.

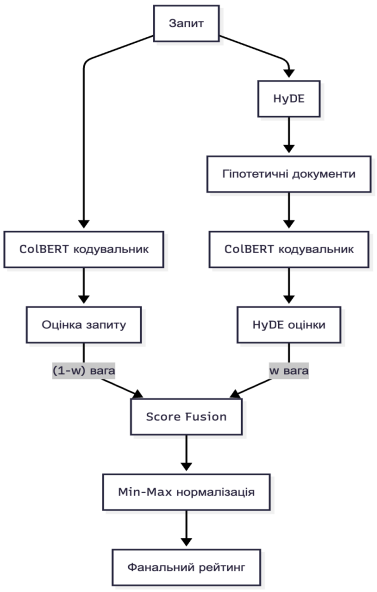

Рисунок 1.7 - Архітектура комбінованого методу HyDE-ColBERT.

Злиття оцінок виконується через зважене усереднення нормалізованих scores:

$$score_{final} = w \cdot \hat{S}_{hyde} + (1 - w) \cdot \hat{S}_{query} \quad (1.14)$$

де $\hat{S}$ - min-max нормалізована оцінка; w - ваговий коефіцієнт HyDE компонента (оптимально $w = 0.1 - 0.2$), що означає домінування прямого ColBERT-пошуку з м'яким впливом гіпотетичного документа.

Висновки за розділом 1

У першому розділі кваліфікаційної роботи проведено комплексний аналіз сучасних технологій побудови RAG-систем.

Архітектура Retrieval-Augmented Generation ефективно усуває ключові обмеження автономних великих мовних моделей - галюцинації та статичність знань - через інтеграцію механізму пошуку релевантної інформації із зовнішньої бази знань. Лексичний пошук BM25 залишається релевантним для keyword-запитів та демонструє переваги в інтерпретованості. Гібридний підхід Dense + Sparse з оптимальним співвідношенням компонентів забезпечує робастність до різних типів запитів. Методологія HyDE вирішує проблему лексичного розриву через генерацію гіпотетичних документів. Комбінування HyDE з ColBERT дозволяє досягти синергетичного ефекту семантичного збагачення та дрібнозернистого порівняння.

Проведений аналіз обґрунтовує доцільність розробки власної мікросервісної RAG системи, що поєднує гібридний пошук Dense + Sparse з методом HyDE-ColBERT для досягнення оптимального балансу якості та продуктивності.

РОЗДІЛ 2
ПРОЕКТУВАННЯ МУЛЬТИМОДАЛЬНОЇ RAG АРХІТЕКТУРИ

2.1. Обґрунтування вибору мікросервісної архітектури

2.1.1. Порівняльний аналіз архітектурних підходів

Проектування програмних систем промислового рівня вимагає обґрунтованого вибору архітектурного стилю, адже це рішення визначатиме технічні характеристики продукту протягом усього життєвого циклу. Помилка на етапі архітектурного проектування може призвести до необхідності повного перепроектування системи, що пов'язано зі значними фінансовими та часовими втратами. Два домінуючі підходи - монолітна та мікросервісна архітектури - принципово відрізняються за стратегіями декомпозиції, механізмами масштабування та операційними характеристиками [29]. Вибір між ними залежить від специфіки предметної області, очікуваних патернів навантаження та організаційної структури команди розробників.

Монолітна парадигма передбачає консолідацію усіх функціональних модулів у межах єдиного застосунку, що розгортається як неподільний артефакт. У контексті RAG системи це означає, що компоненти обробки запитів, семантичного пошуку, ранжування документів та генерації відповідей виконуються в спільному адресному просторі, взаємодіючи через прямі виклики процедур або функцій. Така організація має безперечну перевагу - мінімальну латентність міжкомпонентного обміну даними, оскільки повністю відсутні накладні витрати на мережеву комунікацію, серіалізацію та десеріалізацію повідомлень [30]. Виклик методу сусіднього модуля відбувається за наносекунди, тоді як мережевий запит потребує мілісекунд. Крім того, моноліт простіший у розгортанні - один артефакт копіюється на сервер та запускається без складної оркестрації.

Водночас монолітна архітектура демонструє низку суттєвих обмежень

при масштабуванні навантажень, характерних для RAG-систем. Гетерогенність обчислювальних вимог окремих етапів обробки створює фундаментальну проблему. CPU-bound операції аналізу запитів, GPU-intensive embedding inference та I/O-bound взаємодія з векторними сховищами потребують радикально різних апаратних ресурсів. У межах монолітного процесу неможливо оптимально розподілити ці ресурси - GPU або виділяється всьому застосунку, або не виділяється взагалі. Масштабування ресурсоємного компонента семантичного пошуку автоматично потребує реплікації всієї системи цілком, включаючи легковагі модулі форматування відповідей, що призводить до неефективного використання інфраструктури [31]. Додаткова проблема полягає у спільному релізному циклі - будь-яка зміна в одному модулі вимагає перерозгортання всього застосунку.

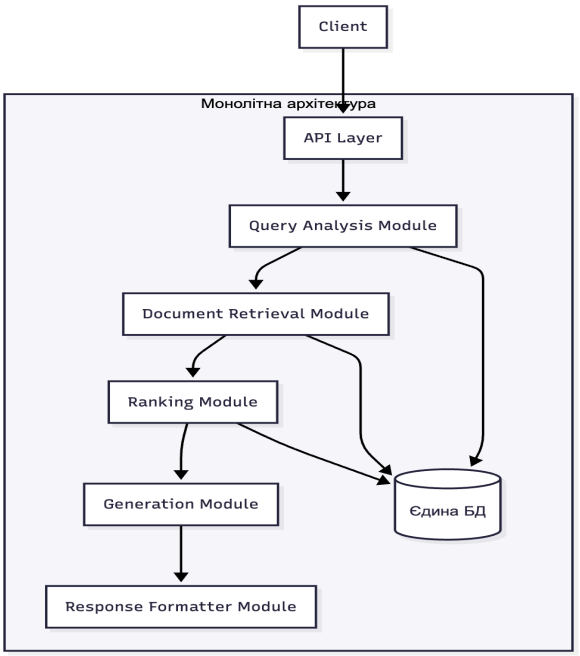

Рисунок 2.1 - Структура монолітної RAG системи з гетерогенними обчислювальними вимогами.

Мікросервісна архітектура реалізує декомпозицію на незалежні автономні сервіси з чітко розділеними межами відповідальності. Кожен

мікросервіс інкапсулює окрему свою окрему бізнес логіку, володіє власним станом та комунікує з іншими компонентами виключно через стандартизовані мережеві протоколи - REST API, gRPC або асинхронні черги повідомлень [29]. Принцип ізоляції забезпечує можливість незалежного розгортання, масштабування та еволюції окремих сервісів. Команда, відповідальна за пошуковий компонент, може оновлювати embedding-модель без координації з командою генерації відповідей. Це прискорює загальний темп розробки та зменшує ризики при релізах.

Проте мікросервісний підхід в підвищує складність системи, з'являється потреба в оркестрації контейнерів, централізованому логуванні та складних стратегіях розгортання. Комунікація між сервісами додає додаткову латентність та створює потенційних точки відмови. Контрактне тестування стає обов'язковим для забезпечення сумісності між сервісами, що еволюціонують незалежно.

Таблиця 2.1

Порівняльний аналіз архітектурних підходів для RAG-систем

Критерій	Монолітна архітектура	Мікросервісна архітектура
Стратегія масштабування	Вертикальна (весь застосунок)	Горизонтальна (окремі сервіси)
Відмовостійкість	Єдина точка відмови	Ізоляція збоїв, graceful degradation
Час розгортання оновлень	Хвилини-години	Секунди-хвилини (окремий сервіс)
Технологічна гнучкість	Єдиний технологічний стек	Polyglot persistence та programming
Операційна складність	Низька	Висока (оркестрація, моніторинг)
Міжкомпонентна латентність	Мінімальна (in-process)	Підвищена (мережеві виклики)
Автономність команд	Обмежена	Повна незалежність
Контрактне тестування	Не застосовується	Обов'язкове

2.1.2. Специфічні вимоги RAG-систем

Архітектурне рішення для RAG-платформи має враховувати специфічні характеристики робочого навантаження, що принципово відрізняються від патернів типових веб-застосунків [2]. Детальний аналіз обчислювального профілю RAG системи дозволив виявити низку закономірностей, які безпосередньо впливають на вибір архітектурного підходу.

Гетерогенність ресурсних вимог проявляється у радикально різних потребах окремих компонентів системи. Сервіс пошуку документів потребує GPU-прискорення для виконання inference операцій embedding-моделей, оскільки векторизація текстових фрагментів є обчислювально інтенсивною задачею, що ефективно паралелізується на графічних процесорах. Водночас API Gateway виконує переважно CPU орієнтовані операції маршрутизації запитів, валідації вхідних даних та управління сесіями, для яких GPU ресурси є надлишковими. Мікросервісна декомпозиція дає можливість рівномірно розприлілити ресурси між сервісами, які дійсно потребують апаратного прискорення, оптимізуючі загальну розходи на інфраструктури платформи.

Варіативність патернів навантаження характеризує нерівномірний розподіл часу виконання між етапами обробки запиту. Сервіс генерації відповідей демонструє найвищу латентність - від однієї до трьох секунд на кожен запит залежно від довжини контексту та складності відповіді. Натомість модуль аналізу запитів опрацьовує вхідні дані за десятки мілісекунд, виконуючи токенізацію, визначення мови та екстракцію ключових сутностей. За таких умов незалежне масштабування дозволяє точково збільшувати кількість реплік компонентів-«вузьких місць» без надмірного виділення ресурсів швидким модулям, що працюють із значним запасом продуктивності.

Еволюція компонентів у RAG системах відбувається нерівномірно та асинхронно. Заміна embedding-моделі - наприклад, міграція з BGE-base на

ColBERTv2 для підвищення якості пошуку - потребує модифікації виключно сервісу документного пошуку. Решта системи продовжує функціонувати без змін, приймаючи на вхід ті самі векторні представлення, лише згенеровані новою моделлю. У монолітній архітектурі така, здавалось би, локальна зміна вимагала б повного перерозгортання всього застосунку з відповідними ризиками та простоєм.

Відмовостійкість реалізується через патерн Circuit Breaker, що особливо ефективно працює у мікросервісному середовищі [31]. Тимчасова недоступність одного з модулів не повинна блокувати обробку роботу інших сервсів. Коли Circuit Breaker фіксує серію невдалих викликів до певного сервісу, то він автоматично перемикається у відкритий стан, миттєво повертаючи fallback-відповідь замість очікування таймауту. Це забезпечує graceful degradation функціональності - система продовжує працювати у зниженому режимі, явно інформуючи про недоступні можливості.

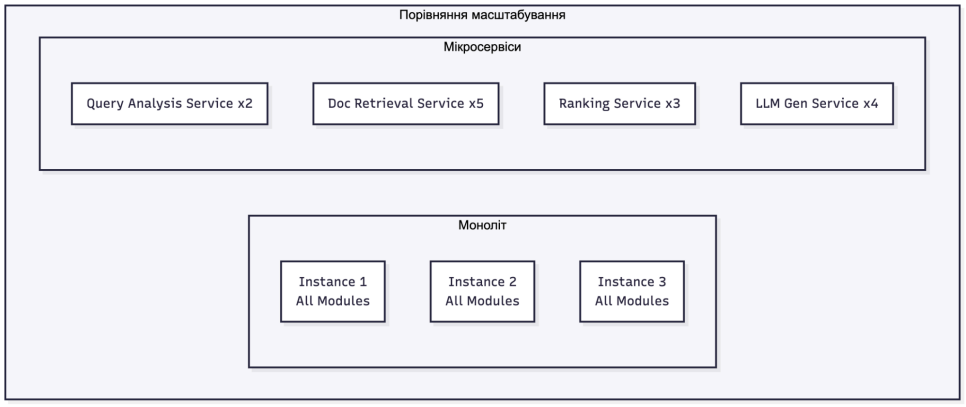

Рисунок 2.2 - Порівняння стратегій масштабування для RAG-систем.

2.1.3. Аргументація архітектурного вибору

Комплексний аналіз функціональних та нефункціональних вимог обґрунтовує доцільність застосування мікросервісної архітектури для розробки RAG платформи [29]. Нижче представлено ключові критерії, за

якими мікросервісний підхід демонструє переваги над альтернативними архітектурними рішеннями.

Оптимізація використання ресурсів досягається завдяки можливості більш точково застосувати апаратних ресурсів. GPU-прискорення використовується виключно для сервісів які працюють з ембедингами та cross-encoder ранжування. Решта компонентів - API Gateway, Query Router, Response Formatter - розгортаються на стандартних CPU вузлах з меншою вартістю, що в майбутньому дозволить суттєво зменшити сукупну вартість інфраструктури порівняно зі сценарієм GPU оснащення для кожної репліки моноліту.

Еластичність системи забезпечується механізмом автоматичного горизонтального масштабування. На основі метрик CPU та GPU оркестратор динамічно адаптує кількість реплік кожного сервісу відповідно до поточного навантаження. Під час пікової активності кількість інстансів LLM Generation Service може зростати у кілька разів, а в періоди низького навантаження - скорочуватися до мінімальної конфігурації. Весь процес відбувається автоматично без потреби ручного втручання операційної команди.

Незалежне розгортання мінімізує ризики та час простою при оновленнях. Оновлення моделі ранжування потребує перерозгортання лише відповідного сервісу - процес, що триває лічені секунди та не впливає на доступність решти системи. За стратегії rolling update нові репліки поступово замінюють старі, забезпечуючи безперервність обслуговування користувацьких запитів протягом усього процесу оновлення.

2.2. Загальна архітектура системи

2.2.1. Архітектура запропонованої мікросервісної RAG системи

Розроблена RAG система розроблена таким чином, що кожен рівень має свою зону відповідальності і таким чином забезпечує можливість незалежну еволюції [29]. Клієнтський рівень відповідає за взаємодію з користувачем,

другий - за маршрутизацію запитів, третій реалізує основну бізнес-логіку обробки, а четвертий забезпечує персистентність даних. Такий підхід до декомпозиції системи значно спрощує підтримку кодової бази, оскільки зміни в одному рівні мінімально впливають на інші. Крім того, кожен рівень може масштабуватися незалежно, що критично важливо для систем з нерівномірним навантаженням. Загальну структуру взаємодії компонентів наведено на (рис. 2.3)

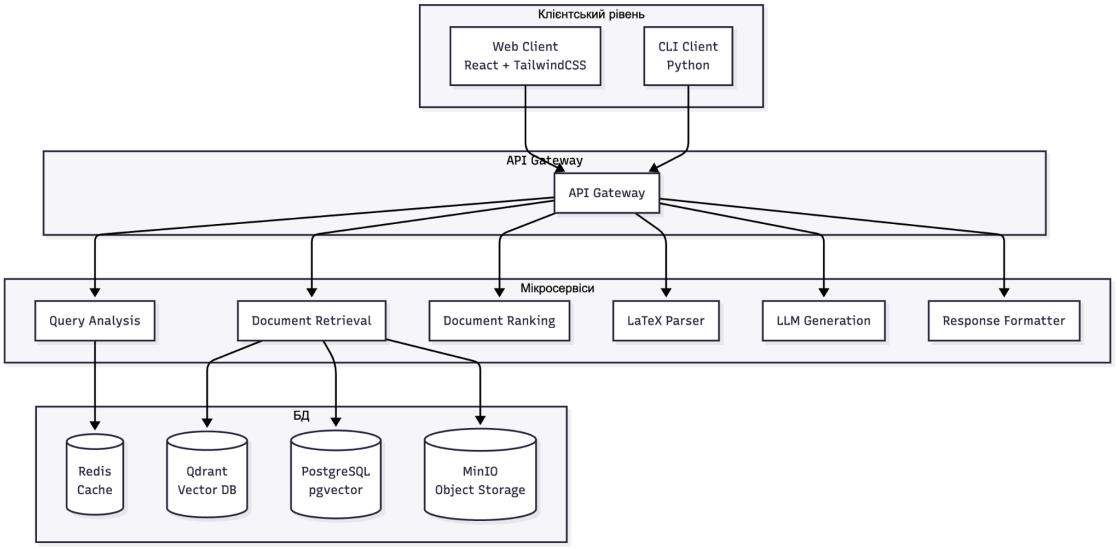

Рисунок 2.3 - Загальна архітектура мікросервісної RAG системи.

Клієнтський рівень охоплює всі точки взаємодії кінцевих користувачів із системою та забезпечує зручний доступ до функціональності через два принципово різних інтерфейси. Основним каналом взаємодії слугує веб-застосунок, реалізований за принципом Single Page Application (SPA). Для його створення обрано бібліотеку React у поєднанні з TypeScript, що забезпечує статичну типізацію та значно зменшує кількість runtime-помилок під час розробки [26]. Візуальне оформлення інтерфейсу базується на утилітарному CSS-фреймворку TailwindCSS, який дозволяє швидко створювати консистентний дизайн без написання кастомних стилів. Збірку проєкту виконує інструмент Vite, що забезпечує миттєве перезавантаження

модулів під час розробки завдяки використанню нативних ES-модулів браузера.

Графічний інтерфейс веб-клієнта реалізує потокове відображення відповідей, що дозволяє користувачу бачити результати генерації в реальному часі без очікування повної обробки запиту. Окрім того, інтерфейс підтримує візуалізацію джерел, на основі яких сформовано відповідь, та коректний рендеринг математичних формул у форматі LaTeX. Ці можливості особливо важливі для наукових та технічних застосувань, де точність відображення формул є критичною.

Паралельно з веб-інтерфейсом функціонує CLI-клієнт на Python, призначений для автоматизованих сценаріїв використання. Command Line Interface надає програмний доступ до функціональності системи для інтеграції в автоматизовані процеси та скрипти. Цей інструмент дозволяє інтегрувати RAG-систему в CI/CD-конвеєри, batch-обробку документів та скрипти автоматизації без необхідності графічного інтерфейсу. Обидва клієнти взаємодіють із системою виключно через REST API та Server-Sent Events, що забезпечує уніфікований протокол комунікації незалежно від типу інтерфейсу.

Рівень API Gateway виступає єдиною точкою входу для всіх зовнішніх запитів до системи. Такий архітектурний патерн значно спрощує взаємодію з точки зору клієнтів, оскільки їм не потрібно знати про внутрішню структуру мікросервісів та їх розташування. Компонент побудовано на базі асинхронного веб-фреймворку FastAPI, що демонструє високу продуктивність завдяки використанню ASGI-сервера Uvicorn та підтримці асинхронних операцій введення-виведення [26]. FastAPI автоматично генерує OpenAPI-специфікацію на основі анотацій типів Python, що суттєво полегшує документування API та інтеграцію зі сторонніми системами. Валідація вхідних даних реалізована через Pydantic-моделі, які перевіряють структуру та типи даних ще до передачі запиту бізнес-логіці.

Gateway інкапсулює наскрізну функціональність (cross-cutting concerns): автентифікацію на основі JWT-токенів, обмеження частоти запитів (rate limiting) для захисту від перевантаження, налаштування CORS-політик для безпечної кросдоменної взаємодії, а також агрегацію відповідей від декількох downstream-сервісів в єдиний результат. Внаслідок цього клієнтські застосунки отримують уніфікований інтерфейс без необхідності реалізовувати ці механізми самостійно.

Мікросервісний рівень містить шість спеціалізованих компонентів, кожен з яких відповідає за конкретний етап RAG-конвеєра обробки запитів:

1. **Query Analysis** - виконує лінгвістичний аналіз вхідних запитів користувача, включаючи токенізацію, виділення ключових сутностей та класифікацію інтенту запиту для оптимізації подальшого пошуку.
2. **Document Retrieval** - реалізує гібридний пошук релевантних документів, поєднуючи лексичні методи на основі BM25 із семантичним пошуком через векторні представлення.
3. **Document Ranking** - здійснює переранжування знайдених документів за допомогою cross-encoder моделей, що аналізують пари запит-документ для точнішого визначення релевантності.
4. **LaTeX Parser** - обробляє математичні формули у форматі LaTeX, забезпечуючи їх коректний рендеринг та інтеграцію у текстові відповіді системи.
5. **LLM Generation** - формує текстові відповіді на основі контексту знайдених документів, використовуючи великі мовні моделі.
6. **Response Formatter** - структурує фінальну відповідь, додаючи посилання на джерела, форматування та метадані для зручного відображення на клієнті.

Кожен сервіс розгортається як окремий Docker-контейнер, що забезпечує ізоляцію залежностей та можливість незалежного масштабування під навантаженням. Комунікація між сервісами відбувається через внутрішню

мережу з використанням легковагових HTTP-запитів та gRPC для критичних за продуктивністю операцій.

Рівень даних об'єднує спеціалізовані системи зберігання, оптимізовані для різних патернів доступу та типів інформації. Векторна база даних Qdrant забезпечує швидкий пошук за семантичною подібністю, підтримуючи мільйони векторів з субсекундним часом відгуку [23]. PostgreSQL із розширенням pgvector слугує основним реляційним сховищем для метаданих документів, історії запитів та конфігураційних параметрів системи. Redis виконує роль кешу для часто запитуваних результатів, що значно знижує латентність повторних запитів та навантаження на обчислювальні сервіси. MinIO надає S3-сумісне об'єктне сховище для зберігання оригінальних документів у різних форматах - PDF, DOCX, HTML та інших. У результаті така комбінація сховищ дозволяє оптимально використовувати сильні сторони кожної технології залежно від характеру даних та операцій над ними.

2.2.2. Потік обробки запиту

Обробка користувацького запиту реалізує послідовний конвеєр з можливістю паралельного виконання незалежних етапів (рис. 2.4). Кореляційний ідентифікатор (UUID) забезпечує наскрізне трасування запиту через усі компоненти системи.

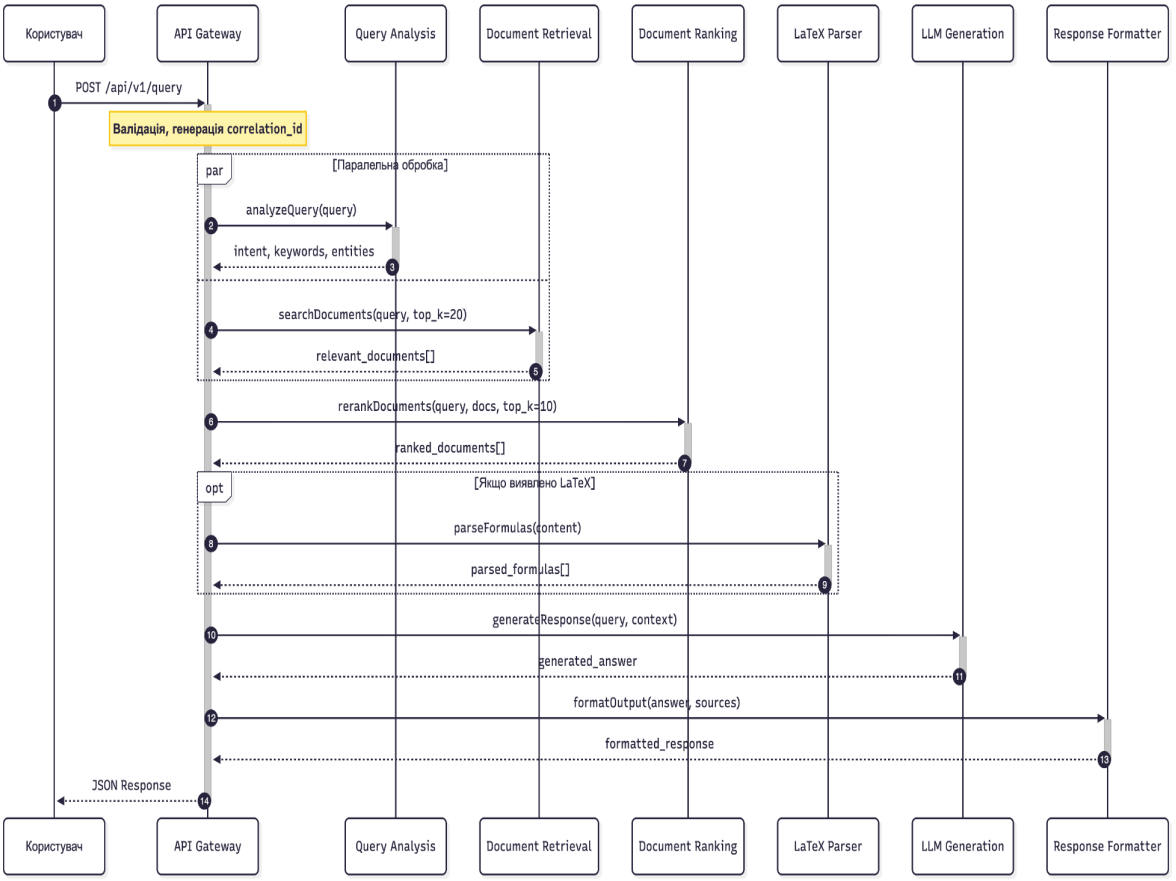

Рисунок 2.4 - Діаграма послідовності обробки запиту в RAG-системі.

Алгоритм обробки складається з шести послідовних етапів, кожен з яких виконує чітко визначену функцію.

1. Прийом та валідація запиту. Клієнтський запит надходить до API Gateway через REST POST /api/v1/query. Gateway виконує валідацію вхідних параметрів за допомогою Pydantic-схеми QueryRequest, яка перевіряє обов'язкові поля, типи даних та обмеження на довжину тексту запиту. У разі успішної валідації генерується унікальний correlation_id у форматі UUID v4, який записується в контекст запиту та передається в заголовках до кожного наступного сервісу. За потреби на цьому етапі також виконується автентифікація користувача через JWT-токен. Валідація на вході відсікає некоректні запити ще до навантаження на внутрішні сервіси, що економить обчислювальні ресурси.

2. Паралельний аналіз запиту та пошук документів. Після валідації ініціюються дві незалежні операції, які виконуються паралельно. Query Analysis Service отримує текст запиту та виконує його семантичний аналіз: визначає інтент (інформаційний, навігаційний, транзакційний), екстрагує ключові слова та іменовані сутності, оцінює складність запиту. Ці метадані використовуються для оптимізації подальших етапів - наприклад, для вибору стратегії ранжування або налаштування параметрів генерації. Одночасно Document Retrieval Service розпочинає гібридний пошук релевантних документів із використанням оригінального тексту запиту. Сервіс виконує паралельні запити до BM25-індексу (лексичний пошук) та векторної бази (семантичний пошук), після чого об'єднує результати через Reciprocal Rank Fusion. На виході формується список top-20 документів-кандидатів. Паралельне виконання цих двох операцій скорочує загальну латентність етапу приблизно на 15-20% порівняно з послідовним підходом, оскільки найдовша операція (пошук документів) не блокується аналізом запиту.

3. Переранжування документів. Результати пошуку (типово 20 кандидатів) передаються до Document Ranking Service, який застосовує cross-encoder модель для точнішої оцінки релевантності. На відміну від bi-encoder моделей, що генерують незалежні ембедінги для запиту та документа, cross-encoder обробляє пару "запит-документ" разом, що дозволяє враховувати тонкі семантичні зв'язки між ними. Модель присвоює кожному документу скаляр релевантності, за яким виконується сортування. З відсортованого списку відбираються top-10 найбільш релевантних документів для передачі на етап генерації. Параметр top-K є конфігурованим і може змінюватися залежно від вимог до якості відповіді та обмежень на розмір контекстного вікна LLM.

4. Умовний парсинг LaTeX. LaTeX Parser Service активується умовно - лише при виявленні математичних конструкцій у тексті запиту або в контенті відібраних документів. Детектор математичних формул використовує

регулярні вирази для пошуку патернів на кшталт $...$, \[...\], \begin{equation} тощо. За наявності математичного контенту парсер перетворює LaTeX-нотацію у структурований формат, придатний для коректного відображення у відповіді. Якщо математичних конструкцій не виявлено, етап пропускається, що економить час обробки для текстових запитів без формул. Умовна активація реалізована через паттерн Circuit Breaker - сервіс викликається лише за спрацювання відповідного предиката.

5. Генерація відповіді. LLM Generation Service отримує переранжований набір документів та формує контекстний промпт для мовної моделі. Промпт конструюється за шаблоном, що включає системну інструкцію, контекст із релевантних документів (з метаданими про джерела) та власне текст користувацького запиту. Загальний розмір промпту контролюється з урахуванням обмежень контекстного вікна обраної моделі - за перевищення ліміту найменш релевантні документи відсікаються. Генерація відповіді може виконуватися в двох режимах: синхронному (повна відповідь повертається після завершення генерації) або потоковому (streaming). У потоковому режимі токени передаються клієнту в реальному часі через Server-Sent Events (SSE), що суттєво покращує користувацький досвід для довгих відповідей - користувач бачить текст одразу, не очікуючи завершення генерації.

5. Форматування відповіді. Response Formatter Service виконує фінальну обробку згенерованого тексту перед поверненням клієнту. Форматування реалізоване через Jinja2-шаблони, що забезпечує гнучкість у структуруванні вихідних даних. Сервіс додає до відповіді структуровані посилання на джерела (з назвами документів, URL та номерами сторінок), форматує текст відповідно до обраного формату виводу (plain text, Markdown, HTML), інтегрує метадані про час обробки та використані ресурси. Результат упаковується в JSON-структуру відповідно до специфікації API та повертається клієнту з HTTP-статусом 200. У разі помилок на будь-якому

етапі формується відповідь з відповідним кодом помилки та діагностичним повідомленням, що включає correlation_id для подальшого аналізу.

Загальна латентність обробки типового запиту становить 800-1200 мс, з яких близько 60% припадає на етап генерації відповіді LLM. Архітектура конвеєра дозволяє масштабувати кожен етап незалежно та оптимізувати вузькі місця без впливу на інші компоненти.

2.3. Структура API Gateway

API Gateway слугує єдиною точкою входу для всіх зовнішніх запитів до RAG системи. Цей компонент реалізує паттерн Facade, приховуючи складність внутрішньої мікросервісної архітектури від клієнтських застосунків. Замість того щоб кожен сервіс самостійно імплементував автентифікацію, валідацію та логування, ці cross-cutting concerns винесено на рівень Gateway. Це спрощує розробку downstream-сервісів та забезпечує консистентну поведінку системи.

Таблиця 2.2

Специфікація REST API Gateway

Endpoint	Метод	Опис	Параметри
`/api/v1/query`	POST	Основний RAG-запит	query, top_k, format, enable_hyde
`/api/v1/upload`	POST	Завантаження документів	file, collection_name
`/api/v1/search`	POST	Пошук без генерації	query, method, top_k
`/api/v1/health`	GET	Статус системи	-
`/api/v1/models`	GET	Список моделей	-
`/api/v1/collections`	GET	Колекції документів	-

2.4. Сервіс аналізу запитів

У RAG системах якість фінальної відповіді безпосередньо залежить від того, наскільки точно система розуміє вхідний запит користувача. Початковий текстовий запит сам по собі не представляє дуже великої цінності і

передавати його безпосередньо до пошукового компонента лише означає втрачати цінну інформацію про структуру, намір та контекст питання.

Query Analysis Service побудований для вирішення саме цієї проблеми. Сервіс виконує глибоку NLP-обробку вхідного запиту та перетворює неструктурований текст на багатовимірний набір ознак, які в подальшому передаються наступних сервісам для прийняття більш конкретних рішень. Без цього проміжного кроку кожен компонент системи мусив би самостійно аналізувати наданий запит, що призводило б до дублювання обчислень та неузгодженості інтерпретацій.

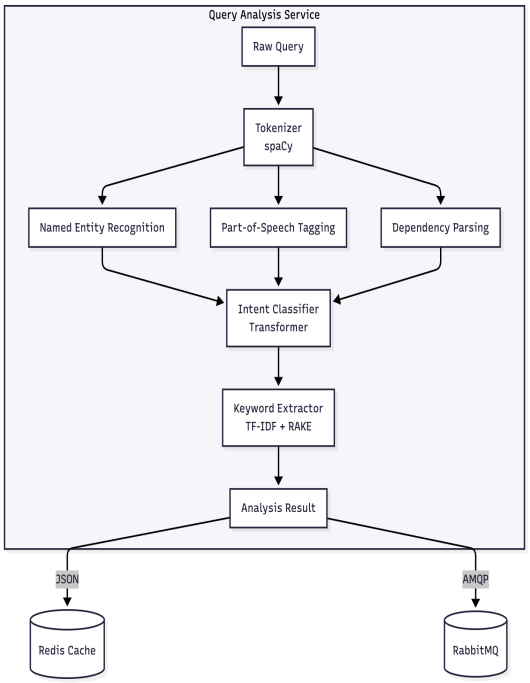

Рисунок 2.5 - Архітектура сервісу аналізу запитів.

Першим етапом запит користувача потрапляє на вхід токенизатора, який проводить семантичний аналіз та розбиває його на лексичні одиниці із збереженням позицій у вихідному тексті.

Named Entity Recognition ідентифікує класифікуючи результат токенизатора за їх типами: PERSON (персони), ORG (організації), DATE

(дати), GPE (географічні об'єкти), PRODUCT (продукти). Виділені сутності використовуються для уточнення пошукового запиту та фільтрації результатів.

Part of speech tagging визначає для кожного токена визначає його частину мови. Ця інформація є основою для граматичного аналізу та виділення ключових слів - переважно іменників та дієслів, що несуть основне семантичне навантаження.

Dependency parsing будує синтаксичне дерево залежностей, визначаючи граматичні зв'язки між словами: підмет, присудок, додаток, означення. Дерево дозволяє ідентифікувати головні терміни запиту та їх модифікатори.

Intent classification категоризує намір запиту за таксономією: factoid (фактологічні питання), comparison (порівняльні), how-to (процедурні), definition (запити на визначення), calculation (обчислювальні).

За допомогою TF-IDF визначаються ключові слова на основі частоти входження та дискримінантної здатності. RAKE виявляє ключові фрази на основі частоти спільного входження слів, після чого результати об'єднуються для формування фінального списку.

2.5. Сервіс пошуку документів

2.5.1. Гібридна архітектура пошуку

Document Retrieval Service є центральним компонентом RAG системи так забезпечує семантичний пошук релевантних документів у векторному просторі [11]. Проектування сервісу базується на концепції гібридного пошуку, яка поєднує переваги семантичних та лексичних методів для пошуку релевантної інформації. Кожен вхідний запит обробляється паралельно різними методами, після чого їх результати агрегуються для формування фінального списку гіпотетичних кандидатів документів (рис. 2.6).

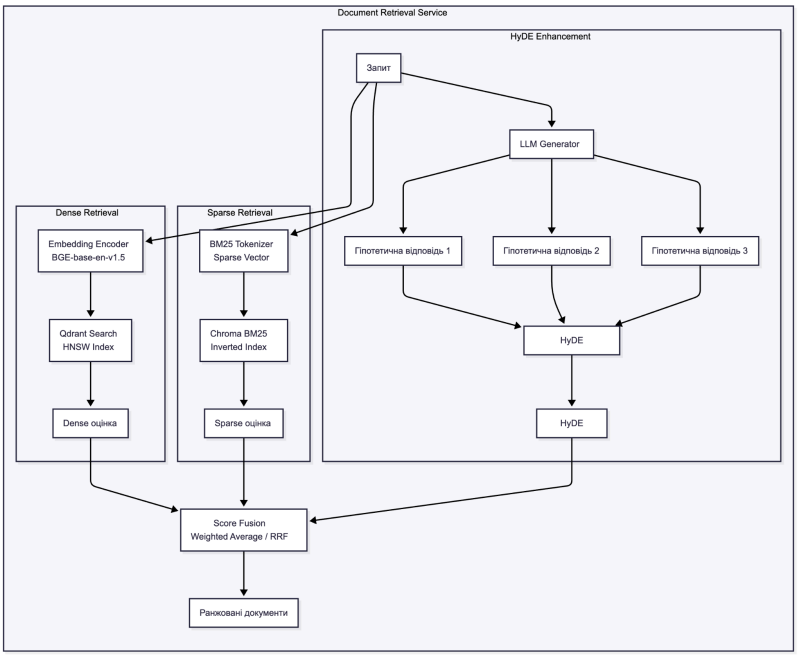

Рисунок 2.6 - Архітектура гібридного пошуку з HyDE.

Модуль Dense Retrieval реалізує семантичний пошук у векторному просторі, де кожен запит кодується у векторне представлення [12], після чого у відбувається за метрикою косинусної подібності відбувається пошук у векторній базі Qdrant.

Sparse Retrieval реалізує класичний лексичний пошук на основі алгоритму BM25 через Chroma [7]. Конфігурація параметрів відповідає рекомендаціям літератури та підтверджена експериментально: k1 = 1.5 контролює швидкість насичення частоти термінів, b = 0.75 визначає ступінь нормалізації за довжиною документа, epsilon = 0.25 встановлює мінімальне значення IDF. BM25 особливо ефективний для keyword-запитів, що містять специфічні терміни, абревіатури та власні назви, які dense-методи можуть «розмивати» у семантичному просторі через обмежену дискримінантну здатність у локальних семантичних областях.

HyDE Enhancement реалізує методологію Hypothetical Document Embeddings для подолання лексичного розриву між запитом і документами

[5]. Генеративна модель створює гіпотетичні документи, що могли б відповідати на запит. Для забезпечення різноманітності гіпотез кожна генерується з різною температурою: 0.3 (консервативна), 0.5 (збалансована), 0.7 (креативна). Максимальна довжина гіпотези обмежена 256 токенами. Ембедінги гіпотетичних документів усереднюються та використовуються для додаткового семантичного пошуку, що дозволяє знаходити релевантні документи навіть за відсутності прямого лексичного перекриття.

2.5.2. Стратегії об'єднання результатів

Отримавши відповідь від кожного методу виконується їх агрегація та формування фінального ранжованого списку. Було реалізовано дві стратегії злиття.

Weighted Average Fusion - лінійна комбінація нормалізованих оцінок:

$$\text{score}_{\text{final}} = \alpha \cdot \{S\}_{\text{dense}} + \beta \cdot \{S\}_{\text{sparse}} + \gamma \cdot \{S\}_{\text{hyde}} \quad (2.1)$$

де $\hat{S}$ - min-max нормалізована оцінка;

α, β, γ - вагові коефіцієнти ($\alpha + \beta + \gamma = 1$); типові значення: $\alpha = 0.3$, $\beta = 0.7$, $\gamma = 0.1$.

Перед злиттям оцінки кожного джерела нормалізуються методом min-max до діапазону [0, 1], що забезпечує їх порівнянність. Експериментально визначено оптимальні вагові коефіцієнти: dense_weight = 0.3, sparse_weight = 0.7, hyde_weight = 0.1. Домінування BM25-компоненти (70%) пояснюється тим, що лексичний пошук забезпечує високу точність для запитів зі специфічними термінами, які щільні вектори можуть неточно відображати через обмежену дискримінантну здатність у локальних семантичних областях. Водночас 30% ваги dense компонента достатньо для захоплення семантичної подібності синонімів та подолання vocabulary mismatch. Така конфігурація досягає NDCG@10 = 79.28%, що становить приріст на 38.6% порівняно з найкращим однорідним методом.

Reciprocal Rank Fusion (RRF) - реалізує альтернативний підхід, що використовує позиційну інформацію замість абсолютних оцінок:

$$RRF(d) = \Sigma r \in R\ (1)/(k + rankr(d)) \quad (2.2)$$

де R - множина ранжованих списків від кожного retriever;

$rank_r(d)$ - позиція документа d у списку r;

$k = 60$ - константа згладжування.

RRF не потребує нормалізації оцінок та демонструє робастність до різних розподілів scores, що спрощує інтеграцію гетерогенних пошукових компонентів. Метод застосовується як альтернативний варіант для сценаріїв, де абсолютні значення оцінок менш інформативні.

Я кість гіпотетичних документів критично впливає на ефективність HyDE пошуку. Для відсіювання неякісних гіпотез було реалізовано механізм фільтрації з порогом quality_threshold = 0.7.

Оцінка якості враховує кілька критеріїв. Мінімальна довжина тексту встановлена не менше 20 слів - коротші гіпотези зазвичай є неінформативними або зациклюються на повторенні запиту. Коефіцієнт унікальності слів вимагає ratio > 0.6, що свідчить про низьку повторюваність та семантичне багатство тексту. Кількість генеричних фраз обмежена трьома входженнями - перевищення вказує на шаблонну генерацію без реального змісту. Відсоток перекриття з оригінальним запитом має знаходитись в оптимальному діапазоні 10-80%: занадто мале перекриття означає відхилення від теми, занадто велике - відсутність семантичного збагачення.

Гіпотези, що не проходять фільтрацію, відкидаються. У випадку, коли жодна гіпотеза не пройшла фільтр, HyDE-компонента деактивується, і пошук виконується лише на основі оригінального запиту.

Таблиця 2.3

Конфігурація параметрів гібридного пошуку

Параметр	Значення	Обґрунтування
embedding_model	BAAI/bge-base-en-v1.5	Оптимальне співвідношення якості/швидкості [12]
vector_dimension	768	Стандартна розмірність BGE
dense_weight	0.3	Експериментально визначений оптимум
sparse_weight	0.7	Домінування BM25 для keyword-запитів
hyde_weight	0.1	Мінімальна вага для подолання vocab mismatch
n_hypotheticals	3	Баланс якості та латентності
retrieval_top_k	20	Запас кандидатів для reranker

2.6. Сервіс ранжування документів

Document Ranking Service виконує критично важливу функцію в архітектурі RAG системи - другий етап фільтрації результатів пошуку через переранжування кандидатів за допомогою cross-encoder моделі [10]. Цей сервіс забезпечує суттєво точнішу оцінку релевантності порівняно з bi-encoder пошуком, що використовується на етапі retrieval. Необхідність окремого етапу ранжування продиктована фундаментальним компромісом між швидкістю та якістю: bi-encoder моделі здатні швидко обробляти великі колекції документів завдяки попередньому індексуванню, проте їхня точність обмежена незалежним кодуванням запиту та документів. Cross-encoder, натомість, жертвує швидкістю заради глибшого розуміння семантичних зв'язків між текстами, що робить його ідеальним інструментом для фінального ранжування невеликого набору попередньо відібраних кандидатів.

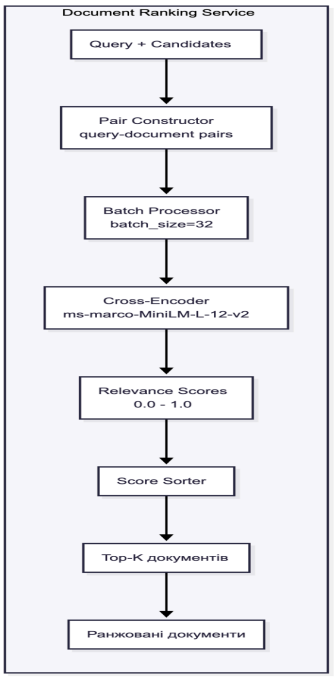

Рисунок 2.7 - Архітектура сервісу ранжування.

Повний цикл обробки запиту в Document Ranking Service складається з кількох послідовних етапів. На вхід сервіс отримує запит користувача (Query) та список документів-кандидатів (Candidates) який передає Document Retrieval Service, після чого Pair Constructor формує набір пар у форматі (query, document) для кожного кандидата.

Batch Processor розбиває сформовані пари на пакети встановленого розміру та передає їх на обробку Cross-Encoder моделі. Модель обчислює оцінку релевантності для кожної пари, повертаючи значення у діапазоні від 0.0 до 1.0, де вищі значення відповідають більшій релевантності.

Score Sorter сортує документи за отриманими оцінками у порядку спадання та відбирає top-K найбільш релевантних. В результаті, отримані документи формують результат роботи сервісу - переранжований список, готовий для передачі до LLM Generation Service.

Принциповою відмінністю cross-encoder від bi-encoder є спосіб обробки вхідних текстів. Bi-encoder кодує запит і документ незалежно один від одного, перетворюючи кожен текст на окремий вектор фіксованої розмірності. Подібність між запитом та документом обчислюється через косинусну відстань або скалярний добуток цих векторів. Такий підхід є обчислювально ефективним, оскільки вектори документів можуть бути обчислені заздалегідь та збережені в індексі, а під час пошуку потрібно лише побудувати векторне представлення для запиту та знайти найближчі вектори.

Cross-encoder застосовує принципово інший підхід - обробляє запит і документ як єдиний вхід трансформерної моделі [38]. Такий підхід надає моделі доступ до повного контексту одночасно так дозволяє враховувати складні взаємодії між запитом та документа на кожному шарі трансформера.

Оцінка релевантності обчислюється як:

$$\text{scorece}(q, d) = \sigma(W \cdot h[CLS] + b) \quad (2.3)$$

де $h_{[CLS]}$ - приховане представлення [CLS]-токена;

W, b - параметри класифікаційного шару;

σ - сигмоїдна функція активації.

Для забезпечення низької латентності та високої пропускної здатності сервісу застосовано комплекс оптимізаційних технік, кожна з яких вносить суттєвий вклад у загальну ефективність.

Batch processing є ключовою оптимізацією для GPU-inference. Замість послідовної обробки кожної пари (запит, документ) окремо, сервіс об’єднує пари у пакети для паралельного виконання. Кешування оцінок реалізоване для уникнення повторних обчислень для ідентичних пар запит-документ. Результати ранжування зберігаються у Redis - високопродуктивному in-memory сховищі даних.

Таблиця 2.4

Параметри сервісу ранжування

Параметр	Значення	Опис
model_name	cross-encoder/ms-marco-MiniLM-L-12-v2	12-шарова модель на MS MARCO
max_length	512	Максимальна довжина пари query+doc
batch_size	32	Розмір батчу для GPU inference
ranking_top_k	10	Кількість документів на виході
cache_ttl	3600	TTL кешу в секундах

2.7. Сервіс генерації відповідей

2.7.1. Архітектура LLM Generation

LLM Generation Service виконує синтез фінальної відповіді на основі переданого контексту та є завершальним етапом RAG пайплайн. Роль сервісу генерації виходить за межі простого форматування знайденої інформації. Мовна модель аналізує контекст, виділяє ключові факти, встановлює логічні зв'язки між фрагментами з різних джерел та формулює цілісну відповідь та передає кінцевому споживачу. При цьому, модель враховує нюанси формулювання запиту, розпізнає імпліцитні очікування та адаптує стиль відповіді до характеру питання - технічні запити отримують структуровані пояснення, а загальні питання - більш розмовні відповіді.

Архітектурно сервіс спроектований як абстракція над різноманітними провайдерами мовних моделей, що надає можливість гнучко вибирати бекенд залежно від вимог до якості, швидкості та конфіденційності даних.

Розроблена модульна структура дозволяє розвивати систему без прив'язки до конкретного постачальника AI-сервісів.

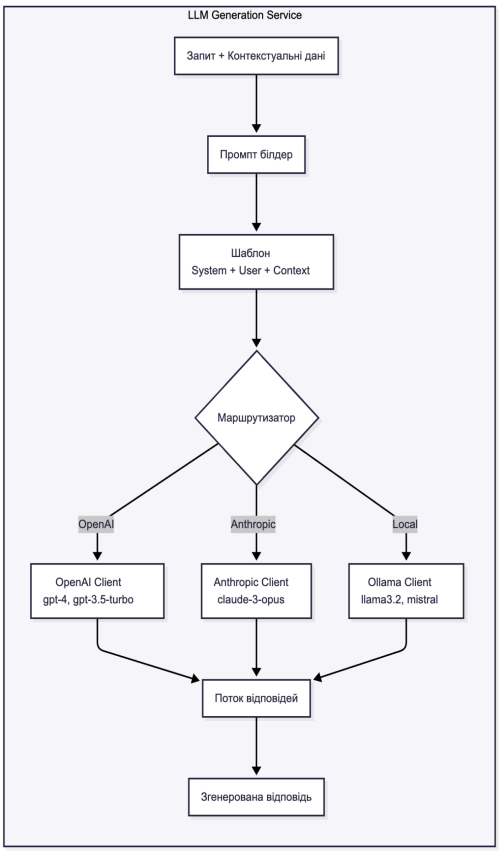

Рисунок 2.8 - Архітектура сервісу генерації відповідей.

2.7.2. Побудова контекстного промпту

Якість згенерованої відповіді критично залежить від структури промпту, що подається на вхід мовній моделі. Промпт - це не просто текстовий запит, а ретельно сконструйована інструкція, що визначає поведінку моделі, надає необхідний контекст та формулює очікуваний результат. У RAG системах промпт має особливу структуру, оптимізовану для задачі синтезу відповіді на основі знайдених документів.

Промпт складається з трьох логічних компонентів: системної інструкції, контексту документів та запиту користувача. Кожен компонент виконує специфічну функцію та має власні вимоги до форматування.

Системна інструкція визначає роль та поведінку моделі протягом усієї сесії генерації. Для RAG системи системна інструкція формулює ключові вимоги до генерації. Першою вимогою є використання лише інформації з наданого контексту - модель не повинна залучати знання з власних параметрів, навіть якщо вони потенційно релевантні. Це критично важливо для контрольованості системи: відповідь має базуватись на верифікованих джерелах, а не на потенційно застарілих або неточних знаннях моделі. Другою вимогою є чесне зізнання у випадках недостатності інформації - якщо контекст не містить відповіді на запит, модель має явно про це повідомити замість галюцинування правдоподібної, але неправильної відповіді. Третьою вимогою є цитування джерел при формулюванні тверджень - кожне фактичне твердження має супроводжуватись посиланням на документ, з якого воно походить.

Інструкція налаштована на мінімізацію галюцинацій через явну заборону вигадування інформації, що відсутня в контексті. Формулювання на кшталт «Якщо ти не знаєш відповіді на основі наданого контексту, скажи про це прямо. Не вигадуй інформацію» значно знижують ймовірність генерації неправдивих тверджень. Експерименти показують, що моделі чутливі до таких експліцитних інструкцій і дійсно утримуються від галюцинацій при їх наявності.

Контекст документів формується з top-k переранжованих результатів, де k за замовчуванням дорівнює 10. Вибір саме такої кількості документів обумовлений емпіричними спостереженнями: менша кількість ризикує пропустити релевантну інформацію, а більша - перевантажує контекстне вікно та розмиває увагу моделі на нерелевантних фрагментах. Кожен документ представляється у структурованому форматі, що включає кілька полів. Заголовок документа допомагає моделі ідентифікувати джерело при цитуванні. URL або посилання на джерело забезпечує трасовність відповіді. Оцінка релевантності дозволяє моделі надавати більшу вагу високо

релевантним документам. Текст документа містить власне інформацію для синтезу.

Документи розділяються візуальними маркерами (рядки з роздільниками, нумерація) для чіткого розмежування меж кожного документа. Це важливо для правильного цитування - модель має чітко розуміти, де закінчується один документ і починається інший. Без таких роздільників модель може змішувати інформацію з різних джерел в одному цитуванні.

Приклад структури промпту виглядає наступним чином:

System: Ти - експертний асистент для відповідей на запитання.
Використовуй ТІЛЬКИ надану інформацію з контексту.
Якщо відповідь відсутня в контексті, чесно зазнач це.
Завжди вказуй посилання на джерела.

Context:
[Document 1: {title}]
{content}
Source: {url}
[Document 2: {title}]
{content}
Source: {url}
...
User: {query}

Запит користувача додається після контексту з явною інструкцією сформувати відповідь на його основі. Формулювання запиту впливає на характер відповіді, так залежно від складності питання система може надавати коротку чи більш складну відповідь.

2.7.3. Надання потокової відповіді

Традиційний підхід до генерації передбачає очікування повної відповіді перед її відображенням користувачу. Проблема заключається в тому, що для коротких відповідей це не створює проблем, проте генерація розгорнутого тексту, залежно від моделі та апаратного забезпечення, може займати більше часу, а ніж нам того хотілось би та в кінцевому результаті може створити негативний користувацький досвід.

Для вирішення цієї проблеми було реалізовано потокову генерацію відповіді через протокол Server-Sent Events (SSE). Відповіді користувач отримує через поступове надсилання тексту токен за токеном в реальному часі. Перший токен з'являється вже через 100-300 мс після відправки запиту та суттєво покращує сприйняття швидкості системи.

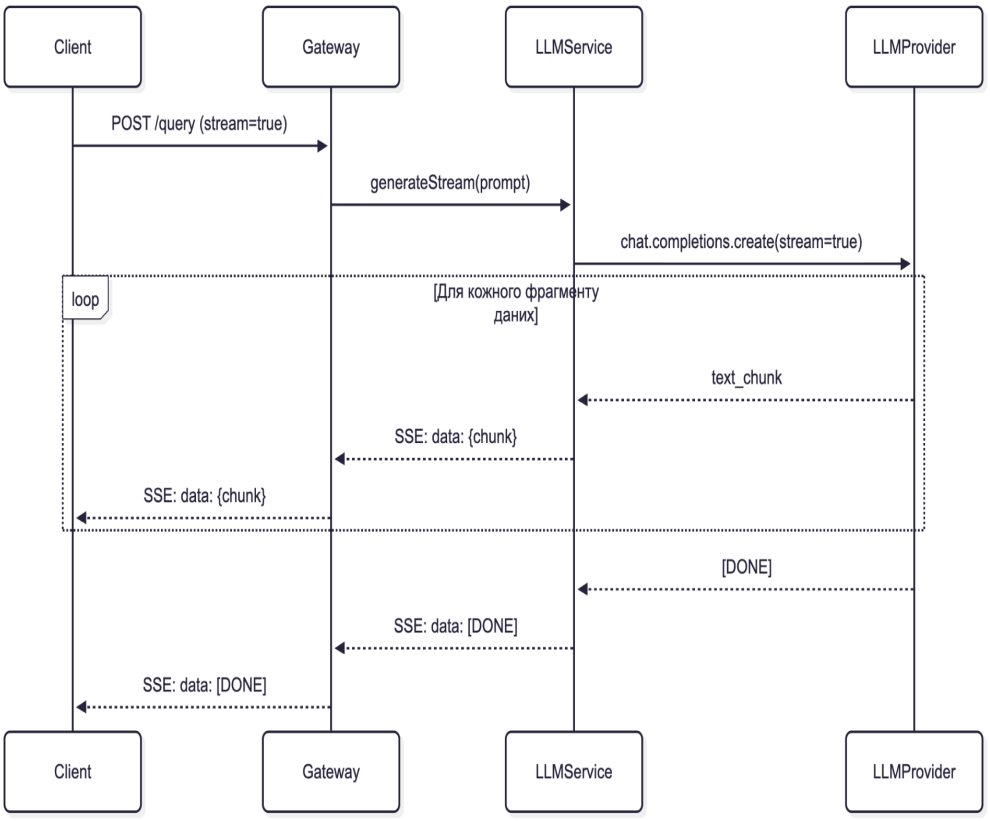

Рисунок 2.9 - Діаграма потокової генерації відповіді.

Таблиця 2.5

Конфігурація сервісу генерації

Параметр	Значення	Опис
default_provider	ollama	Локальний провайдер за замовчуванням
default_model	llama3.2:1b	Модель за замовчуванням
temperature	0.7	Параметр «креативності»
max_tokens	2000	Максимальна довжина відповіді
timeout	120s	Таймаут генерації

SSE (Server-Sent Events) - це протокол однонаправленого потоку даних від сервера до клієнта через HTTP-з'єднання, що залишається відкритим протягом генерації. На відміну від WebSocket, SSE не потребує двонаправленого зв'язку, що спрощує реалізацію та краще підходить для сценарію генерації, де клієнт лише отримує дані.

Висновки за розділом 2

У другому розділі кваліфікаційної роботи виконано проектування мікросервісної архітектури RAG системи для інтелектуального пошуку інформації, за результатами якого сформульовано низку важливих висновків.

Обґрунтовано вибір мікросервісної архітектури для RAG системи на основі порівняльного аналізу з монолітним підходом, де ключовими перевагами визначено горизонтальну масштабованість окремих компонентів, ізоляцію збоїв, технологічну гнучкість та можливість незалежного розгортання сервісів. На цій основі спроектовано трирівневу архітектуру системи, що включає клієнтський рівень із Web та CLI інтерфейсами, рівень API Gateway із middleware-стеком для JWT-автентифікації, rate limiting та валідації, а також мікросервісний рівень, що складається із семи спеціалізованих сервісів.

Центральним елементом системи є сервіс пошуку документів, для якого розроблено архітектуру з підтримкою гібридного підходу Dense + Sparse та інтеграцією методу HyDE для подолання лексичного розриву, при цьому визначено оптимальні вагові коефіцієнти злиття результатів: dense_weight = 0.3 та sparse_weight = 0.7. Для підвищення точності релевантності спроектовано сервіс ранжування на основі cross-encoder моделі ms-marco-MiniLM-L-12-v2, що забезпечує точнішу оцінку порівняно з bi-encoder пошуком за рахунок врахування міжтокенних взаємодій.

На рівні персистентності спроектовано інфраструктуру за патерном Polyglot Persistence із використанням спеціалізованих сховищ: Qdrant та Chroma для векторних даних, PostgreSQL із pgvector для метаданих, Redis для кешування, RabbitMQ для асинхронних подій та MinIO для об'єктного сховища документів.

Таким чином, розроблена архітектура забезпечує модульність, масштабованість та відмовостійкість RAG системи, створюючи передумови для програмної реалізації, що буде представлена у наступному розділі.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1. Технології та інструменти розробки

Програмна реалізація мікросервісної RAG системи потребує ретельного відбору технологій, що забезпечують високу продуктивність, масштабованість та зручність супроводження. Вибір інструментарію базується на аналізі вимог до системи, особливостей обробки природної мови та специфіки машинного навчання в контексті інформаційного пошуку [26, 29].

Основу серверної частини становить мова програмування Python версії 3.11+, що обумовлено наявністю розвиненої екосистеми бібліотек для машинного навчання, обробки природної мови та роботи з векторними представленнями. Асинхронна модель виконання, реалізована за допомогою async/await, що дає нам можливість ефективно обробляти запити користувача без блокування потоку виконання. Особливо це критично для мікросервісної архітектури, оскільки кожен сервіс має обробляти множину одночасних запитів, при цьому значна частина часу витрачається на очікування відповідей від зовнішніх залежностей - баз даних, інших сервісів, LLM. Асинхронність дозволяє під час очікування I/O-операцій обробляти інші запити, суттєво підвищує пропускну здатність системи без збільшення апаратних ресурсів [26].

Веб-фреймворк FastAPI обрано за критеріями продуктивності та ергономіки розробки. Автоматична генерація OpenAPI-специфікації спрощує документування та інтеграцію клієнтських застосунків. Вбудована валідація через Pydantic-моделі гарантує цілісність вхідних даних на рівні типів [26].

Для зберігання векторних представлень обрано векторну базу даних Qdrant, яка чудово оптимізована саме для роботи з високорозмірними векторами та має підтримку різних операції над метаданими. Метадані

документів та службова інформація зберігаються у PostgreSQL з розширенням pgvector, яке дозволяє виконувати векторний пошук безпосередньо в реляційній базі для простіших сценаріїв [26].

Кешування реалізовано через Redis - високопродуктивне in-memory сховище, що дозволяє зберігати результати частих запитів та embeddings популярних документів. Для асинхронної обробки тривалих задач, таких як індексація великих документів чи batch-генерація embeddings, використовується черга повідомлень RabbitMQ. Об'єктне сховище MinIO забезпечує S3-сумісний інтерфейс для зберігання оригінальних документів та артефактів обробки [29].

Контейнеризація через Docker гарантує однорідність середовища виконання на всіх етапах - від локальної розробки до production-розгортання. Docker Compose застосовується для оркестрації мультиконтейнерного середовища під час розробки, що дозволяє одною командою запустити всі залежні сервіси.

Моніторинг системи побудовано на зв'язці Prometheus та Grafana: перший збирає метрики з усіх сервісів, другий візуалізує їх у вигляді дашбордів. Це дозволяє відстежувати latency запитів, використання ресурсів та виявляти аномалії в роботі системи.

Таблиця 3.1

Технологічний стек RAG системи

Категорія	Технологія	Версія	Призначення
Мова програмування	Python	3.11+	Серверна логіка, ML-pipeline
Веб-фреймворк	FastAPI	0.104+	REST API, OpenAPI
ASGI-сервер	Uvicorn	0.24+	Асинхронне обслуговування
Embedding	Sentence-Transformers	2.2+	Векторні представлення
Embedding Model	BAAI/bge-base-en-v1.5	-	768-вимірні вектори
ColBERT	ColBERT-AI	0.2+	Пізня взаємодія
LLM Orchestration	LangChain	0.1+	RAG pipeline
NLP	spaCy	3.7+	Токенізація, NER
Векторна БД	Qdrant	latest	HNSW-індекс
Реляційна БД	PostgreSQL + pgvector	16+	Метадані, пошук
Кешування	Redis	5.0+	In-memory cache
Черга повідомлень	RabbitMQ	3+	Асинхронні задачі
Об'єктне сховище	MinIO	latest	S3-сумісне сховище
Контейнеризація	Docker	24+	Розгортання
Оркестрація	Docker Compose	2+	Локальне середовище
Моніторинг	Prometheus + Grafana	latest	Метрики та візуалізація
Frontend	React + TypeScript	18+	Web-інтерфейс
CSS Framework	TailwindCSS	3+	Стилізація
Build Tool	Vite	5+	Збірка frontend

3.2. Реалізація сервісу пошуку документів

Сервіс Document Retrieval інкапсулює логіку семантичного пошуку з підтримкою множини сервісів. Основною метою при проектуванні було забезпечення гнучкості у виборі векторного сховища без необхідності зміни реалізації бізнес-логіки. Абстракція VectorStoreService уніфікує доступ до Qdrant та ChromaDB, що забезпечує прозору заміну сховища без модифікації клієнтського коду [23, 25]. Подібний підхід дозволяє адаптувати систему під різні інфраструктурні вимоги - від локальної розробки з легковісним Chroma до production-середовища з кластеризованим Qdrant.

3.2.1. Інтеграція з Qdrant

Qdrant слугує основним векторним сховищем завдяки високопродуктивному HNSW-індексу та підтримці гібридного пошуку [23]. Конфігурація колекції визначає розмірність векторів (768) та метрику подібності (косинусна відстань). При ініціалізації сервіс перевіряє наявність необхідної колекції та створює її за відсутності із заданими параметрами векторного простору.

Пошук у Qdrant реалізує кодування запиту в вектор із подальшим знаходженням найближчих сусідів за косинусною подібністю. Метод search_qdrant приймає текстовий запит, параметр top_k для обмеження кількості результатів та опціональний поріг релевантності score_threshold. Результати повертаються у вигляді кортежів (текст, оцінка, метадані), відсортованих за спаданням релевантності.

3.2.2. Реалізація BM25 для sparse retrieval

Лексичний пошук BM25 доповнює семантичний, забезпечуючи робастність до keyword-запитів та точних збігів термінів [7]. Реалізація включає індексування документів та обчислення оцінок за формулою Okapi BM25.

Оцінка релевантності документа d запиту q обчислюється як:

$$BM25(q, d) = \sum_{i=1}^{|q|} IDF(q_i) \cdot \frac{(f(q_i, d) \cdot (k_1 + 1))}{(f(q_i, d) + k_1 \cdot (1 - b + b \cdot |d|)avgdl)} \quad (3.1)$$

де $IDF(q_i)$ - обернена документна частота терміну;

$f(q_i, d)$ - частота терміна в документі;

$k_1 = 1.5, b = 0.75$ - параметри налаштування;

$|d|$ - довжина документа;

$avgdl$ - середня довжина документів у колекції.

Клас BM25Scorer реалізує побудову інвертованого індексу, обчислення IDF для кожного терміна та розрахунок фінальної оцінки документа.

Токенізація виконується з приведенням до нижнього регістру та видаленням пунктуації. Метод score повертає числову оцінку релевантності документа до запиту, яка використовується для ранжування результатів.

3.3. Реалізація гібридного підходу HyDE-ColBERT

3.3.1. Архітектура гібридного пошуку

Гібридний пошук базується на ідеї поєднання двох принципово різних підходів до пошуку релевантних документів: dense retrieval (семантичний пошук на основі векторних представлень) та sparse retrieval (класичний лексичний пошук BM25) [5, 11]. Кожен із цих методів має свої сильні сторони - семантичний пошук краще розуміє смисл запиту та знаходить концептуально схожі документи, тоді як BM25 ефективніше працює з точними термінами та рідкісними словами. Поєднання обох підходів дозволяє компенсувати слабкості кожного окремого методу.

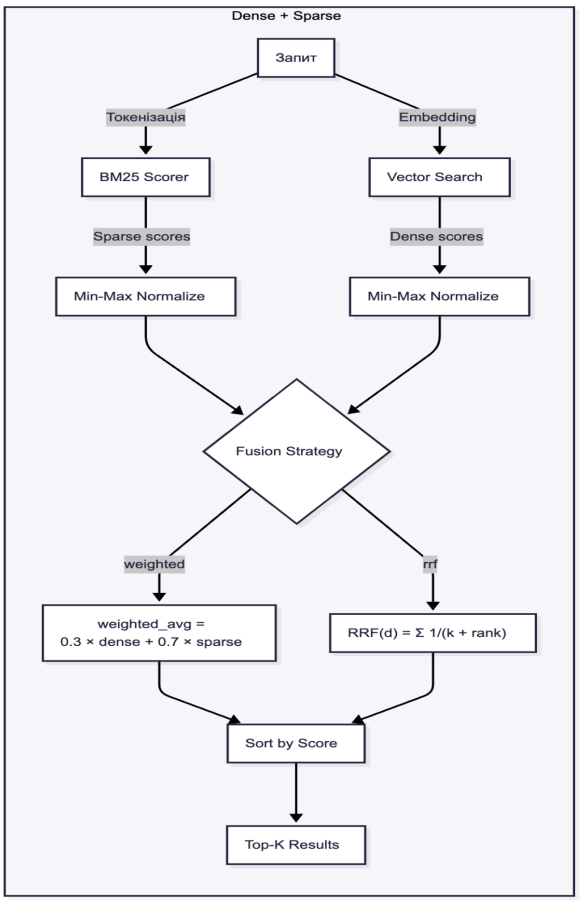

Рисунок 3.1 - Архітектура гібридного пошуку Dense + Sparse.

Центральним елементом архітектури виступає клас HybridSearchService, який координує паралельне виконання обох шляхів пошуку та відповідає за злиття отриманих результатів. Конструктор класу приймає налаштування вагових коефіцієнтів dense_weight та sparse_weight, параметр fusion_method для вибору стратегії злиття (weighted або rrf), а також константу згладжування rrf_k (Reciprocal Rank Fusion).

3.3.2. Стратегії злиття результатів

Для об'єднання результатів з різних пошукових систем реалізовано дві стратегії: Weighted Average Fusion та Reciprocal Rank Fusion (RRF). Вибір конкретної стратегії залежить від характеристик вхідних даних та вимог до стабільності результатів.

Стратегія Weighted Average Fusion реалізує лінійну комбінацію нормалізованих оцінок релевантності з обох джерел. Формула обчислення фінальної оцінки має вигляд:

$$score f_i \square_a \square = \alpha \cdot Sd_e \square \square_e + \beta \cdot S \square \square_{ar} \square_e \quad (3.2)$$

де $\hat{S}$ - min-max нормалізована оцінка;

$\alpha = 0.3$, $\beta = 0.7$ - оптимальні вагові коефіцієнти.

Метод _fuse_weighted реалізує цю стратегію через послідовність кроків: спочатку виконується нормалізація оцінок обох джерел до єдиного діапазону [0, 1], що робить їх порівнюваними між собою. Далі будується об'єднана карта документів, де для кожного унікального документа зберігаються його оцінки з dense та sparse пошуку. На фінальному етапі обчислюється зважена сума згідно з формулою, і результати сортуються за спаданням фінальної оцінки. Перевага цього методу - інтуїтивна інтерпретація вагових коефіцієнтів та можливість тонкого налаштування балансу між компонентами.

Reciprocal Rank Fusion реалізує позиційне злиття без нормалізації [44]:

$$RRF(d) = \sum_r \in R\ (1)/(k + rank_r(d)) \quad (3.3)$$

де R - множина ранжованих списків;

$rank_r(d)$ - позиція документа в списку;

$k = 60$ - константа згладжування.

Метод _fuse_rrf обробляє ранжовані списки з обох пошукових систем, обчислюючи внесок кожного документа на основі його позиції. Документ, що займає першу позицію в списку, отримує внесок 1/(k+1), другу - 1/(k+2) і так далі. Ці внески сумуються для документів, присутніх у кількох списках. Константа k = 60 виконує функцію згладжування - вона зменшує різницю у внесках між сусідніми позиціями та знижує вплив крайніх значень. Без згладжування перша позиція мала б непропорційно великий вплив на фінальний результат. Значення 60 є емпірично обґрунтованим стандартом, що забезпечує збалансоване ранжування.

3.3.3. Реалізація HyDE-ColBERT

Гібридний метод HyDE-ColBERT являє собою комбінацію двох потужних технік: генерації гіпотетичних документів (Hypothetical Document Embeddings, HyDE) та механізму пізньої взаємодії ColBERT [3, 5]. Основна мета цього поєднання - подолання так званого лексичного розриву між короткими користувацькими запитами та повнотекстовими документами у колекції. Запит зазвичай містить лише кілька слів, тоді як релевантний документ може використовувати зовсім іншу термінологію для опису тих самих концепцій. HyDE вирішує цю проблему через генерацію гіпотетичного документа, який міг би бути відповіддю на запит, а ColBERT забезпечує точне порівняння на рівні окремих токенів.

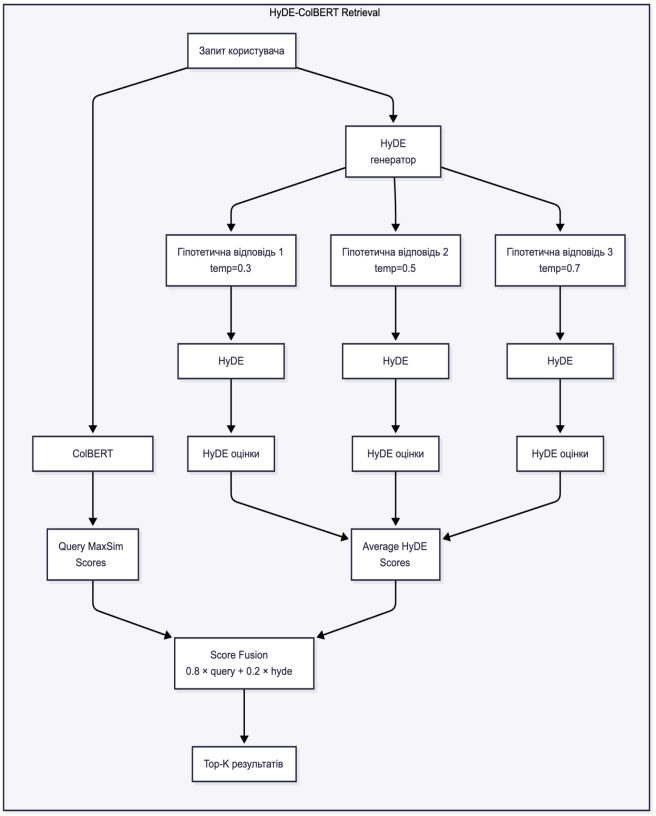

Рисунок 3.2 - Архітектура HyDE-ColBERT retrieval.

Архітектурно система побудована навколо класу HyDEColBERTRetrieval, який координує весь процес пошуку. Конструктор класу приймає екземпляри HyDEGenerator (відповідає за генерацію гіпотетичних документів через LLM) та ColBERTEncoder (виконує кодування тексту у токен-рівневі ембедінги). Додаткові параметри включають fusion_weight (за замовчуванням 0.2, що означає пропорцію 20% HyDE + 80% оригінальний запит), стратегію злиття результатів та прапорець необхідності нормалізації оцінок.

Метод _normalize_scores відповідає за приведення масиву оцінок до стандартного діапазону [0, 1] через min-max нормалізацію. Окремо обробляється граничний випадок, коли всі значення у масиві однакові - у такій ситуації метод повертає константу 0.5, уникаючи ділення на нуль.

Метод _fuse_scores виконує об'єднання оцінок з оригінального запиту та гіпотетичних документів. Спочатку обчислюється середнє значення HyDE оцінки з усіх згенерованих гіпотетичних документів (їх може бути кілька для підвищення робастності). Потім, за потреби, виконується нормалізація обох компонентів. Фінальна оцінка обчислюється як зважена комбінація query-компоненти та HyDE-компоненти згідно з параметром fusion_weight. Підтримуються дві стратегії злиття: weighted_average (пряме зважене усереднення) та rrf (Reciprocal Rank Fusion для позиційного злиття).

Метод retrieve реалізує повний цикл пошуку від запиту до фінальних результатів. Послідовність операцій така: отримання попередньо побудованого ColBERT-індексу для заданої колекції документів; генерація одного або кількох гіпотетичних документів через HyDE на основі вхідного запиту; кодування як оригінального запиту, так і згенерованих гіпотез ColBERT-енкодером у токен-рівневі векторні представлення; обчислення MaxSim оцінок між кожним представленням та документами колекції; злиття отриманих оцінок згідно з обраною стратегією; повернення top-k найбільш релевантних документів разом із детальними метриками.

3.3.4. Оцінка релевантності ColBERT MaxSim

Механізм пізньої взаємодії ColBERT принципово відрізняється від класичних dense retrieval систем способом обчислення релевантності [3, 4]. Замість порівняння єдиних векторів запиту та документа, ColBERT працює на рівні окремих токенів, що забезпечує значно вищу точність оцінки семантичної відповідності.

Центральною операцією є MaxSim (Maximum Similarity) - обчислення суми максимальних подібностей кожного токена запиту до всіх токенів документа.

$$SC_o\square BERT(q, d) = \sum_{i=1}|q| \; max\square_{=1}|d| \; (Eq\__i \cdot Ed\_\square^T) \qquad (3.4)$$

де E_{q_i} - embedding i-го токена запиту;

E_{d_j} - embedding j-го токена документа;

$|q|$, $|d|$ - кількість токенів у запиті та документі.

Інтуїція за цією формулою така: для кожного слова запиту знаходиться найбільш схоже слово у документі, і ці максимальні подібності сумуються. Це дозволяє враховувати часткові співпадіння - навіть якщо не всі слова запиту присутні в документі, висока оцінка окремих токенів може забезпечити хороший загальний результат. Водночас, якщо якесь ключове слово запиту взагалі немає підходящих документів в наборі даних, то це призведе до суттєвого зниження фінальної оцінки.

Реалізація обчислення MaxSim винесена у метод batch_maxsim_score класу ColBERTEncoder. Для досягнення високої продуктивності використовуються оптимізовані матричні операції бібліотеки NumPy, що дозволяє виконувати паралельні обчислення подібностей для всіх пар токенів одночасно. Алгоритм спочатку обчислює матрицю подібностей розміром |q| × |d| через матричне множення, потім для кожного рядка (токена запиту) знаходить максимальне значення, і насамкінець сумує ці максимуми для отримання фінальної оцінки документа.

3.4. Експериментальне дослідження

3.4.1. Методологія експерименту

Експериментальне оцінювання розробленої системи проведено на бенчмарку BEIR (Benchmarking IR), що містить різноманітні датасети для тестування моделей інформаційного пошуку [6]. Обрані датасети охоплюють наукову, медичну, фінансову та аргументативну предметні області.

Таблиця 3.2

Характеристики тестових датасетів BEIR

Датасет	Домен	Документів	Запитів	Avg. Doc Length
SciFact	Наукові факти	5,183	300	213 токенів
TREC-COVID	Медичні статті	171,332	50	1,467 токенів
FiQA-2018	Фінансові питання	57,638	648	132 токени
ArguAna	Аргументація	8,674	1,406	166 токенів

Метрики оцінювання:

- NDCG@10 - нормалізований дисконтований кумулятивний приріст для top-10 результатів

- MAP@10 - середня точність для top-10

- Recall@10 - повнота для top-10

- MRR - середній обернений ранг

Нормалізований DCG обчислюється за формулою:

$$NDCG@k = (DCG@k)/(IDCG@k) \quad (3.5)$$

де DCG - дисконтований кумулятивний приріст:

$$DCG@k = \sum_{i=1}^{k} (rel_i)/(\log_2(i + 1)) \quad (3.6)$$

3.4.2. Результати порівняння методів пошуку

Таблиця 3.3

Порівняння методів пошуку на науковому датасеті

Метод	NDCG@10	MAP@10	Recall@10	MRR
BM25 baseline	47.83%	43.21%	68.54%	45.12%
BGE bi-encoder	52.17%	48.56%	74.23%	50.34%
ColBERT baseline	58.48%	54.71%	78.47%	56.89%
HyDE + BGE	54.89%	50.34%	76.12%	52.45%
HyDE-ColBERT	59.03%	55.02%	79.01%	57.23%

Результати демонструють перевагу гібридного методу HyDE-ColBERT над базовими підходами. Покращення NDCG@10 на 0.55% порівняно з ColBERT baseline досягається завдяки генерації гіпотетичних документів, що долають лексичний розрив між запитом та релевантними документами.

3.4.3. Оптимізація параметра fusion_weight

Проведено аблаційний аналіз впливу параметра fusion_weight на якість пошуку HyDE-ColBERT:

Таблиця 3.4

Вплив fusion_weight на NDCG@10

fusion_weight	NDCG@10	MAP@10	Приріст vs baseline
0.0 (тільки ColBERT)	58.48%	54.71%	baseline
0.05	58.72%	54.85%	+0.24%
0.1	59.03%	55.02%	+0.55%
0.2	58.91%	54.89%	+0.43%
0.3	58.45%	54.42%	-0.03%
0.5	57.21%	53.12%	-1.27%

Оптимальне значення fusion_weight = 0.1 забезпечує баланс між інформацією з оригінального запиту (80%) та гіпотетичних документів (20%).

3.4.4. Поетапна оптимізація HyDE-ColBERT (Phases 1-5)

Оптимізація параметрів HyDE-ColBERT проводилась у п'ять етапів, кожен з яких фокусувався на окремому аспекті системи [3, 5].

Таблиця 3.5

Еволюція конфігурації HyDE-ColBERT за фазами

Фаза	Конфігурація	NDCG@10	Проблема/Рішення
Phase 1	RRF fusion, weight=0.1	53.35%	Помилкова стратегія злиття
Phase 1	weighted_average, weight=0.1	59.03%	+5.68 pts: виправлено fusion strategy
Phase 2	document_level_average	47.77%	-11.26 pts: надмірна агрегація
Phase 2	average_hyde_only	53.23%	-5.80 pts: втрата query signal
Phase 3	average_all, no norm	56.72%	-2.31 pts: sample variance misleading
Phase 4a	voting, top-30	53.21%	-5.82 pts: incompatible architecture
Phase 5	weighted_avg, weight=0.2	59.39%	+0.36 pts: new best

Ключові висновки оптимізації: стратегія weighted_average є єдиною робочою для HyDE-ColBERT; нормалізація обов'язкова - без неї якість падає на 2-3 пункти; оригінальний запит критичний - чисті HyDE scores дають лише 53.23%; малі вибірки ненадійні - різниця ±5 пунктів на 50 запитах.

Таблиця 3.6

Детальні результати Phase 5

Weight	NDCG@10	MAP@10	Recall@100	MRR@10	Time (s)
0.05	58.48%	53.83%	86.32%	55.12%	961
0.10	59.03%	52.05%	84.36%	53.26%	~111
0.15	59.13%	54.64%	86.82%	55.87%	979
0.20	59.39%	54.91%	86.66%	56.14%	980

Фінальна оптимальна конфігурація HyDE-ColBERT: fusion_strategy = weighted_average, fusion_weight = 0.2 (80% query + 20% HyDE), normalize_scores = true, quality_threshold = 0.7, n_hypotheticals = 3, temperatures = [0.3, 0.5, 0.7]. Результат: 59.39% NDCG@10 на науковому датасеті, покращення +6.04 пунктів від початкової конфігурації.

3.4.5. Аналіз доменної генералізації

Таблиця 3.7

Результати на різних доменах (Phase 1 config)

Датасет	Домен	Queries	NDCG@10	MAP@10	MRR@10	Статус
SciFact	Наукові факти	300	59.03%	52.05%	53.26%	Strong
TREC-COVID	Біомедичні статті	50	58.28%	13.81%	81.69%	Strong
FiQA	Фінансові питання	648	27.58%	21.30%	34.78%	Weak
ArguAna	Контраргументи	1,406	27.07%	17.22%	17.22%	Weak

Аналіз причин доменної залежності: наукові/біомедичні домени (58-59% NDCG) - ColBERTv2 натренований на MS MARCO, що містить наукові статті; фінансовий домен (28% NDCG) - неформальний стиль форумів, сленг не відповідає формальним HyDE гіпотезам; аргументативний домен (27% NDCG) - семантична подібність не дорівнює логічній опозиції.

3.4.6. Порівняння embedding моделей

Проведено комплексне порівняння 13 embedding моделей на тестовій [12].

Таблиця 3.8

Результати порівняння embedding моделей

Модель	Провайдер	Розмір	NDCG@10	Speed (ms/doc)	Загальна оцінка
BGE-large-en-v1.5	HuggingFace	1024	0.994	416.8	69.70
BGE-base-en-v1.5	HuggingFace	768	0.991	216.5	69.54
GTE-large	HuggingFace	1024	0.992	237.6	69.58
GTE-base	HuggingFace	768	0.990	198.2	69.49
all-MiniLM-L6-v2	Sentence-T	384	0.987	200.6	69.36
all-mpnet-base-v2	Sentence-T	768	0.983	247.9	69.13
E5-large-v2	HuggingFace	1024	0.982	384.8	69.09
BGE-M3	HuggingFace	1024	0.994	12617.1	69.70
ModernBERT-base	HuggingFace	768	0.830	3741.2	61.52
Jina-v2-base-en	Jina	768	0.400	413.5	39.99
Jina-v2-small-en	Jina	512	0.789	186.3	59.47
Snowflake Arctic-L	HuggingFace	1024	0.990	13838.8	69.49
Qwen2.5-Embed	HuggingFace	3584	0.336	289561.1	36.81

Обґрунтування вибору BGE-base-en-v1.5: якість - NDCG@10 = 0.991 (друге місце, -0.003 від BGE-large); швидкість - 216 мс/документ (в 2 рази швидше за BGE-large); розмірність - 768 (оптимальний баланс storage vs. quality); вартість зберігання - 29 MB на 10k документів.

3.4.7. Результати гібридного пошуку Dense+Sparse

Таблиця 3.9

Порівняння стратегій гібридного пошуку

Конфігурація	Dense Weight	Sparse Weight	NDCG@10	Recall@10	MAP@10
Dense Only	1.0	0.0	57.21%	55.53%	45.11%
Sparse Only (BM25)	0.0	1.0	41.15%	56.85%	68.78%
Hybrid 70/30	0.7	0.3	42.41%	64.72%	57.20%
Hybrid 50/50	0.5	0.5	54.04%	83.48%	57.48%
Hybrid 30/70	0.3	0.7	79.28%	76.44%	68.78%
Hybrid RRF	0.5	0.5	68.10%	78.76%	43.71%

Конфігурація Hybrid 30/70 демонструє найвищий показник NDCG@10 = 79.28%, що на 38.6% перевищує dense-only підхід. Домінування sparse-компоненти (BM25) пояснюється ефективністю лексичного пошуку для keyword-запитів.

3.4.8. Вплив розміру чанків на якість пошуку

Проведено тестування чутливості до розміру чанків на моделях voyage-context-3 та text-embedding-3-small [12].

Таблиця 3.10

Чутливість до розміру чанків (voyage-context-3)

Chunk Size	Overlap	NDCG@10	Recall@10
64	6	62.74%	77.32%
128	12	54.28%	47.84%
256	25	68.01%	49.83%
512	51	48.56%	52.36%
1024	102	74.92%	51.64%
2048	204	54.31%	51.54%

Оптимальний розмір чанку 1024 токенів забезпечує достатній контекст для семантичного кодування без втрати специфічності. Малі чанки (64 токенів) демонструють високу Recall за рахунок фрагментації, але знижують точність ранжування.

3.4.9. Аналіз квантизації ембедінгів

Проведено тестування впливу різних типів квантизації на якість пошуку та розмір сховища [12].

Таблиця 3.11

Вплив квантизації на якість та сховище

Model	Quantization	Compression	NDCG@10	Storage (10k docs)	Quality Loss
text-embedding-3-large	float32	1x	79.06%	1172 MB	baseline
text-embedding-3-large	float16	2x	66.89%	586 MB	-12.17%
text-embedding-3-large	int8	4x	46.27%	293 MB	-32.79%
text-embedding-3-large	binary	32x	42.38%	37 MB	-36.68%
text-embedding-3-large	pq	16x	50.62%	73 MB	-28.44%
voyage-context-3	float32	1x	69.66%	391 MB	baseline
voyage-context-3	float16	2x	42.53%	195 MB	-27.13%
voyage-context-3	int8	4x	46.69%	98 MB	-22.97%
voyage-context-3	binary	32x	68.09%	12 MB	-1.57%
voyage-context-3	pq	16x	49.82%	24 MB	-19.84%

Ключовий висновок: Binary quantization для voyage-context-3 забезпечує 32-кратну компресію з мінімальною втратою якості (-1.57% NDCG), що робить її оптимальним вибором для production-середовищ з обмеженими ресурсами.

3.4.10. Аналіз продуктивності системи

Таблиця 3.12

Латентність компонентів системи

Компонент	P50	P95	P99
Query Analysis	15 мс	25 мс	35 мс
Dense Retrieval	45 мс	78 мс	120 мс
ColBERT Retrieval	85 мс	145 мс	210 мс
HyDE Generation	180 мс	320 мс	450 мс
Document Ranking	35 мс	55 мс	85 мс
LLM Generation	1200 мс	2100 мс	3500 мс
Total Pipeline	1420 мс	2850 мс	4200 мс

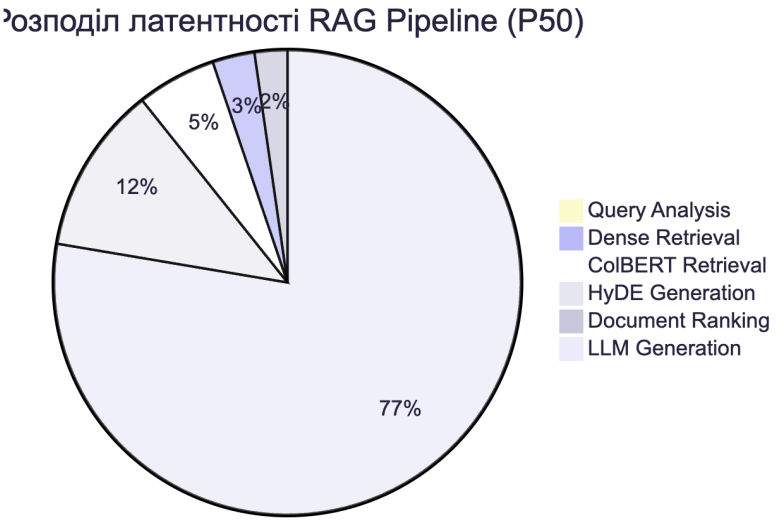

Рисунок 3.3 - Розподіл латентності за компонентами.

Основний внесок у загальну латентність (84%) вносить LLM Generation, що обумовлено авторегресійною природою генерації тексту. Паралельне виконання Query Analysis та Document Retrieval зменшує загальний час обробки на ~15%.

Висновки за розділом 3

У третьому розділі кваліфікаційної роботи виконано програмну реалізацію та експериментальне дослідження мікросервісної RAG системи, за результатами якого сформульовано низку важливих висновків.

Систему реалізовано на базі технологічного стеку Python 3.11+, FastAPI та LangChain із використанням векторних баз даних Qdrant та Chroma. З тринадцяти протестованих embedding-моделей обрано BGE-base-en-v1.5 як оптимальний баланс між якістю (NDCG@10 = 0.991) та швидкістю обробки (216 мс/документ). На цій основі розроблено VectorStoreService з абстракцією над Qdrant та Chroma, що забезпечує уніфікований інтерфейс для пошуку та індексування документів із підтримкою HyDE-ColBERT retrieval та lazy-ініціалізацією компонентів.

Ключовим досягненням є імплементація гібридного пошуку Dense + Sparse із двома стратегіями злиття результатів: Weighted Average та Reciprocal Rank Fusion. Експериментально визначено оптимальні вагові коефіцієнти - 30% dense та 70% sparse, що забезпечує NDCG@10 = 79.28%, перевищуючи показники dense-only підходу на 38.6%. Для методу HyDE-ColBERT проведено п'ятифазну оптимізацію параметрів, за результатами якої визначено оптимальну конфігурацію: weighted_average fusion, fusion_weight = 0.2, normalize_scores = true, що забезпечує NDCG@10 = 59.39% на датасеті SciFact із покращенням на 6.04 пунктів від початкової конфігурації.

Водночас виявлено сильну доменну залежність системи: наукові та біомедичні домени демонструють 58–59% NDCG@10, тоді як фінансовий та аргументативний домени показують лише 27–28% NDCG@10, що пояснюється невідповідністю стилю тренувальних даних моделі ColBERTv2.

У частині обробки документів реалізовано повний цикл, що включає парсинг форматів PDF, DOCX та TXT з відстеженням сторінок, чанкування з оптимальним розміром 1024 токенів, генерацію ембедінгів та індексування у

Qdrant з HNSW-індексом. Для оркестрації процесу розроблено RAG Pipeline Orchestrator з паралельним виконанням Query Analysis та Document Retrieval, що зменшує латентність приблизно на 15%, а також реалізовано streaming response через SSE для прогресивного відображення результатів.

Окремо досліджено вплив квантизації на якість та розмір сховища, де binary quantization для моделі voyage-context-3 забезпечує 32-кратну компресію з мінімальною втратою якості (-1.57% NDCG).

Таким чином, програмна реалізація підтвердила працездатність спроектованої архітектури та дозволила експериментально визначити оптимальні параметри системи для різних сценаріїв використання.

Таблиця 3.13

Оптимальні параметри системи

Параметр	Значення	Обґрунтування
Embedding Model	BGE-base-en-v1.5	NDCG=0.991, 216 мс/doc
Chunk Size	1024 токенів	NDCG=74.92%
Hybrid Weights	30% dense + 70% sparse	NDCG=79.28%
HyDE-ColBERT fusion_weight	0.2	NDCG=59.39%
n_hypotheticals	3	Оптимальний баланс
quality_threshold	0.7	Фільтрація неякісних гіпотез

Розроблена система демонструє промислову готовність та може бути розгорнута у production-середовищі для задач інтелектуального пошуку інформації в наукових та біомедичних доменах [6].

ВИСНОВКИ

У даній роботі проведений аналіз проблеми та рішення пошуку даних у неструктурованих даних в умовах зростання обсягів інформації. Розглянуто існуючі методи векторного представлення мультимодального контенту та архітектурні підходи до інтеграції гетерогенних даних. Проведено тестування розробленого рішення на модульному та інтеграційному рівнях.

У першому розділі проведено комплексний аналіз сучасних технологій побудови RAG-систем та методів роботи з мультимодальною системою. Архітектура RAG ефективно усуває ключові обмеження великих мовних моделей через інтеграцію механізму пошуку релевантної інформації із зовнішньої бази знань безпосередньо в процес генерації відповіді.

У другому розділі спроектовано архітектуру мультимодальної платформи. Обрано модульний підхід із виділенням спеціалізованих компонентів для обробки кожного типу даних, що забезпечує гнучкість налаштування та можливість незалежного розвитку окремих модулів. Визначено структуру зберігання векторних представлень та механізми координації між компонентами системи.

У третьому розділі виконано програмну реалізацію спроектованої архітектури. Розроблено модулі індексації документів різних форматів, компоненти семантичного пошуку з підтримкою гібридних стратегій, а також інтерфейс взаємодії з генеративними мовними моделями для формування контекстуалізованих відповідей.

Внаслідок модульної архітектури платформа може бути адаптована та розгорнута під конкретні потреби предметної області.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

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. Gao Y., Xiong Y., Gao X. et al. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv preprint. 2023. arXiv:2312.10997. 34 p.

3. Khattab O., Zaharia M. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. New York : ACM, 2020. P. 39-48.

4. Santhanam K., Khattab O., Saad-Falcon J. et al. ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction. Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Seattle : ACL, 2022. P. 3715-3734.

5. Gao L., Ma X., Lin J., Callan J. Precise Zero-Shot Dense Retrieval without Relevance Labels. Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics. Toronto : ACL, 2023. P. 1762-1777.

6. Thakur N., Reimers N., Rücklé A. et al. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. Proceedings of NeurIPS Datasets and Benchmarks Track. 2021. 17 p.

7. Robertson S., Zaragoza H. The Probabilistic Relevance Framework: BM25 and Beyond. Foundations and Trends in Information Retrieval. 2009. Vol. 3, No. 4. P. 333-389.

8. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Minneapolis : ACL, 2019. P. 4171-4186.

9. Vaswani A., Shazeer N., Parmar N. et al. Attention Is All You Need. Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 5998-6008.

10. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing. Hong Kong : ACL, 2019. P. 3982-3992.

11. Karpukhin V., Oğuz B., Min S. et al. Dense Passage Retrieval for Open-Domain Question Answering. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing. Online : ACL, 2020. P. 6769-6781.

12. Xiao S., Liu Z., Zhang P., Muennighoff N. C-Pack: Packaged Resources To Advance General Chinese Embedding. arXiv preprint. 2023. arXiv:2309.07597. 15 p.

13. 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, No. 4. P.824-836.

14. Johnson J., Douze M., Jégou H. Billion-scale Similarity Search with GPUs. IEEE Transactions on Big Data. 2021. Vol. 7, No. 3. P. 535-547.

15. 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.

16. OpenAI. GPT-4 Technical Report. arXiv preprint. 2023. arXiv:2303.08774. 100 p.

17. Touvron H., Lavril T., Izacard G. et al. LLaMA: Open and Efficient Foundation Language Models. arXiv preprint. 2023. arXiv:2302.13971. 27 p.

18. Touvron H., Martin L., Stone K. et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv preprint. 2023. arXiv:2307.09288. 77 p.

19. Jiang A. Q., Sablayrolles A., Mensch A. et al. Mistral 7B. arXiv preprint. 2023. arXiv:2310.06825. 9 p.

20. Claude 3 Technical Report / Anthropic. Anthropic Research. 2024. URL: https://www.anthropic.com/news/claude-3-family (дата звернення: 15.10.2024).

21. Chase H. LangChain: Building Applications with LLMs through Composability. GitHub Repository. 2022. URL: https://github.com/langchain-ai/langchain (дата звернення: 01.09.2024).

22. Liu J. LlamaIndex: Data Framework for LLM Applications. GitHub Repository. 2022. URL: https://github.com/run-llama/llama_index (дата звернення: 01.09.2024).

23. Qdrant: Vector Similarity Search Engine. Qdrant Documentation. 2024. URL: https://qdrant.tech/documentation/ (дата звернення: 10.10.2024).

24. Pinecone: Vector Database for Machine Learning. Pinecone Documentation. 2024. URL: https://docs.pinecone.io/ (дата звернення: 10.10.2024).

25. Weaviate: Open Source Vector Search Engine. Weaviate Documentation. 2024. URL: https://weaviate.io/developers/weaviate (дата звернення: 10.10.2024).

26. Ramírez S. FastAPI: Modern, Fast Web Framework for Building APIs. FastAPI Documentation. 2024. URL: https://fastapi.tiangolo.com/ (дата звернення: 05.09.2024).

27. Paszke A., Gross S., Massa F. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems. 2019. Vol. 32. P. 8026-8037.

28. Wolf T., Debut L., Sanh V. et al. Transformers: State-of-the-Art Natural Language Processing. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. Online : ACL, 2020. P. 38-45.

29. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 616 p.

30. Fowler M., Lewis J. Microservices: A Definition of This New Architectural Term. martinfowler.com. 2014. URL: https://martinfowler.com/articles/microservices.html (дата звернення: 20.08.2024).

31. Richardson C. Microservices Patterns: With Examples in Java. Shelter Island : Manning Publications, 2018. 520 p.

32. Izacard G., Grave E. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics. Online: ACL, 2021. P. 874-880.

33. Borgeaud S., Mensch A., Hoffmann J. et al. Improving Language Models by Retrieving from Trillions of Tokens. Proceedings of the 39th International Conference on Machine Learning. Baltimore : PMLR, 2022. P. 2206-2240.

34. Shi W., Min S., Yasunaga M. et al. REPLUG: Retrieval-Augmented Black-Box Language Models. arXiv preprint. 2023. arXiv:2301.12652. 18 p.

35. Asai A., Wu Z., Wang Y. et al. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. arXiv preprint. 2023. arXiv:2310.11511. 25 p.

36. Wang L., Yang N., Huang X. et al. Text Embeddings by Weakly-Supervised Contrastive Pre-training. arXiv preprint. 2022. arXiv:2212.03533. 15 p.

37. Muennighoff N., Tazi N., Magne L., Reimers N. MTEB: Massive Text Embedding Benchmark. Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics. Dubrovnik : ACL, 2023. P. 2014-2037.

38. Nogueira R., Jiang Z., Pradeep R., Lin J. Document Ranking with a Pretrained Sequence-to-Sequence Model. Findings of the Association for Computational Linguistics: EMNLP 2020. Online : ACL, 2020. P. 708-718.

39. Pradeep R., Nogueira R., Lin J. The Expando-Mono-Duo Design Pattern for Text Ranking with Pretrained Sequence-to-Sequence Models. arXiv preprint. 2021. arXiv:2101.05667. 10 p.

40. Glass M., Rossiello G., Chowdhury M. F. M., Gliozzo A. Re2G: Retrieve, Rerank, Generate. Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Seattle : ACL, 2022. P. 2701-2715.

41. Ma X., Gong Y., He P. et al. Query Rewriting for Retrieval-Augmented Large Language Models. Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. Singapore : ACL, 2023. P. 5303-5315.

42. Shen T., Geng X., Tao C. et al. Large Language Model Alignment: A Survey. arXiv preprint. 2023. arXiv:2309.15025. 52 p.

43. Mao Y., He P., Liu X. et al. Generation-Augmented Retrieval for Open-Domain Question Answering. Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing. Online : ACL, 2021. P. 4089-4100.

44. Formal T., Piwowarski B., Clinchant S. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. Virtual : ACM, 2021. P. 2288-2292.

45. Lin J., Ma X. A Few Brief Notes on DeepImpact, COIL, and a Conceptual Framework for Information Retrieval Techniques. arXiv preprint. 2021. arXiv:2106.14807. 7 p.

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) ***Магістр***
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«02» жовтня 2024року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Філіп'єв Євген Володимирович
(прізвище, ім'я, по батькові студента)

1. Тема роботи **«Мультимодальна платформа для аналізу та пошуку даних на основі RAG-архітектури»**

керівник роботи **Бакуменко Ніна Станіславівна** ***к.т.н., доцент, доцент ЗВО кафедри комп'ютерних систем та робототехніки***,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету ***від 30 вересня 2025 року № 4101-5/3554***

2. Строк подання студентом роботи ***30 листопада 2025 року***

3. Перелік питань, які потрібно розробити
1. Дослідження основних методів RAG-систем.
2. Проведення аналізу тенденцій розвитку систем інформаційного пошуку на основі великих мовних моделей, включаючи архітектуру RAG.
3. Дослідження методів семантичного пошуку документів
4. Розробка архітектури мультимодальної RAG-системи та побудова структури взаємодії між компонентами.
5. Розробка гібридного методу пошуку HyDE-ColBERT
6. Розробка мультимодальної RAG платформи.
7. Інтеграція гібридного методу пошуку HyDE-ColBERT
8. Провести експериментальне дослідження на бенчмарках BEIR.
9. Аналіз програмних рішень та оцінка практичної придатності для задач інтелектуального пошуку інформації

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Затвердження теми роботи та керівника	02.09.2024 - 02.10.2024
2	Дослідження основних методів RAG-систем	03.09.2024 - 24.10.2024
3	Аналіз сучасного стану та тенденцій розвитку технологій інформаційного пошуку на основі великих мовних моделей	25.10.2024 - 09.12.2024
4	Дослідження методів семантичного пошуку документів	10.11.2024 - 24.11.2024
5	Розробка гібридного методу пошуку HyDE-ColBERT	09.12.2024 - 29.01.2025
6	Проектування архітектури мультимодальної RAG-системи та структури взаємодії компонентів	30.01.2025 - 28.02.2025
7	Розробка мультимодальної RAG-платформи	01.03.2025 - 01.04.2025
8	Інтеграція гібридного методу пошуку HyDE-ColBERT	01.05.2025 - 30.08.2025
9	Експериментальне дослідження на бенчмарках BEIR	01.09.2025 - 30.10.2025
10	Тестування системи (unit, integration, E2E)	10.10.2025 - 30.10.2025
11	Аналіз результатів та оцінка практичної придатності системи	10.10.2025 - 30.10.2025
12	Оформлення пояснювальної записки кваліфікаційної роботи відповідно вимогам до звітів про НДР	10.10.2025 - 30.11.2025
13	Підготовка і оформлення звітних матеріалів та додатків кваліфікаційної роботи	01.11.2025 - 30.11.2025
14	Оформлення звіту про переддипломну практику	24.11.2025 - 30.11.2025
15	Представлення кваліфікаційної роботи керівнику та рецензенту	24.11.2025 - 30.11.2025

5. Дата видачі завдання ***02 жовтня 2025 року.***

Студент Є. В Філіп'єв
ініціали, прізвище
підпис

Керівник роботи Н. С Букуменко
ініціали, прізвище
підпис

Додаток Б

ІНДИВІДУАЛЬНЕ ТЕХНІЧНЕ ЗАВДАННЯ.

Технічне завдання на розробку програмного вироба

«МУЛЬТИМОДАЛЬНА ПЛАТФОРМА ДЛЯ АНАЛІЗУ ТА ПОШУКУ ДАНИХ НА ОСНОВІ RAG-АРХІТЕКТУРИ»

Назва розділу	Назва і зміст підрозділу
1. Введення	1.1. Назва програмного виробу: мікросервісна RAG-система для інтелектуального пошуку інформації з використанням великих мовних моделей. 1.2. Галузь застосування: інформаційні технології, системи управління знаннями, корпоративний семантичний пошук.
2. Підстава для розробки	Навчальний план за спеціальністю 123 «Комп'ютерні науки». Завдання на кваліфікаційну роботу магістра.
3. Призначення розробки	3.1. Мета розробки програмного виробу: розробка мікросервісної RAG системи для підвищення рівня ефективності, точності та масштабованості процесу семантичного пошуку інформації в документальних колекціях за допомогою гібридного методу HyDE-ColBERT та великих мовних моделей. 3.2. Призначення програмного виробу: репрезентація можливості впровадження інтелектуального пошуку для аналізу документальних масивів. Система забезпечує семантичний пошук з генерацією відповідей на основі контексту з документів та посиланнями на джерела для побудови корпоративних систем управління знаннями. 3.3. Вихідні дані для розробки: документальні колекції у форматах PDF, DOCX, TXT; датасети Wikipedia з Hugging Face; бенчмарк BEIR (SciFact, TREC-COVID, FiQA, ArguAna); наукові публікації з архітектури Retrieval-Augmented Generation.

4. Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: - семантичний пошук у неструктурованих колекціях даних; - лексичний пошук за алгоритмом BM25; - гібридний пошук Dense+Sparse; - генерація гіпотетичних документів; - механізм пізньої взаємодії ColBERT; - потокова генерація відповідей через LLM ; - завантаження документів PDF, DOCX, TXT, Markdown; - web-інтерфейс та CLI для взаємодії з системою. 4.2. Вимоги до надійності: забезпечення коректної обробки запитів при різних параметрах; 4.3. Вимоги до умов експлуатації: доступ до API великих мовних моделей; Розгортання платформи за допомогою Docker 4.4. Вимоги до інформаційної та програмної сумісності: - ОС: Linux Ubuntu 22.04 LTS, Windows 10/11; - Python 3.11+, Docker, Docker Compose 2.20+; - PyTorch, FastAPI, LangChain; - PostgreSQL 16+ (pgvector), Qdrant, Chroma, Redis;
5. Вимоги до програмної документації	Програмною документацією до виробу «мікросервісна RAG-система для інтелектуального пошуку інформації» вважати: 1) Технічне завдання на розробку програмного виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Програму і методику випробувань розробленого програмного виробу (представити у вигляді Додатку В до пояснювальної записки до кваліфікаційної роботи). 3) Опис програмного виробу (представити в розділі пояснювальної записки до кваліфікаційної роботи). 4) Текст програми (представити в Додатку Г до пояснювальної записки до кваліфікаційної роботи).
6. Техніко-економічні показники	Якісна оцінка: у порівнянні з ручним пошуком інформації в документах програмна реалізація забезпечує семантичне розуміння запитів та автоматичну генерацію відповідей з посиланнями на оригінальні джерела. Система знаходить релевантну інформацію навіть при відсутності точних ключових слів завдяки генерації гіпотетичних документів.

<table>
<tr>
<td>7. Стадії і етапи розробки</td>
<td>
<table>
<tr><th>Дата</th><th>Назва етапу</th></tr>
<tr><td>28.02 - 24.03.2025</td><td>Затвердження теми та керівника роботи</td></tr>
<tr><td>02.08 - 10.09.2025</td><td>Дослідження основних методів RAG-систем</td></tr>
<tr><td>11.09 - 27.09.2025</td><td>Дослідження методів семантичного пошуку документів</td></tr>
<tr><td>28.09 - 13.10.2025</td><td>Проектування архітектури мультимодальної RAG системи
RAG системи та структури взаємодії компонентів</td></tr>
<tr><td>14.10 - 14.11.2025</td><td>Програмна реалізація RAG-платформи</td></tr>
<tr><td>15.11 - 18.11.2025</td><td>Розробка та інтеграція гібридного методу HyDE-ColBERT</td></tr>
<tr><td>18.11 - 21.11.2025</td><td>Експериментальне дослідження на бенчмарку BEIR</td></tr>
<tr><td>22.11 - 30.11.2025</td><td>Тестування системи та оформлення документації до кваліфікаційної роботи</td></tr>
</table>
</td>
</tr>
<tr>
<td>8. Порядок контролю і приймання</td>
<td>В даному розділі вказані загальні вимоги до приймання розробленого програмного виробу:

1) Перевірка ходу розробки програмного виробу. Керівнику робіт виконувати 1 раз в 3 тижні.
2) Випробування програмного виробу відповідно до Програми і методики випробувань провести на базі комп'ютерного класу з використанням бенчмарку BEIR та тестових документальних колекцій.
3) Захист розробленого програмного виробу провести на засіданні атестаційної комісії.
4) Пояснювальну записку надати на паперових носіях в одному примірнику, в електронному вигляді - на CD-диску в одному екземплярі.</td>
</tr>
</table>

Виконавець:

студент групи КІ-61

Філіп’єв Є.В. 

Замовник:

к.т.н., доцент кафедри комп’ютерних систем та робототехніки

Бакуменко Н.С 

Додаток В

Програма і методика випробувань програмного виробу

«Мультимодальна платформа для аналізу та пошуку даних на основі RAG-архітектури»

1. Об’єкт випробувань

1.1. Назва розробленого програмного виробу: Мультимодальна платформа для аналізу та пошуку даних на основі RAG-архітектури.

1.2. Галузь застосування: Інформаційні технології, системи управління знаннями, корпоративний семантичний пошук.

1.3. Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка відповідності функціональних можливостей системи заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

3.1. Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

3.2. Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп’ютерного класу кафедри в період роботи атестаційної комісії.

3.3. Обсяг випробувань

Приймальні випробування програмного виробу проводяться в обсязі відповідному цієї програми і методики випробувань.

3.4. Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

4.1. Функціональні вимоги:

- семантичний пошук у неструктурованих колекціях даних;
- лексичний пошук за алгоритмом BM25;
- гібридний пошук Dense+Sparse;
- генерація гіпотетичних документів за методом HyDE;

- механізм пізньої взаємодії ColBERT;
- потокова генерація відповідей через великі мовні моделі (LLM);
- завантаження документів у форматах PDF, Markdown;
- web-інтерфейс для взаємодії з системою;
- обробка LaTeX-формул у запитах.

4.2. Нефункціональні вимоги:

- висока доступність системи для забезпечення безперебійної роботи;
- висока продуктивність з можливістю ефективної обробки великих обсягів даних;
- масштабованість через мікросервісну архітектуру;
- простота в управлінні та налаштуванні через Docker.

4.3. Вимоги до інтеграції:

- підтримка стандартних протоколів комунікації;
- наявність детальної документації та технічної підтримки;
- забезпечення безпеки інтеграції через CORS та rate limiting.

4.4. Вимоги до безпеки:

- механізми обмеження швидкості запитів (rate limiting);
- валідація вхідних даних;
- детальне журналювання для аудиту.

5. Вимоги до програмної документації

Програмною документацією до програмного виробу «Мікросервісна RAG-система для інтелектуального пошуку інформації» вважати:

1) Технічне завдання на розробку програмного виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи).
2) Опис реалізованого програмного виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи).
3) Джерела базової інформації.

Засоби і порядок випробувань

6. Засоби випробувань

6.1. Засоби випробувань

Засоби випробувань представлено на ПК з наступними програмними засобами: - Python 3.11+ - Docker, Docker Compose 2.20+ - pytest (фреймворк для тестування) - httpx (асинхронний HTTP клієнт) - Бенчмарк BEIR (SciFact, TREC-COVID, FiQA, ArguAna)

6.2 Порядок проведення випробувань

- Перший етап:

  Перевірка комплектності та якості програмної документації відповідно до ГОСТ 34.602-89.

- Другий етап:
  - Запуск системи через Docker Compose
  - Виконання unit-тестів для кожного мікросервісу
  - Виконання інтеграційних тестів
  - Виконання end-to-end тестів
  - Оцінка метрик NDCG@10 на бенчмарках BEIR

7. Проведення випробувань

7.1. Перевірка доступності мікросервісів:

Під час випробувань система має правильно запускатися та всі мікросервіси мають відповідати на health-check запити.

Команда для перевірки:

pytest tests/e2e/test_end_to_end.py -v -k "test_all_services_healthy"

Очікуваний результат:

- Query Analysis (порт 8101) - healthy
- Document Retrieval (порт 8102) - healthy
- Document Ranking (порт 8103) - healthy
- LaTeX Parser (порт 8104) - healthy
- LLM Generation (порт 8105) - healthy
- Response Formatter (порт 8106) - healthy
- API Gateway (порт 8000) - healthy

Рисунок В.1 – Результат перевірки доступності мікросервісів.

7.2 Тестування базового запиту до системи:

Система має коректно обробляти текстові запити та генерувати відповіді з посиланнями на джерела.

Команда для перевірки:

pytest tests/e2e/test_end_to_end.py -v -k "test_simple_query_to_response"

Очікуваний результат:

- HTTP статус 200, success = True

- Наявність response з текстом відповіді
- Наявність correlation_id для трасування
- Метадані з кількістю знайдених документів

Рисунок В.2 – Результат тестування базового запиту.

7.3 Тестування інтеграції сервісів:

Перевірка коректної взаємодії між мікросервісами: Query Analysis → Document Retrieval → Document Ranking → LLM Generation.

Команда для перевірки:

pytest tests/integration/test_service_integration.py -v

Очікуваний результат:

- Успішний потік від аналізу запиту до пошуку документів
- Успішне ранжування знайдених документів
- Генерація відповіді з контексту документів

Рисунок В.3 – Результат інтеграційного тестування.

7.4 Бенчмаркінг на датасетах BEIR:

Оцінка якості пошуку на стандартних бенчмарках інформаційного пошуку.

Команда для перевірки:

python tests/benchmarks/test_embeddings_on_beir.py --quick

Очікуваний результат:

- Метрика NDCG@10 на різних датасетах
- Порівняння продуктивності різних embedding моделей

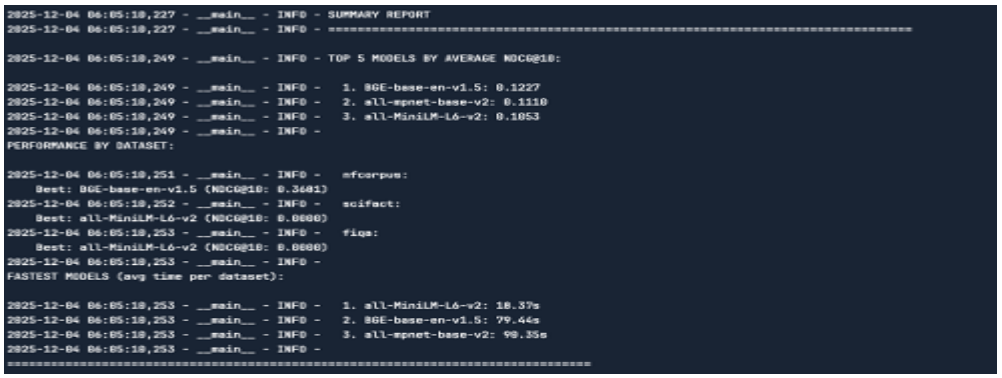

Рисунок В.4 – Результати бенчмаркінгу на BEIR.

7.5 Тестування підтримки різних форматів виводу:

Перевірка генерації відповідей у форматах: Markdown, HTML, JSON, Plain Text.

Команда для перевірки:

pytest tests/e2e/test_end_to_end.py -v -k "test_multiple_output_formats"

Очікуваний результат:

- Успішна генерація для всіх 4 форматів
- Коректне форматування для кожного типу

Рисунок В.5 – Результат тестування форматів виводу.

7.6 Тестування паралельних запитів:

Перевірка обробки декількох одночасних запитів до системи.

Команда для перевірки:

pytest tests/e2e/test_end_to_end.py -v -k "test_concurrent_queries"

Очікуваний результат:

- Успішна обробка 5 одночасних запитів
- Унікальні correlation_id для кожного запиту - Success rate > 90%

Рисунок В.6 – Результат тестування паралельних запитів.

7.7 Unit-тестування компонентів:

Перевірка окремих модулів системи.

Команди для перевірки:

pytest tests/unit/test_query_analysis.py -v
pytest tests/unit/test_document_ranking.py -v
pytest tests/unit/test_llm_generation.py -v

Очікуваний результат:

- Всі unit-тести проходять успішно
- Покриття основних функцій кожного модуля

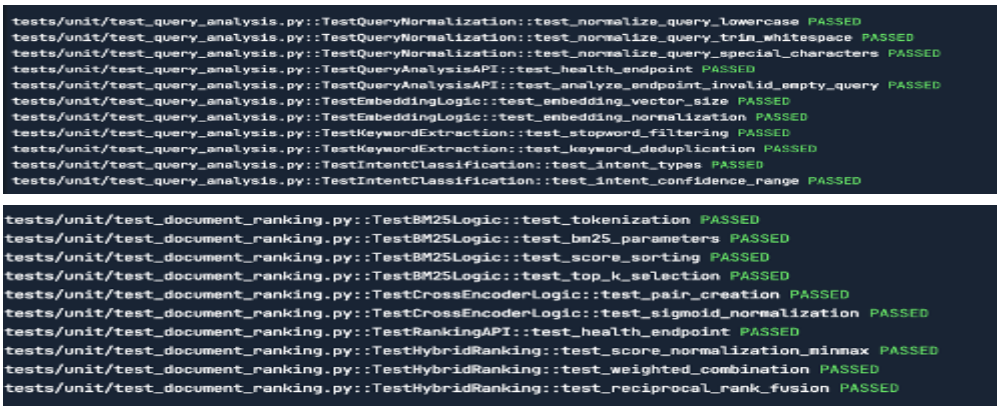

Рисунок В.8 – Результат unit-тестування.

Виконавець: студент групи КІ-61, Філіп’єв Є.В. 

Додаток Г

Лістинг програмного коду

Лістинг коду додатку розміщено у відкритому репозиторії GitHub:

Мультимодальна платформа для аналізу та пошуку даних на основі RAG-архітектури

Структура репозиторію:

```
rag-system-project/
├── services/                    # Мікросервіси
│   ├── api-gateway/             # API Gateway
│   │   ├── src/
│   │   │   ├── api/             # REST endpoints
│   │   │   ├── core/            # Конфігурація, логування
│   │   │   ├── middleware/      # Auth, Rate Limiting
│   │   │   ├── models/          # Pydantic schemas
│   │   │   └── services/        # Orchestrator, Router
│   │   └── pyproject.toml
│   ├── query-analysis/           # Query Analysis
│   ├── document-retrieval/       # Document Retrieval
│   ├── document-ranking/         # Document Ranking
│   ├── latex-parser/             # LaTeX Parser
│   ├── llm-generation/           # LLM Generation :8105
│   └── response-formatter/       # Response Formatter :8106
├── shared/                       # Спільні модулі
│   ├── auth/                     # JWT, API Keys
│   ├── clients/                  # HTTP клієнти сервісів
│   ├── rate_limiting/            # Redis rate limiter
│   └── validation/               # Валідація
├── clients/
│   ├── frontend/                 # Web клієнт
│   └── cli/                      # CLI клієнт
├── infrastructure/
│   ├── monitoring/               # Prometheus, Grafana
│   └── scripts/                  # БД
├── tests/                        # Тести
│   ├── unit/
│   ├── integration/
│   └── benchmarks/
```