ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістр

на тему: **«Інтеграція технік пошуково-підкріпленої генерації та великих мовних моделей з базою знань компанії»**

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи та технології
спеціальності 126 Інформаційні системи та технології
ступеня вищої освіти магістр
групи 126ІСТ_мд_2024
Шишлін Владислав Олександрович
Керівник: Слюсар Вадим Іванович
Рецензент: Муравльов Володимир В’ячеславович

Полтава – 2025 року

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

Освітня програма Інформаційні управляючі системи та технології
Спеціальність 126 Інформаційні системи та технології
Рівень вищої освіти другий (магістерський)

ЗАТВЕРДЖУЮ
Завідувач кафедри
________________ Юрій УТКІН
«08» листопада 2024 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Шишліна Владислава Олександровича

1. Тема кваліфікаційної роботи:
«Інтеграція технік пошуково-підкріпленої генерації та великих мовних моделей з базою знань компанії»,
Керівник роботи: д. т. н., професор, професор кафедри інформаційних систем та технологій Слюсар Вадим Іванович.
Затверджено наказом закладу вищої освіти від «31» жовтня 2025 року № 1332-ст

2. Строк подання здобувачем вищої освіти роботи «09» грудня 2025 р.

3. Вихідні дані до роботи: наукові джерела наукометричних баз, дані інтернет-ресурсів, Large Language Model, Retrieval-Augmented Generation, Fine-tuning LLM, LoRA, QLoRA, Vector DB, VectorDBBench, Milvus, Qdrant

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Розділ 1 Аналіз особливостей використання пошуково-підкріпленої генерації
Розділ 2. Формування технік пошуково-підкріпленої генерації для інтеграції з базою знань компанії
Розділ 3. Рекомендації щодо використання технік пошуково-підкріпленої генерації для інтеграції

5. Перелік графічного матеріалу: схеми, рисунки, діаграми за темою та об’єктом дослідження.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання отримав
Оцінювання економічної ефективності результатів дослідження	Каліниченко О. В., к. е. н., доцент, доцент кафедри економіки та публічного управління	24.11.2025	04.12.2025

7. Дата видачі завдання «08» листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1.	Вибір і затвердження теми роботи	29.10.2024 р.	
2.	Складання та погодження розгорнутого плану та завдання на кваліфікаційну роботу	30.10.2024 р. – 08.11.2024 р.	
3.	Опрацювання джерел інформації	11.11.2024 р. – 27.12.2024 р.	
4.	Збір, вивчення і обробка інформації, необхідної для виконання роботи	30.12.2024 р.– 19.01.2025 р.	
5.	Виконання теоретико-методологічного розділу роботи	17.02.2025 р.– 16.05.2025 р.	
6.	Виконання дослідницько-аналітичного розділу роботи	02.06.2025 р.– 13.07.2025 р.	
7.	Виконання проєктно-рекомендаційного розділу роботи	08.09.2025 р.– 14.11.2025 р.	
8.	Оцінювання економічної ефективності результатів дослідження	24.11.2025 р.– 04.12.2025 р.	
9.	Оформлення тексту роботи	05.12.2025 р.– 08.12.2025 р.	
10.	Попередній захист роботи на кафедрі	09.12.2025 р.	
11.	Доопрацювання роботи з урахуванням зауважень і пропозицій	10.12.2025 р.- 14.12.2025 р.	
12.	Нормоконтроль	15.12.2025 р. – 16.12.2025 р.	
13.	Захист кваліфікаційної роботи	18.12.2025 р.	

Здобувач вищої освіти **Владислав ШИШЛІН**

Керівник роботи **Вадим СЛЮСАР**

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ, ПРАВА ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

ШИШЛІН ВЛАДИСЛАВ ОЛЕКСАНДРОВИЧ

«ІНТЕГРАЦІЯ ТЕХНІК ПОШУКОВО-ПІДКРІПЛЕНОЇ ГЕНЕРАЦІЇ ТА ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ З БАЗОЮ ЗНАНЬ КОМПАНІЇ»

Освітньо-професійна програма
Інформаційні управляючі системи та технології
Спеціальність 126 Інформаційні системи та технології
Ступінь вищої освіти Магістр

РЕФЕРАТ
кваліфікаційної роботи на здобуття кваліфікації –
магістр з інформаційних систем та технологій

Полтава – 2025 року

Кваліфікаційна робота складається із вступу, 3 розділів, висновків, списку використаних джерел (41 найменування), додатків. Кваліфікаційна робота містить 1 таблицю, 15 рисунків, викладена на 71 сторінці.

Основний зміст роботи

У першому розділі «Аналіз особливостей використання пошуково-підкріпленої генерації» розглянуто класичну схему RAG та кейси практичного застосування RAG. Визначено особливості використання RAG для LLMз метою впровадження певних механізмів зниження галюцінацій. На їх основі виконаний порівняльній аналіз RAG і fine-tuning.

У другому розділі «Формування технік пошуково-підкріпленої генерації для інтеграції з базою знань компанії» сформовано номенклатуру технік RAG для інтеграції з базою знань компанії, детально розглянуто операції з векторними БД, визначені пріоритетні техніки RAG. Окрема увага приділяється перспективній концепції Agentic RAG.

У третьому розділі «Рекомендації щодо використання технік пошуково-підкріпленої генерації для інтеграції» розглянуто можливості практичної реалізації запропонованих рішень обґрунтовані рекомендації щодо використання технік RAG для інтеграції в IT-інфраструктуру компанії. Виконано оцінку ефективності технік RAG та запропоновано рекомендації щодо вибору векторної БД. Детальна увага приділяється економічному обґрунтуванню прийнятих рішень.

Висновки

Ефективність RAG значною мірою залежить від коректного налаштування технічних елементів, зокрема параметрів чанкінгу, методів пошуку та прийомів роботи з великими текстами. Важливу роль відіграють точність ретривера та релевантність вибраних фрагментів, що потребує системного тестування та аналізу метрик. За умови правильної конфігурації RAG-система демонструє стабільну й передбачувану роботу у практичних застосуваннях. Узагальнено принципи формування ембедингів та показано їх ключову роль у семантичному пошуку. Операції над векторами забезпечують швидке опрацювання даних, а вибір типу індексу визначає баланс між точністю та швидкодією. Доведено, що векторні бази даних є основою сучасних RAG-проєктів, оскільки надають гнучкий і продуктивний доступ до знань. Систематизація технік побудови RAG показала, що якість отриманого контексту залежить від ефективності векторизації, схеми сегментації даних і механізмів реранжування. Методи покращення запитів та комбіновані стратегії пошуку зменшують семантичний розрив між питанням і джерелами та підвищують точність генерації. Agentic RAG продемонстрував подальший розвиток підходу, забезпечивши динамічний добір інформації та

адаптивну поведінку системи відповідно до потреб користувача. Узагальнено підходи до вибору векторних баз даних та показано, що їхня ефективність визначається масштабом даних, вимогами до швидкодії й доступною інфраструктурою. Тестування за допомогою VectorDBBench дозволило об'єктивно оцінити вплив фільтрації, потокових оновлень та індексації на продуктивність систем. Підкреслено важливість гібридного пошуку, квантизації та інтеграції з реляційними СУБД у складних RAG-сценаріях. Запропоновані рекомендації допомагають коректно обрати архітектуру та підтверджують необхідність тестування на власних даних. Аналіз технік RAG продемонстрував відмінності в їхній результативності та чутливість до параметрів сегментації, реранжування і способів обробки контексту. Метрики RAGAS, BertScore і ROUGE показали переваги підходів HyDE, Fusion Retrieval і Reranking RAG, тоді як інші методи виявили обмеження, пов'язані з шумом або структурою даних. Результати підкреслюють, що правильний вибір техніки та конфігурації ретривера є ключовою умовою надійної роботи RAG-систем. Економічна оцінка свідчить, що впровадження RAG-рішень є фінансово виправданим і сприяє суттєвому скороченню витрат на пошук інформації та експертну підтримку. Встановлено, що RAG-технології є ефективним інструментом цифрової трансформації та підвищення продуктивності бізнес-процесів.

Таким чином, результатами роботи є систематизація технік пошуково-підкріпленої генерації для інтеграції з базою знань компанії; оцінка ефективності використання технік пошуково-підкріпленої генерації; рекомендації щодо використання технік пошуково-підкріпленої генерації для інтеграції з базою знань компанії. Вони можуть бути використані для застосування в системах підтримки прийняття рішень, службах технічної підтримки, консультаційних сервісах та в проектах цифрової трансформації та подальших досліджень за даною тематикою.

Достовірність результатів роботи базується на досконалому виборі моделі RAG та апробованих методів досліджень.

Список публікацій здобувача

1. Силантьєв В.М., Шишлін В.О. Інтеграція генеративного AI і RAG для вирішення питань логістики. *Сучасні аспекти та перспективні напрямки розвитку науки: матеріали X Міжнародної студентської наукової конференції* (жовтень 2025 р. м. Луцьк), 2025. С. 235, 236.

2. Шишлін В. Застосування механізму Re-Ranking для завдань пошуку на основі LLM у корпоративних базах знань. *Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій: матеріали XXII щорічного міждисциплінарного семінару* (листопад 2025 р., м. Полтава), 2025. С. 114, 115.

АНОТАЦІЯ

Шишлін В. О. «Інтеграція технік пошуково-підкріпленої генерації та великих мовних моделей з базою знань компанії». Кваліфікаційна робота на правах рукопису.

Кваліфікаційна робота на здобуття ступеня вищої освіти магістр за освітньо-професійною програмою Інформаційні управляючі системи та технології спеціальності 126 Інформаційні системи та технології. Полтавський державний аграрний університет, Полтава, 2025.

Систематизовано техніки RAG для інтеграції з базою знань компанії; проведена оцінка ефективності використання технік RAG; розроблено рекомендації щодо використання технік RAG для інтеграції з базою знань компанії. Вони можуть бути використані для застосування в системах підтримки прийняття рішень, службах технічної підтримки, консультаційних сервісах та в проектах цифрової трансформації та подальших досліджень за даною тематикою.

Ключові слова: GenAI, LLM, галюцинація LLM, ChatGPT, AI-агент, Chatbot, RAG, Agentic RAG, векторна БД.

ANNOTATION

Shyshlin V. O. "Integration of Retrieval-Augmented Generation Techniques and Large Language Models with a Company Knowledge Base." Qualification work on manuscript rights.

Qualification work for obtaining a master's degree of higher education under the educational and professional program Information management systems and technologies specialty 126 Information systems and technologies. Poltava State Agrarian University, Poltava, 2025.

RAG techniques for integration with the company's knowledge base have been systematized; the effectiveness of using RAG techniques has been assessed; recommendations for using RAG techniques for integration with the company's knowledge base have been developed. They can be used for application in decision support systems, technical support services, consulting services, and in digital transformation projects and further research on this topic.

Keywords: GenAI, LLM, hallucination LLM, ChatGPT, hallucination LLM AI agent, Chatbot, RAG, Agentic RAG, vector DB.

ЗМІСТ

ВСТУП ………………………………………………………………………. 6
РОЗДІЛ 1. АНАЛІЗ ОСОБЛИВОСТЕЙ ВИКОРИСТАННЯ ПОШУКОВО-ПІДКРІПЛЕНОЇ ГЕНЕРАЦІЇ ……………………………….. 8
1.1 Особливості використання великих мовних моделей …………. 8
1.2 Класична пошуково-підкріплена генерація ……………………... 12
1.3 Кейси застосування пошуково-підкріпленої генерації ………… 17
1.4 Порівняльній аналіз RAG і fine-tuning …………………………… 20
Висновки до розділу 1 ……………………………………………….. 24
РОЗДІЛ 2. ФОРМУВАННЯ ТЕХНІК ПОШУКОВО-ПІДКРІПЛЕНОЇ ГЕНЕРАЦІЇ ДЛЯ ІНТЕГРАЦІЇ З БАЗОЮ ЗНАНЬ КОМПАНІЇ ……….. 26
2.1 Особливості практичної реалізації пошуково-підкріпленої генерації ……………………………………………………………………. 26
2.2 Операції з векторними БД …………………………………………. 29
2.3 Визначення технік пошуково-підкріпленої генерації ………….. 34
2.4 Концепція Agentic RAG ……………………………………………….. 46
Висновки до розділу 2 ……………………………………………….. 49
РОЗДІЛ 3. РЕКОМЕНДАЦІЇ ЩОДО ВИКОРИСТАННЯ ТЕХНІК ПОШУКОВО-ПІДКРІПЛЕНОЇ ГЕНЕРАЦІЇ ДЛЯ ІНТЕГРАЦІЇ ……….. 51
3.1 Формування рекомендацій щодо вибору векторної БД ……….. 51
3.2 Оцінка ефективності технік пошуково-підкріпленої генерації .. 54
3.3 Економічне обґрунтування прийнятих рішень ………………….. 60
Висновки до розділу 3 ……………………………………………….. 63
ВИСНОВКИ ……………………………………………………………….. 65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ …………………………………. 67
ДОДАТКИ ………………………………………………………………….. 71

ВСТУП

Актуальність теми кваліфікаційної роботи підтверджується необхідністю адаптації до корпоративних баз знань великих мовних моделей (LLM). Для цього може використовуватись пошуково-підкріплена генерація (RAG). Однак, питання її інтеграції з корпоративною базою знань для удосконалення систем корпоративного інформаційного пошуку потребують додаткових досліджень. Все це свідчить про актуальність теми роботи.

Зв'язок роботи з науковими програмами, темами. Робота відповідає дослідженням в межах науково-дослідної ініціативної тематики «Організаційно-методологічні аспекти впровадження інформаційно-комунікаційних систем і технологій в управлінні діяльністю сучасних організацій та підприємств за умов переходу до цифрової економіки» (ДРН 0123U105060, 2023-2028 рр.), що реалізується на кафедрі інформаційних систем та технологій, тематиці досліджень навчально-дослідної лабораторії інтелектуальних систем, комп'ютерних мереж та інтернет речей кафедри інформаційних систем та технологій Полтавського державного аграрного університету.

Метою кваліфікаційної роботи є підвищення ефективності використання корпоративної бази знань за рахунок інтеграції технік пошуково-підкріпленої генерації та великих мовних моделей.

Завданнями кваліфікаційної роботи є:

- проаналізувати особливості використання LLM та RAG;
- систематизувати техніки RAG;
- виконати оцінку ефективності застосування технік RAG;
- обґрунтувати рекомендації щодо використання технік RAG для інтеграції з базою знань компанії.

Об'єктом дослідження є процес використання LLM у взаємодії з корпоративними інформаційними системами та базами знань.

Предметом дослідження є техніки RAG, їх архітектури та механізми інтеграції з векторними базами даних у завданнях покращення інформаційного пошуку на підприємстві.

Методами дослідження для створення систематизації технік RAG і економічного обґрунтування прийнятих рішень використовувався аналітичний метод досліджень, а для оцінки ефективності використання технік RAG – моделювання.

Інформаційна база кваліфікаційної роботи сформована з ресурсів, що містять інформацію про векторні БД, LLM, RAG а також інструментарій для дослідження RAG.

Елементи наукової новизни роботи полягають у систематизації технік пошуково-підкріпленої генерації для інтеграції з базою знань компанії; оцінці ефективності використання технік пошуково-підкріпленої генерації.

Практична значущість роботи полягає в розробці рекомендацій щодо використання технік пошуково-підкріпленої генерації для інтеграції з базою знань компанії, які можуть бути використані для застосування в системах підтримки прийняття рішень, службах технічної підтримки, консультаційних сервісах та в проектах цифрової трансформації та подальших досліджень за даною тематикою.

Апробація результатів відбувалася в рамках X Міжнародної студентської наукової конференції «Сучасні аспекти та перспективні напрямки розвитку науки» (жовтень 2025 р., м. Луцьк) та XXII щорічного міждисциплінарного семінару «Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій» (листопад 2025 р., м. Полтава).

Структура кваліфікаційної роботи логічно пов’язана з завданнями досліджень і містить вступ, три розділи основної частини, висновки, список використаних джерел, додатки. Загальний обсяг пояснювальної записки кваліфікаційної роботи складає 71 сторінку формату А4. Вона містить 15 рисунків і 1 таблицю.

РОЗДІЛ 1
АНАЛІЗ ОСОБЛИВОСТЕЙ ВИКОРИСТАННЯ ПОШУКОВО-ПІДКРІПЛЕНОЇ ГЕНЕРАЦІЇ

1.1 Особливості використання великих мовних моделей

На даний час, в багатьох компаніях використовуються великі мовні моделі (Large Language Model, LLM) [1]. При цьому спостерігаємо появу альтернатив стандартним LLM – від моделей текстової дифузії до останніх гібридних архітектур з лінійною увагою (рис. 1.1) [2]. Деякі з них націлені на підвищення ефективності, інші, такі як моделі світу на основі коду, прагнуть поліпшити якість моделювання.

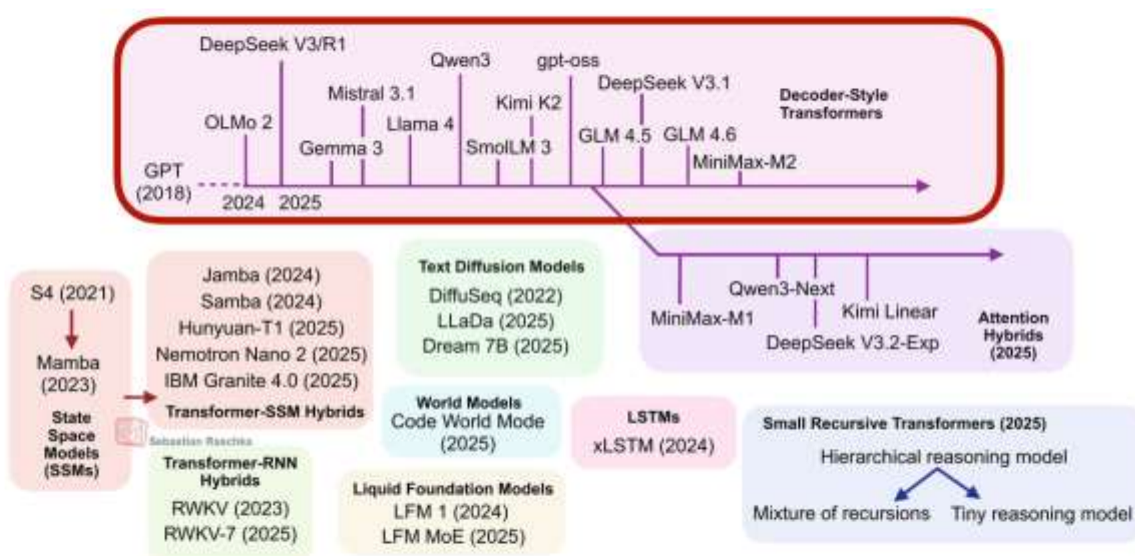

Рисунок 1.1 – Різноманіття архітектур LLM

В цілому, генерація відповідей LLM відбувається виключно з урахуванням інформації та шаблонів, засвоєної під час навчання (те, що знає сама модель). Однак такі моделі обмежені тими даними, на яких вони були навчені, що призводить до відомої проблеми галюцинацій – створення неправдивих відповідей з погляду фактології або логіки. Основні причини галюцинацій включають декілька чинників.

1. Дані для навчання. LLM навчаються на великих та різноманітних наборах даних з множини джерел. Це навчання відбувається без нагляду, що ускладнює перевірку даних на справедливість та точність. Моделі не можуть самостійно відрізнити правду від вигадки. Вони ґрунтуються на шаблонах, знайдених у даних, які можуть містити суперечливу або суб'єктивну інформацію. Це призводить до відповідей, що виглядають логічно, але фактично невірні.

2. Відсутність точності у специфічних областях. LLM, такі як GPT, розроблені для виконання спільних завдань. При застосуванні у конкретних галузях, таких як медицина, юриспруденція чи фінанси, вони можуть давати неправильні результати. Галюцинації виникають через спробу моделі створити зв'язну відповідь без достатніх знань у цій галузі.

3. Неправильне складання запитів (Bad Prompting). Користувачі взаємодіють із моделлю через запити (prompts). Написання чітких і точних запитів дуже важливо, як у програмуванні. Якщо запит недостатньо деталізований або контекстуальний, модель може згенерувати нерелевантну або неправильну відповідь. Щоб отримати правильні результати, важливо ставити правильні питання.

4. Умисна упередженість. Або, простіше кажучи, модель навчена «брехати». Це особливо вірно для загальнодоступних моделей, таких як ChatGPT [3] або Gemini [4]. Вони навчені з урахуванням багатьох сумнівних політик, щоб відповідати законам і регуляціям, що призводить до безглуздих відповідей на чутливі для суспільства теми.

5. Втрата контексту. Модель має обмежене вікно контексту або короткочасну пам'ять. Якщо розмова занадто довга або обсяг даних занадто великий, модель починає втрачати деталі контексту та генерувати безглузді результати. Я вважаю, що цю проблему буде вирішено у найближчому майбутньому зі збільшенням вікна контексту, що у свою чергу пов'язане з доступною обчислювальною потужністю. Однак, на даний момент це залишається проблемою, і важливо правильно проектувати системи RAG,

щоб оптимізувати використання контексту. Чи можна усунути галюцинації? Коротка відповідь: ні. Однак існують способи зменшити ймовірність їх виникнення:

– додаткове навчання моделі на релевантних даних із конкретної області.

– використання RAG (Retrieval-Augmented Generation) [5] для надання валідного контексту, уникаючи перевантаження контексту.

– інженерія запитів (Prompt Engineering) для створення запитів, які явно інструктують модель, не додавати інформацію, в якій вона не впевнена.

– точне налаштування в процесі запиту (In-prompt tuning) за допомогою few-shot прикладів.

– використання стратегій самоперевірки LLM, таких як «Дерево думки» (Tree of Thought), для перевірки логіки та результатів моделі.

– прохання до моделі повернути розподіл токенів з їхніми ймовірностями, щоб оцінити, наскільки впевнена модель у відповіді. Якщо можливості низькі (що вважається низьким, залежить від завдання), модель, швидше за все, почала фантазувати.

– валідація результатів за допомогою інших моделей або оракулів.

– постобробка та фільтрація, щоб цензурувати відповіді, які не відповідають політиці компанії або можуть бути шкідливими.

Проте, жоден із цих методів не може гарантувати повну відсутність галюцинацій. Як наслідок, коли LLM тільки-но з'явилися, їх обов'язково донавчали (fine-tuning) [6]. Під fine-tuning мається на увазі донавчання LLM на спеціалізованому датасеті, щоб модель краще вирішувала завдання у певній області. Однак тренувальні дані швидко можуть стати неактуальними – за день можуть з'явитися нові терміни та тренди, про які LLM не знає, а сама LLM не може вийти в БД та в Інтернет [7]. Як наслідок, такий підхід мав недоліки.

Для класичного fine-tuning доводиться збирати та розмічати датасет, підбирати гіперпараметри та використовувати потужні графічні процесори

для навчання. Така адаптація моделі під галузевий корпус може розтягнутися на тижні та вимагати значних витрат на хмарні обчислення – рахунок іде на тисячі доларів. Так, існують легші методи, наприклад, LoRA [8] або QLoRA [9]. Вони донавчають модель більш економно, змінюючи лише частину параметрів або зберігаючи поправки в адаптерах. Це зменшує навантаження та дозволяє працювати з меншими ресурсами. Іноді достатньо однієї-двох карт GPU та кількох годин. Однак навіть з такими прийомами fine-tuning залишається складним та ресурсомістким процесом, особливо якщо порівнювати його з RAG [10] або іншими підходами, де не потрібне втручання у ваги моделі. Модель навчається на даних, доступних на момент тренування. Згодом частина відомостей втрачає актуальність, а модель продовжує спиратися на старі знання. Щоб оновити контент, її доводиться навчати заново або донавчати на нових даних, що, як говорилося вище, займає багато часу і коштує дорого. В результаті без свіжої інформації страждає фактологічність відповідей, LLM може видавати формально зв'язний текст, але зі застарілими або неточними даними. Якщо у модель не вбудований механізм доступу до зовнішніх даних, то додавання навіть тисяч нових документів у базу нічого не змінить, вона їх просто не побачить. Щоб увімкнути свіжу інформацію, модель доводиться навчати заново, фактично починаючи процес тренування з нуля. У результаті будь-яке велике оновлення перетворюється на повноцінний цикл навчання: збір датасету, підготовка інфраструктури та тижня роботи на GPU. Саме тому класичний fine-tuning погано масштабується. Коли переналаштовуємо модель на новому датасеті, є небезпека того, що вона почне гірше пам'ятати те, що знала раніше. Нові знання хіба що затирають частину старих. Наприклад, вивчивши модель на текстах однієї тематики, можна зіпсувати її відповіді інших темах. Щоб донавчати, потрібно розбиратися в глибокому навчанні, архітектурі моделей, вміти готувати датасети, оцінювати результати.

Перш ніж витрачати ресурси на навчання величезної LLM, компанії все частіше вибирають більш легкі шляхи: інженерію промптів, підключення

зовнішніх баз знань або використання інструментів на зразок LangChain. У багатьох випадках цього достатньо, щоб вирішити задачу без важких та дорогих циклів навчання. З іншого боку, для надспеціалізованих чи критично важливих сценаріїв класичний fine-tuning залишається затребуваним. Майбутнє, швидше за все, за комбінацією 3-ох підходів:

– коли потрібно підмішувати свіжі факти, то використовувати пошуково-підкріплену генерацію (Retrieval Augmented Generation, RAG);

– коли важлива форма – задавати стиль та тональність через промпт-інжиніринг,

– а там, де без навчання не обійтися, використовувати точковий fine-tuning, найчастіше у форматі економічних технік (LoRA або QLoRA).

Таке поєднання дозволяє одночасно тримати актуальність знань, керувати стилем та зберігати контроль над вузькими завданнями, не роздмухуючи вартість експлуатації моделей до небес. Як наслідок, надалі доцільно розглянути RAG.

1.2 Класична пошуково-підкріплена генерація

RAG – це архітектурний підхід до генеративних моделей, що поєднує навички пошуку інформації з генеративними можливостями LLM. Ідея RAG була запропонована у 2020 р., щоб подолати обмеження LLM – замкнутість на знаннях з навчальних даних [11]. Замість спроб «вживити» всі знання параметри моделі, RAG підхід дозволяє моделі запитувати актуальні відомості із зовнішніх джерел (баз знань) під час генерації відповіді. Це забезпечує більш точні та актуальні відповіді, що спираються на факти, а не лише на згадку про модель.

Основна ідея RAG системи – розділити завдання відповіді на два етапи: пошук релевантної інформації та генерація відповіді з урахуванням знайдених даних. Таким чином, RAG архітектура складається з двох

ключових компонентів: retriever (пошуковий модуль) та generator (генеративна LLM модель) [12].

1. Індексація даних (offline): Перед тим, як RAG система зможе відповідати на запити, необхідно підготувати базу знань у зручному для пошуку вигляді. Для цього вихідні дані (документи, статті, довідкові посібники тощо) проходять етап інгестації та індексування. Спочатку дані завантажуються за допомогою конекторів або завантажувачів документів (наприклад, PDF, бази даних, веб сторінки) – цей крок підтримується безліччю інструментів. Потім документи розбиваються фрагменти оптимального розміру.

Розбиття необхідно, щоб довгі тексти перетворити на дрібніші логічні шматки – абзаци або пропозиції, які легше ефективно шукати і які помістяться в контекст вікна LLM.

Правильне розбиття – важлива практика, де зазвичай прагнуть до того, щоб кожен фрагмент був самодостатнім за змістом і містив цілісну думку. Часто використовують перекриття між фрагментами, щоби важливий контекст не обривався на кордоні.

Після цього кожен фрагмент пропускається через модель ембеддингу – це може бути передбачена нейромережа (наприклад, трансформер типу BERT/SBERT, або API на кшталт OpenAI Embeddings [13]), яка перетворює текст на векторне уявлення у просторі чисел.

Ці вектори (ембеддінги) відбивають семантичний зміст тексту: близькі за змістом фрагменти мають близькі вектори. Нарешті, всі ембеддінги зберігаються у векторній базі даних (vector store) – спеціалізованому сховищі, оптимізованому для пошуку схожих векторів за метрикою близькості.

Таким чином, формується індекс знань: він дозволяє швидко на запит знаходити фрагменти тексту, близькі за змістом до запиту, без перебору всіх документів.

Етапи індексації зазвичай виконуються оффлайн заздалегідь і можуть періодично оновлюватися в міру надходження нових даних.

2. Пошук (retrieval) [14] при запиті: коли користувач ставить питання системі, починається онлайн етап. Система бере текст запиту і перетворює його на вектор тим самим способом, як індексувалися документи (за допомогою тієї ж моделі ембеддингу). Через війну запит представлений як вектор у тому просторі, як і вектори документів. Потім виконується пошук найближчих векторів у векторній базі – тобто знаходження найбільш релевантних фрагментів знань за змістовною схожістю із запитом.

Зазвичай, використовується метричний пошук (косинусна близькість, евклідова відстань або dot product) для вибору top K фрагментів з найбільшою схожістю.

Сучасні векторні БД, наприклад, Qdrant або Milvus, здатні виконувати такий пошук за мільйонами та мільярдами ембеддингів із мілісекундними затримками за рахунок спеціалізованих структур (як HNSW графи) та оптимізації під обчислення близькості. Також на цьому кроці можуть застосовуватись фільтри або re ranking. Наприклад, спочатку знаходять кандидати за векторами, а потім перебудовують їх порядок більш точним, але повільним алгоритмом або застосовують додаткові критерії (дата, джерело тощо). Підсумком кроку Retrieval є набір K фрагментів (контекстів) з бази знань, які найімовірніше містять інформацію відповіді питання.

3. Генерація відповіді (Generation): Нарешті, знайдені фрагменти знань разом із вихідним запитом користувача формують розширений промпт LLM генератора. Генеративна модель (наприклад, GPT-4, LLaMA [15] або інша Seq2Seq модель) отримує на вхід текст запиту плюс доданий контекст (наприклад: «Питання: Контекст: Відповідь:»). LLM генерує відповідь, спираючись як у свої внутрішні знання (параметри), і на надані зовнішні дані. По суті модель навчена продовжувати текст, тому вона враховує допоміжну інформацію, яку ми їй «підклали», і видає більш фактологічний і детальний результат. В ідеалі генератор переформулює знайдені факти своїми словами, складаючи зв'язну відповідь, що максимально точно відповідає питанням. RAG тим самим розширює знання моделі в режимі

реального часу: якщо LLM спочатку не знала чогось (наприклад, свіжу статистику або внутрішні дані компанії), вона може отримати ці відомості через retriever і коректно включити їх у відповідь.

Цінність RAG полягає в тому, що LLM доповнюється актуальними даними та не «галюцинує» застарілі факти. Таким чином, RAG об'єднує два світи – інформаційний пошук та генерацію тексту. Retriever виступає як зовнішній довгостроковий «пам'ятний блок», а генератор – як «мозок», що вміє розмірковувати та формулювати відповідь природною мовою. RAG тримає знання у явній формі (база документів) і звертається до них за потребою. Це робить відповіді більш точними, актуальними та підкріпленими джерелами. Коли RAG комбінує генеративні моделі з пошуковими системами або БД для генерації відповідей збагаченими зовнішніми даними, класична схема відповідає рис. 1.2.

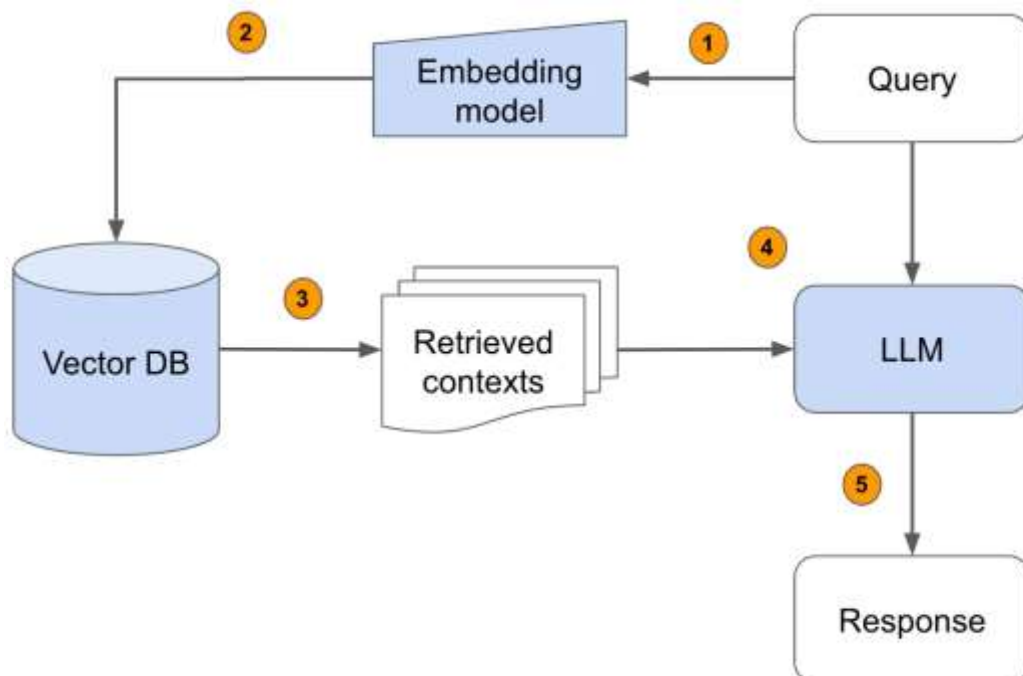

Рисунок 1.2 – Класичне представлення RAG

В даному контексті виконується наступна послідовність.

1. Подаємо текстовий запит у pipeline.
2. Векторизуємо текстовий запит за допомогою методів векторизації.

3. За вектором запиту шукаємо у БД N текстових документів з близькою векторною відстанню (чим ближче вектори, тим схожі запит та документ з відповіддю).

4. Отримані документи можемо реранжувати для покращення релевантності.

5. Отримані документи разом із запитом інтегруємо у промпт.

6. Генеруємо відповідь щодо сформованого промпту.

Таку систему з усіма етапами, які проговоримо нижче, можна швидко розгорнути на прикладах з Llama Index [16] або Langchain [17]. При цьому підготовки бази реалізується за допомогою наступної послідовності кроків.

1. Вся база знань «нарізається» на невеликі фрагменти тексту, звані chunks (чанки) [18]. Розмір цих chunks може змінюватись від декількох рядків, до декількох абзаців, тобто приблизно 100 до 1000 слів.

2. Далі ці чанки оцифровуються за допомогою «ембедера» і перетворюються на ембединги або іншими словами вектору, деякі набори чисел. Вважається, що в цих числах зашитий прихований зміст всього чанка, і саме за цим змістом і можна шукати.

3. Далі всі ці отримані вектори складаються в спеціальну базу даних, де лежать і чекають поки що над ними і почнуть робити ту саму операцію пошуку (найбільш релевантних, тобто близьких за змістом чанків пошуковим запитом).

4. Коли користувач відправляє своє питання в LLM, то текст його запиту точно за таким же алгоритмом (як правило тим самим ембеддером), кодується також в ще один ембеддинг і далі над базою даних містить наші ембеддинги чанков проводиться пошук найближчих «за змістом» ембеддингов (векторів). В реальності, як правило, вvраховується косинусна близькість вектору запиту та вектору кожного чанка і далі вибираються топ векторів N найбільш близьких до запиту. Косинусна близькість (cosine similarity) – це міра схожості між двома векторами, що показує, наскільки вони спрямовані в один бік. Вона не враховує абсолютну величину (довжину)

векторів, а дивиться лише на кут між ними. Таким чином, вона не залежить від масштабу тексту (довгі чи короткі документи); добре працює в високовимірних просторах ембеддингів; показує саме смислову схожість, а не схожість величин.

5. Далі текст чанків, який відповідає цим знайденим векторам, разом із запитом користувача об'єднується в єдиний контекст і подається на вхід мовної моделі. тобто модель «думає», що користувач написав їй не лише питання, але ще й надав дані на основі яких потрібно відповісти на поставлене запитання.

1.3 Кейси застосування пошуково-підкріпленої генерації

За останні два роки RAG має статус ключової технології для впровадження AI в корпоративні програми та наукові дослідження. Тому доцільно розглянути кілька типових сфер, де RAG вже довела свою корисність.

1. Чат-боти з доступом до бази знань (Virtual Assistants). RAG значно посилює можливості чат-ботів відповідати на складні питання користувачів. Замість обмеженого сценарію із заздалегідь зашитими відповідями, RAG бот може динамічно черпати інформацію з документальних джерел. Наприклад, робота технічної підтримки може при запиті користувача шукати відповідь в основі знань компанії (мануалах, інструкціях, FAQ) і видавати актуальну довідку.

Такий бот відповідатиме не загальними фразами, а конкретними витягами з посібників. Внутрішньо-корпоративні помічники також використовують RAG для доступу до внутрішньої документації: співробітники можуть у природній формі ставити питання про політиків компанії, інструкції, звіти, а робота знайде релевантний документ і сформулює відповідь. Тобто, популярний кейс – інтеграція RAG чата в

інтерфейс довідкових центрів, сайтів з документацією, де користувач може ставити питання замість ручного пошуку за статтями.

2. Пошук за документами та аналітика даних. RAG програми дозволяють реалізувати інтелектуальний пошук, що виходить за рамки простого ключового слова. Наприклад, пошук з наукової літератури: система бере питання дослідника і повертає не просто посилання, а синтезовану відповідь із цитатами зі статей. В освіті – помічники для студентів, які можуть знаходити відповіді у підручниках та лекціях. У бізнес-аналітиці – асистенти, здатні проходити за звітами, логами, фінансами та збирати коротке резюме або конкретний інсайт. У цих сценаріях генеративна модель з доступом до масиву текстів постає як аналітик, який читає сотні документів за секунди. Наприклад, у фінансовій сфері RAG може використовуватися для аналізу квартальних звітів: поставивши питання про динаміку показників, отримати відповідь, сформовану на основі актуальних даних з кількох документів. У науковій сфері за допомогою RAG створюють помічників для роботи з величезними колекціями публікацій – система на льоту перевіряє гіпотези, шукає згадки необхідних фактів та оформляє їх складно. Нещодавні дослідження, такі як HiPerRAG, показали ефективність RAG для наукових питань, які потребують фактичної точності.

3. Служби підтримки та клієнтські інтерфейси. RAG знайшов застосування у автоматизації підтримки клієнтів. Замість скриптів або довгих списків FAQ, компанії впроваджують RAG ботів у контакт центрах, які здатні відповісти практично на будь-яке питання, якщо відповідь десь записана. Вони шукають по базі тикетів, по мануалах, за даними knowledge base і дають відповідь користувачу відразу в чат. Якщо інтегрувати такий бот в систему допомоги (наприклад, при натисканні «Допомога» на сайті), користувач отримує інтерактивний пошук: ставить запитання своїми словами і отримує конкретну відповідь. Важлива перевага – відповідь містить актуальну інформацію: якщо база даних оновилася вчора, бот вже сьогодні відповідатиме з урахуванням нових даних (на відміну від моделей, які

потрібно перенавчувати). Багато фірм впроваджують RAG для внутрішньої підтримки співробітників (відповіді на питання HR, IT-підтримка) або зовнішньої підтримки клієнтів (за продуктами, послугами). Згідно з звітами, це знижує навантаження на call-центри і прискорює час реакції на запити.

4. Спеціалізовані домени (медицина, юриспруденція): У професійних галузях, де потрібний і великий обсяг знань, і точність, RAG виявилася особливо цінним. Медичні асистенти можуть отримати доступ до бази медичних знань (дослідження, керівництва, історії хвороби) та на запитання лікарів чи пацієнтів давати відповіді, посилаючись на клінічні дані. LLM самі по собі схильні галюцинувати, що небезпечно в медицині, але з RAG перевірка фактів йде за довіреними джерелами (наприклад останні протоколи лікування). Юридичні помічники здатні шукати за основою законів, судових рішень та контрактів, видаючи вичавлення релевантної практики у заданій справі. Замість того, щоб юристу вручну перелопачувати тони документів, модель RAG знайде потрібні прецеденти і навіть чорнову відповідь сформує на їх основі.

Інші області – e-commerce (боти, які уточнюють запит та шукають товари в каталозі, описуючи їх, ґрунтуючись на картках товарів), переклад та локалізація (генерація перекладу з урахуванням термінології з бази даних, чого складно добитися простою генерацією), бізнес-асистенти (резюмування внутрішніх звітів, порівняльний аналіз документів). У всіх цих випадках RAG дає динамічність та контекстність. Тобто, модель підлаштовується під поточні дані, а не лише те, чого її навчили.

Багатомовні та мультимодальні програми. Цікавий напрямок – це комбінація RAG з іншими типами даних. Наприклад, чат бот, який не лише текст шукає, а й переглядає зображення чи бази коду. Вже існують реалізації RAG, де retriever працює за графом знань чи базою коду (наприклад, GraphRAG для пошуку за графовими даними, CodeRAG – за вихідним кодом). Це розширює спектр завдань – від аналізу зображення (описуючи картинку, робот може звертатися до базі знань у тому, що у ній) до допомоги

розробникам (робот читає документацію API і частини коду, відповідаючи питання). Однак, текстові дані залишаються головним полем RAG на даний момент. В цілому, наведені кейси показують, що RAG є універсальним підходом для ситуацій, коли потрібно поєднати міць генеративної моделі та актуальність зовнішньої інформації.

RAG системи вже допомагають і вченим, і бізнес-аналітикам, і звичайним користувачам отримувати відповіді на складні питання швидше і точніше. RAG часто протиставляють традиційному fine-tuning LLM. Тому доцільно їх порівняти між собою.

1.4 Порівняльній аналіз RAG і fine-tuning

Обидва підходи – RAG і fine tuning – мають адаптувати моделі під потреби користувача, але роблять це принципово різними способами. Розглянемо їх ключові відмінності та сценарії застосування за кількома пріоритетними аспектами.

1. Спосіб збагачення знань. RAG підключає зовнішні дані під час запиту, модель отримує актуальні відомості з бази знань, не змінюючи своїх параметрів, а знання зберігаються окремо (як документи). Fine-tuning вбудовує знання у самі параметри моделі, тобто модель оновлює ваги на новому датасеті, запам'ятовуючи інформацію та шаблони з нього, та після навчання знання статично знаходяться в моделі.

2. Актуальність інформації. При RAG вона висока, можна одразу оновлювати базу документів. Відповіді ґрунтуються на свіжих даних (наприклад, останні новини, нові документи). Ризик старіння мінімальний – достатньо додати нові документи. У fine tuning вона обмежена навчальним датасетом: модель відповідає за «знімком» знань на момент навчання. Якщо дані змінилися, потрібен новий fine-tuning. Без перенавчання модель даватиме застарілі відповіді.

3. Точність та деталізація. RAG-відповіді збагачені контекстом, наводять факти та деталі із джерел. При цьому мінімізується галюцинація (модель, зазвичай, не виходить за межі наданих фактів). Однак, якість залежить від релевантності знайдених документів. Можлива «розмова мовою документів», іноді відповіді виходять громіздкими. Fine-tuning глибоко інтегрує доменні знання, завдяки чому може давати дуже точні за термінологією та стилем відповіді у своїй галузі. Наприклад, модель, навчена на юридичних текстах, впевнено оперуватиме юридичними поняттями. Вона може бути лаконічнішою через те, що генерує відповідь із «засвоєних» знань, але якщо питання виходить за рамки вивчених даних – можливі вигадки. Галюцинації знижуються за рахунок фокусування на домені, але не повністю усуваються.

4. Прозорість і зрозумілість. У RAG вони мають високий рівень – кожну відповідь можна порівняти з конкретними джерелами (документами). Легко довести, звідки взялася інформація. Це важливо у критичних додатках (медицина, фінанси) та для довіри користувачів. Для fine-tuning – низькі. Модель як «чорна скринька», і важко зрозуміти, на основі якого фрагмента даних вона видала той чи інший факт. Пояснити відповідь можна лише загальними словами («модель настільки навчена»). Для відповідальних застосувань – це мінус.

5. Складність реалізації. Для RAG потрібно більше інженерних навичок: треба налаштувати pipeline завантаження даних, підняти векторну БД, забезпечити продуктивність пошуку. Однак ML-експертизи потрібно менше – не потрібно налаштовувати навчання нейромережі, достатньо вміти користуватись готовими API та базами. Власне, складність – програмна інтеграція компонентів. Fine-tuning вимагає глибоких знань у ML, потрібно підготувати датасет, вибрати гіперпараметри, можна написати скрипти навчання, потім ретельно валідувати якість моделі.

Fine-tuning великих LLM – нетривіальне завдання, що потребує досвіду у NLP та доступу до потужного заліза (GPU/TPU). Проте після навчання

використання моделі тривіальне (просто виклик моделі, без зовнішніх залежностей).

6. Швидкість виводу (inference). RAG додає крок пошуку, що може збільшити час відповіді, особливо якщо база знань дуже велика. У середньому відповідь RAG дорівнює час пошуку (десятки мілісекунд) + час генерації LLM (сотні мілісекунд для GPT-3.5, більше для GPT-4). Для невеликих баз (<100k документів) затримка пошуку є незначною, але на мільйонах документів може вимагати ~0,5 с, що відчутно. Масштабування бази іноді упирається у зростання затримок чи пам'яті (індекси займають ресурси). Fine-tuning модель відповідає за один крок без додаткових зовнішніх операцій. Тому, при разовому запиті може бути швидшим. Однак, якщо fine-tuning модель дуже велика (багато хто використовує 13B+ параметрів), сама по собі генерація повільна і вимоглива до ресурсів. Це можна компенсувати скороченням моделі (quantization) або вибором меншої архітектури для навчання. В цілому, fine-tuning підхід більш передбачуваний за часом відповіді – він залежить тільки від моделі та не зростає з обсягом даних, як у RAG (де зростання бази може сповільнювати пошук).

7. Вартість та підтримка. RAG дешевше за розробкою та особливо з підтримки знань. Основні витрати – це зберігання та пошук даних (інфраструктура під векторну БД) та виклики LLM по API. У розробці RAG потрібні інженери-програмісти (для pipeline), але не обов'язково тримати дослідників ML. Fine-tuning вимагає великих початкових інвестицій: підготовка датасету (розмітка, чищення), прогін навчання на GPU (що може коштувати тисячі доларів для GPT-сумісної моделі), експерименти щодо підбору параметрів. Після отримання моделі – її теж треба десь хостити (якщо це 20-мільярдна модель, потрібні потужні сервери або дорогий inference API).

Оновлення знань – знову новий цикл навчання (з відповідними витратами). Тому fine-tuning виправданий, коли вигоди покривають ці зусилля (наприклад, дуже критична точність чи великий тираж застосування

моделі). Для рідкісних або швидко змінюваних запитів fine-tuning економічно програє RAG.

З врахуванням цього потрібно визначити коли вибирати RAG, а коли fine-tuning. Якщо область знань часто оновлюється або велика, і вам потрібні найактуальніші відповіді, RAG – очевидний вибір. Він же кращий, коли є багато незв'язаних фактів (наприклад, база документів) і користувач може запитати про будь-який з них – немає сенсу вливати їх все в модель, краще зберігати зовні. RAG незамінний, коли важлива доказовість відповідей: наприклад, в аналітичних звітах, рекомендаціях, де потрібно бачити джерело даних – fine-tuning модель не пояснить, на чому ґрунтується відповідь, а RAG дасть посилання. Також RAG швидше прототипується: за день можна зібрати працюючий прототип чат бота на своїх даних, тоді як fine-tuning за день точно не зробиш. З іншого боку, донавчання моделі дає плюси в наступних ситуаціях.

1. Потрібна специфічна поведінка або стиль виведення. Fine-tuning відмінно підходить для налаштування тону відповіді, формату. Наприклад, навчити модель відповідати строго у форматі JSON, або говорити юридичною мовою – це простіше через навчання.

2. Дані відносно статичні та добре структуровані. Якщо є корпус із, скажімо, 1000 документів вузької тематики, який рідко змінюється, fine-tuning LLM на ньому може повністю усунути потребу в зовнішньому пошуку – модель і так знатиме відповіді. Особливо якщо питання завжди приблизно в одному стилі (наприклад, класифікація, добування сутностей) – fine-tuning дає чудові результати в таких завданнях.

3. Комбінація методів. Як зазначалося, іноді найкращим виявляється гібрид. Наприклад, донавчили LLM на своєму стилі спілкування та базових знаннях (щоб не пояснювати щоразу терміни), а поверх цього все одно використовуєте RAG для фактів. Такий бот буде говорити вашим тоном, але цифри та факти підтягуватиме з бази. Проте суміщення ускладнює систему та вимагає враховувати ризики обох сторін.

4. Обмеження на інфраструктуру. RAG-системі для роботи потрібен ще й індекс і десь обертовий retriever. У деяких випадках (мобільний додаток, оффлайн оточення) простіше мати одну навчену модель і тільки її. Fine-tuning модель може бути оптимізована та запущена, наприклад, прямо на пристрої (особливо з технологіями на кшталт quantization, distillation). Це автономне рішення, тоді як RAG в off-line режимі важко уявити без встановленої бази даних.

У загальному випадку, RAG і fine-tuning – не конкуренти, а інструменти, що взаємодоповнюють. RAG швидкий та дешевий у підключенні знань, fine-tuning покращує саму модель під ваші потреби. Зараз в індустрії явно простежується тренд на користь RAG для широкого кола додатків, тому що він дає гнучкість та масштабованість знань без необхідності щоразу тренувати модель заново. Fine-tuning же зарезервований для випадків, де без адаптації моделі не обійтися (наприклад, треба навчити модель нову задачу, якої вона не вміє, або домогтися extra якості там, де RAG не справляється).

Висновки до розділу 1

Сучасні LLM демонструють високу ефективність, але їхні знання залишаються статичними, що призводить до ризику галюцинацій та швидкої втрати актуальності. Класичний fine-tuning дозволяє адаптувати модель до специфічних задач, проте потребує значних ресурсів і може «перезаписувати» попередні знання. Легші підходи, такі як LoRA або QLoRA, частково зменшують ці проблеми, однак не роблять процес донавчання простим чи оперативним. Тому на практиці дедалі частіше використовують комбінацію промпт-інжинірингу, зовнішніх баз знань та RAG-архітектури. RAG поєднує генеративні можливості LLM із зовнішніми джерелами даних, долаючи обмеження статичного навчання. Система

відокремлює етап пошуку інформації від етапу генерації, що забезпечує точніші та актуальні відповіді. Використання ембеддингів та векторних баз дозволяє знаходити релевантні фрагменти, які інтегруються у промпт і збагачують знання моделі в режимі реального часу. Це суттєво знижує кількість галюцинацій і відкриває можливість масштабного застосування LLM у сферах, де критично важлива фактологічність.

RAG знаходить застосування у чат-ботах, довідкових системах, юридичному та медичному аналізі, роботі з бізнес-звітами та науковими текстами. У всіх цих випадках технологія забезпечує оперативний доступ до актуальної інформації, скорочує навантаження на служби підтримки та прискорює пошук потрібних даних. Поява мультимодальних і багатомовних RAG-модулів додатково доводить потенціал цього підходу в майбутніх інтелектуальних системах.

Порівняння RAG і fine-tuning показує, що обидва методи вирішують задачу адаптації моделі, але різними шляхами. RAG забезпечує актуальність і масштабованість завдяки зовнішньому пошуку, тоді як fine-tuning вбудовує доменні знання безпосередньо в модель, формуючи сталий стиль і поведінку. Fine-tuning є дорожчим і повільнішим, але точним у вузьких галузях; RAG – дешевшим, швидшим і більш гнучким, однак залежним від якості ретривера. На практиці найкращий результат забезпечує комбінування цих підходів.

РОЗДІЛ 2
ФОРМУВАННЯ ТЕХНІК ПОШУКОВО-ПІДКРІПЛЕНОЇ ГЕНЕРАЦІЇ ДЛЯ ІНТЕГРАЦІЇ З БАЗОЮ ЗНАНЬ КОМПАНІЇ

2.1 Особливості впровадження пошуково-підкріпленої генерації

Залежно від того, яке запитання поставив користувач, система RAG повинна знайти відповідну статтю в базі знань, і подати на вхід LLM не тільки питання користувача, але й релевантну запиту частину вмісту бази знань, щоб LLM могла сформувати правильну відповідь. Таким чином, фраза Retrieval Augmented Generation досить точно описує суть того, що відбувається:

– Retrieval – пошук та вилучення релевантної інформації. Частина системи, яка відповідає за пошук та вилучення інформації, так і називають – ретрівер (retriever);

– Retrieval Augmented – доповнення запиту користувача, знайденої релевантною інформацією;

– Retrieval Augmented Generation – генерація відповіді користувачу з урахуванням додатково знайденої релевантної інформації.

Як видно з опису нічого складного: розбили текст на шматки, кожен фрагмент оцифрували, знайшли найбільш близькі за змістом шматки і подали текст цих шматків на вхід великої мовної моделі разом із запитом від користувача. Більш того, весь цей процес як правило вже реалізований у так званому pipeline і все, що вам потрібно, це власне запустити pipeline (з якої готової бібліотеки). Хоча концепція RAG має досить прямолінійний вигляд, на практиці розробка таких систем пов'язана з низкою технічних викликів.

1. Розмір статей з бази знань – якого розміру фрагменти тексту треба давати LLM, щоб вона формувала відповідь?

2. Нечіткий пошук – просто взяти запит користувача та знайти за точною відповідністю усі фрагментами з бази знань не вийде. Яким

алгоритмом реалізувати пошук, щоб шукав релевантні і лише релевантні фрагменти тексту?

3. Якщо у базі знань знайшлося кілька статей? А якщо вони великі? Як їх «обрізати», як комбінувати, можливо стиснути?

Це самі базові питання, з якими стикається будь-який розробник RAG. На даний момент є загальноприйнятий підхід, з якого потрібно починати створення RAG, базується на кількох положеннях.

1. Розмір чанків та їх кількість. Зараз рекомендується розпочати експерименти зі зміни розміру чанків та їх кількості. Якщо розробник подає на вхід LLM занадто багато непотрібної інформації або навпаки, занадто мало, то він просто не залишаєте шансів моделі відповісти правильно. Тобто, Чим менший чанк за розміром, тим точнішим буде буквальний пошук, чим більший розмір чанку тим більше пошук наближається до смислового. Різні запити користувача можуть містити різну кількість чанків, які потрібно додавати в контекст. Необхідно досвідченим шляхом підібрати той самий коефіцієнт, нижче якого чанк сенсу не має і лише засмічуватиме ваш контекст. Чанки повинні перекривати один одного, щоб у вас був шанс подати на вхід послідовність чанків, які йдуть один за одним разом, а не просто шматки, що вирвані з контексту. Початок і кінець чанка повинні бути осмисленими, в ідеалі повинні збігатися з початком і кінцем речення, а краще абзацу, щоб уся думка була в чанці цілком.

2. Додавання інших методів пошуку. Дуже часто пошук «за змістом» через ембеддінг не дає потрібного результату. Особливо якщо йдеться про якісь специфічні терміни чи визначення. Як правило, до пошуку через ембеддинги підключають також TF IDF пошук і об'єднують результати пошуку в пропорції, підібраної експериментальним шляхом. Також дуже часто допомагає ранжування знайдених результатів, наприклад, алгоритмом BM25.

3. Мультиплікація запиту. Як правило, запит користувача має сенс кілька разів перефразувати (за допомогою LLM) і здійснювати пошук чанків

по всіх варіантах запиту. На практиці, роблять від 3-ох до 5-ти варіацій запитів і потім результати пошуку об'єднують в один.

4. Сумаризація чанків. Якщо за запитом користувача знайдено дуже багато інформації і вся ця інформація не поміщається в контекст, то її можна також «спростити» за допомогою LLM і подати на вхід у вигляді контексту (на додаток до питання користувача) щось на кшталт заархівованого знання, щоб LLM могла використовувати суть (вижимання з бази знань) для формування відповіді.

5. Системний промпт та донавчання моделі на формат RAG. Щоб модель краще розуміла, що від неї потрібно також донавчити LLM на правильний формат взаємодії з нею. У підході RAG контекст завжди складається з двох частин (ми поки не розглядаємо варіант діалогу у форматі RAG): питання користувача і знайденого контексту. Відповідно модель можна донавчити розуміти саме такий формат: тут питання, тут інформація для відповіді, видай відповідь, до питання. На початковому етапі цю проблему можна спробувати вирішити через системний промпт, пояснивши моделі, що «питання тут, інформація тут, не переплутай!».

Варто відзначити важливу річ, яку необхідно реалізувати при роботі з RAG. Це питання оцінки якості її роботи. Це актуально при зміні бібліотеки ембеддера або LLM. Потрібно запустити щонайменше кілька десятків тестів та оцінювати якість результату. Як правило, для оцінки якості RAG моделі використовують наступні підходи.

1. Питання для перевірки – мають бути написані людьми. Тут нікуди не можна подітися, тільки розробник (або замовник) знає які питання будуть задаватися системі і ніхто, крім вас, перевірочні питання не напише.

2. Референсні відповіді – теж мають бути написані людьми, і бажано різними і у двох (а то й 3-ох) примірниках, щоб позбавитися залежності від людського чинника. Але тутє варіанти. Наприклад, асистенти від OpenAI цілком собі непогано справляються з цим завданням, особливо якщо в них не вантажити великі (більше 100 сторінок) документи. Для отримання

референсних відповідей, можна написати скрипт, який може звернутися до асистента (до якого завантажені релевантні документи) і поставить йому всі тестові запитання для перевірки та/або кілька разів одне й те саме питання.

3. Близькість відповідей LLM до референсних вимірюється кількома метриками, такими як BERTScore, BLEURT, METEOR та навіть простим ROUGE. Як правило, експериментальним шляхом підбирається середньозважена метрика, яка є сумою перерахованих вище (з відповідними коефіцієнтами) яка найбільш точно відображає реальну близькість відповідей вашої LLM до золотих відповідей.

4. Референсні чанки та близькість знайдених чанків до референсних. Як правило, розробник RAG у своєму бажанні швидше отримати хороші відповіді від генератора забуває про те, що знайдені чанки, які подаються на вхід цьому генератору забезпечують 80 % якості відповіді. Тому, при розробці RAG першочергову увагу необхідно приділити тим самим даним, які знайшов ретрівер. В першу чергу, необхідно створити БД з питань та відповідних їм чанків і щоразу модифікувати RAG та перевіряти наскільки точно ретривер знайшов чанки, що відповідають запиту від користувача.

2.2 Операції з векторними БД

Згідно п.1.2, в системах RAG застосовується таке поняття embedding. Це числове векторне представлення даних (тексту, зображень, аудіо, коду, відео та ін.), отриманої за допомогою спеціалізованої моделі, яка називається енкодером (або ембеддером). Така модель – фактичний «пакувальник сенсу». Вона перетворює будь-які вхідні дані в масив чисел фіксованої довжини (вектор), де семантично схожі об'єкти розташовуються близько один до одного в багатовимірному просторі.

Наприклад, фраза «клієнт просив повернути гроші» перетворюється на умовний вектор [0.23, -0.15, 0.89, ...], а фраза «замовник хотів отримати

повернення коштів» у вектор [0.21, -0.14, 0.87, ...]. Відстань між векторами вимірюється метриками, наприклад, наступними.

1. Косинусна схожість має значення від -1 (протилежний зміст) до 1 (ідентичний зміст):

$$\cos(\theta) = \frac{A \cdot B}{|A| + |B|}, \tag{2.1}$$

де A і B – вектори.

2. Відстань Евкліда (менша відстань означає велику схожість):

$$E = \sqrt{\sum \left(A_i - B_i\right)^2}\,. \tag{2.2}$$

Для вказаного прикладу, косинусна схожість між векторами дорівнює 0.95, тобто обидві фрази дуже близькі за змістом, хоча написані зовсім різними словами. Якщо ми візьмемо «кіт стрибає по столу», то буде інший вектор, маленька косинусна подібність і, як наслідок, дуже далекий сенс від перших двох. Тобто, умовно ембедінг – це сенс. Але всі працюємо з різними сенсами і це велика складність. Поняття сенсу суб'єктивно і залежить від предметної області. Навіть під одними і тими ж словами можна розуміти різне: реверс (двигуна, монети чи інжиніринг?), артефакт (спотворення чи об'єкт?) або ж відноситься до різних індустрій (наприклад, слово «протокол» – воно про IT, медицину чи криміналістику?). Тому ембединги, а точніше, моделі, які їх генерують, можна умовно розділити на дві великі групи: загального призначення (навчені на всьому корпусі мови) або спеціалізовані. Звичайно, спеціалізовані працюватимуть краще: якщо робимо медичний RAG, то «аспірин» має швидше асоціюватися з «циклооксигеназа-1», ніж із просто головним болем. Але спеціалізовані моделі – недешеві у всіх сенсах застосування. У ембедингів є розмірність. Якщо спрощено, то чим вона більша, тим більше інформації в них є, але й тим більше пам'яті для цього потрібно. Для зберігання цих самих ембедингів призначені бази даних. Але не лише зберігання, а ще й різних операцій. Пошук працює не за буквальним збігом слів, а за змістом. Емведдінги створюються один раз і зберігаються,

тому потрібний фрагмент знаходиться швидко і без зайвих обчислень. Це робить векторні БД зручним інструментом для роботи з LLM і великими масивами даних. Таким чином, в таких БД оперуємо цими векторами тому що над ними зручно робити математичні операції, але зрештою нам потрібні вихідні дані. Наприклад, по векторах вже знайшли 10 схожих відео на наше, але не можемо віддати ці вектори клієнту, нам потрібні саме ці відео. Тому, крім самого зберігання ембедингу, потрібно зберігати мета-інформацію – ким завантажено, категорію, назву, сам файл тощо. В даному випадку, є два варіанти. Першій – зберігання у самій векторній БД (вбудоване). З плюсів – одна система, одна точка відмови, атомарність операцій. Мінуси – векторні БД насамперед оптимізовані під вектори і зберігати велику кількість мета-інформації буде дорожче. Таке доречно у невеликих RAG-системах (<1M документів), прототипи або системи з короткими чанками (<1000 умовних токенів). Другий – зовнішнє сховище + reference. Векторна БД зберігає лише ембединги, ID та опціонально – короткі метадані, а всі оригінали – у звичній базі. Із плюсів тут те, що кожен інструмент робить свою роботу, з мінусів – потрібна синхронізація. Є ще один частковий спосіб у вигляді гібридного рішення: вектор у PostgreSQL, blob у S3.

Суть RAG – це пошук, а «R» в абревіатурі в сто разів важливіший за «G». RAG так добре зайшов не тому, що це нова технологія, а потім, що LLM вміють розрізнену інформацію однаково подати. Процес: документи розбиваються на фрагменти, кожен перетворюється на ембединг; запит користувача векторизується та відшукуються схожі фрагменти; знайдені фрагменти передаються LLM як контекст. Таким чином, завдання векторної БД RAG: семантичний пошук (основне завдання), фільтрація метаданих (наприклад, дата, автор, тип документа), гібридний пошук (векторний + BM25), переранжування результатів (reranking) [19], робота з великим обсягом документів, інкрементальна індексація.

Тепер визначимо операції над ембедінгами. Основна операція – векторний пошук (ANN search): «Знайди K найбільш схожих векторів на

запит». Це не точний збіг як у SQL (WHERE id = 5), а пошук по близькості – топ-10, топ-100 найсемантичніших об'єктів. Результат повертається в міру близькості/відстань, відсортований за релевантністю.

CRUD-операції: Insert (додати нові вектори), Update (оновити метадані, сам вектор зазвичай не змінюється), Delete (видалити), Upsert (оновити чи створити, якщо немає). При вставці вектор автоматично додається до індексу для швидкого пошуку. Про індекс ми ще поговоримо.

Гібридний пошук: наприклад, комбінація векторного пошуку + повнотекстого (BM25) + фільтрація метаданих. Вектор знаходить за змістом, BM25 посилює точні ключові слова, фільтри відсікають метаданим.

Фільтрування метаданих: векторний пошук + умови на метадані в одному запиті. Наприклад: «знайди схожі документи, але тільки category='tech' AND date >= 2025». Критично важливо, щоб фільтри застосовувалися під час пошуку (pre-filtering), а не після (post-filtering), інакше втрачається точність.

Batch-операції: масове завантаження тисяч/мільйонів векторів за один раз (bulk insert), масове видалення, переіндексація. Критично для початкового завантаження великих датасетів чи періодичної синхронізації.

Реіндексація: перестворення індексу з іншими параметрами (тип індексу, метрики подібності/відстань, квантизація) без перезавантаження оригінальних векторів. Дозволяє експериментувати з налаштуваннями продуктивності.

Всі ці операції мають працювати швидко навіть на мільйонах векторів – це головна відмінність спеціалізованих векторних БД від просто сховища масивів чисел. У всіх векторних БД є спільні властивості. Базова архітектура векторних БД схожа на реляційні, але з векторною специфікою: колекції (collections/tables) – аналог таблиць SQL, кожна колекція зберігає вектори однієї розмірності; документи/точки (points/documents) – аналог рядків. Кожен документ містить: вектор (масив чисел) + метадані (JSON-подібна структура з категоріями, датами, тегами тощо); індекси – обов'язкова

структура швидкого пошуку, про них ми поговоримо; метрики близькості замість JOIN'ів (векторні БД використовують математичні операції для пошуку: заходи подібності – Cosine Similarity та Dot Product, та заходи відстані – L2/Euclidean), а метрика фіксується під час створення колекції – зміна вимагає переіндексації; векторний пошук замість SELECT. Базова операція: «Знайди K найближчих векторів до запиту» (замість «вибери рядки WHERE умова»), а результат – відсортований список за similarity score.

На відміну від реляційних БД, у векторних замість точного пошуку (WHERE id = 5) – відбувається приблизний пошук (топ-10 схожих). Тобто, замість нормалізації (зв'язок між таблицями) – є денормалізація (все у одному документі). Багато векторних БД орієнтовані на масштабованість, і поведінка узгодженості може бути менш суворою, ніж у класичних реляційних БД.

Надалі доцільно визначити індекси у векторних БД. Слово «індекс» – для векторів може мати різний сенс. Від вибору індексу залежить швидкість пошуку, споживання пам'яті та точність результатів:

– HNSW (Hierarchical Navigable Small World) – багатошаровий граф навігації, основний стандарт для швидкості та точності;

– IVF (Inverted File Index) – сімейство індексів, що розбиває простір на кластери, швидка індексація та робота з диском;

– DiskANN – індекс для SSD (NVMe), що дозволяє працювати з терабайтами векторів при мінімальній RAM;

– Flat (Brute Force) – точний пошук без індексу, порівняння з усіма векторами, використовується для малих датасетів (<100K);

– Product Quantization (PQ) – індекс, що стискає, економить пам'ять у 8-32 рази за рахунок невеликої втрати точності.

Кожна векторна БД не просто підтримує свій набір індексів, а оптимізує їх під своє сховище та движок. Наприклад, Qdrant модифікував HNSW для кращої роботи з фільтрами, а Timescale створив свій StreamingDiskANN спеціально під PostgreSQL. Ці оптимізації дають перевагу 20-40 % за продуктивністю проти універсальних реалізацій. Квантизація

індексів – ще один метод оптимізації. Вектори зберігаються в float32 (4 байти на число), а квантизація стискає їх до int8 або навіть бітів. БД із вбудованою квантизацією (Qdrant, Milvus, pgvectorscale) дозволяють експериментувати: завантажили вектори один раз (вони зберігаються як є) → перетворили індекс з різною квантизацією → порівняли пам'ять/швидкість/точність. Оптимізовані під движок версії можуть працювати набагато краще самописних. Наприклад, Binary Quantization від Qdrant з oversampling за їх заявами дає recall 95 % + проти 70-80 % у наївної реалізації.

2.3 Визначення основних технік пошуково-підкріпленої генерації

При проектуванні RAG-системи інженер щоразу стикається з безліччю питань: як отримувати чанки, яку векторну базу використовувати, як організувати отримання релевантної інформації з бази, та навіть вибір ембеддера може зайняти пристойний час – і це лише вершина айсберга. Ідеальним рішенням є перебір основних варіантів, потім оцінка якості та вибір придатних для конкретного завдання. Адже те, що добре працює, наприклад для технічної підтримки, може провалитися в юридичному аналізі, і навпаки. Згідно п. 1.2, спочатку розглянемо просту схема роботи RAG (рис. 2.1) [20]. Принцип роботи досить простий. Запит користувача векторизується (1), потім вектор порівнюється з векторами у базі даних з допомогою косинусної близькості (2). Топ семантично близьких векторів дістаються з бази даних і подаються разом із запитом користувача подаються в контекст LLM, яка і видає підсумкову відповідь (3). До переваг відноситься те, що добре працює з коробки на більшості завдань; простота реалізації; працює як швидкий старт (MVP). Серед недоліків слід вказати: фіксована кількість чанків у контексті; сильна залежність від ембеддера. Сфери застосування – прості системи, де готові відповіді зазвичай є у документації. При цьому, спочатку мають йти найбільш релевантні чанки. LLM добре

використовують інформацію з початку та кінця промпту, а інформація в середині обробляється гірше [21].

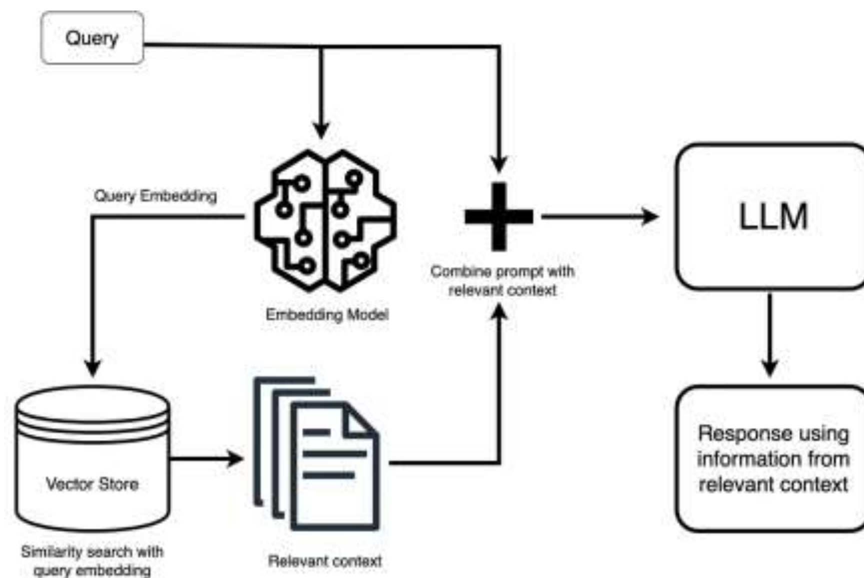

Рисунок 2.1 – Проста схема роботи RAG

І якщо врахувати, що промпт RAG-система часто виглядає так: «System промпт + інструкції + питання + контекст», то перший і найбільш релевантний (як ми припускаємо) чанк виявляється десь у середині. І така проблема справді є. На даний час, існують дослідження пристрасті LLM до певних позицій релевантної інформації у промпті. Наприклад, дослідникам вдалося показати, що Qwen2.5-7B-Instruct точніше відповідає на питання, коли релевантні чанки знаходяться в кінці. Llama3-8B-Instruct навпаки працює краще, коли релевантні чанки знаходяться зверху. Причому дана особливість помічається як під час роботи з текстами російською, так і англійською, німецькою, хінді і навіть в'єтнамською.

Особливу увагу потрібно приділити вибору векторизатора, оскільки він відповідає за отримання релевантної інформації. Найчастіше FRIDA чи BGE-M3 [22] справляється з цим завданням високому рівні. Однак все одно першим кроком налаштування retrieval-модуля має бути перевірка якості різних векторизаторів. Також потрібно налаштувати оптимальну довжину чанків, яка також може колосально вплинути на якість retrieval-модуля.

Наступна техніка – сегментація речень на смислові блоки (Propositions Chunking). Принцип роботи наступний. LLM отримує чанк та розбиває його на пропозиції, що містять точні факти. LLM оцінює отримані пропозиції. Вони, як мінімум, повинні нести корисну інформацію, бути точними і ясними. Отримані пропозиції завантажуються у векторну базу даних. Далі простий RAG. Цей метод спрямовано прості точні питання. Завдяки другому етапу відсівається потенційний шум, що зберігається у тексті. А завдяки першому етапу ми отримуємо семантично насичені чанки замість простих шматків тексту, які можуть «розмивати» семантику. Тому факти в такому форматі ближчі до атомарних одиниць сенсу – це допомагає retrieval-модулю бути точнішими. В якості переваг виділяємо: кожен запис у базі несе концентроване смислове навантаження, що полегшує роботу retrieval-модуля; зменшення навантаження на контекст LLM. Через видалення води LLM витрачає меншу кількість токенів. Це також позначається на швидкості. Слабкі сторони можна вказати такі: збільшення обчислювальної вартості підготовки бази даних (кожен чанк потрібно прогнати через LLM щоб отримати розмітки); залежність від якості роботи LLM на етапі розбиття та фільтрації (помилки спричиняють втрату фактів); потенційна фрагментація знань (довгі логічні зв'язки між пропозиціями можуть загубитися). Дана техніка може застосовуватись у системи з простими питаннями, які потребують точного факту, а також, довідники та словники, у яких кожен факт самодостатній.

Query Transformation (рис. 2.2) [23] – у даній техніці намагаємося боротися з неточними запитами до системи шляхом їх переформулювання. Насамперед, запит потрапляє в LLM, яка повинна зробити його більш конкретним та повним. Далі цей запит потрапляє до стандартної системи RAG. Наприклад: «Машина Шумахера?» → «Яку машину використовував Міхаель Шумахер під час перегонів у Формулі-1?». До переваг відносимо наступне: запити стають більш релевантними до базових документів; знижує ймовірність промаху через надто коротке або неконкретне питання. А до

недоліків: помилка на етапі переформулювання призводить до неправильного контексту; іноді система починає шукати те, чого користувач не мав на увазі.

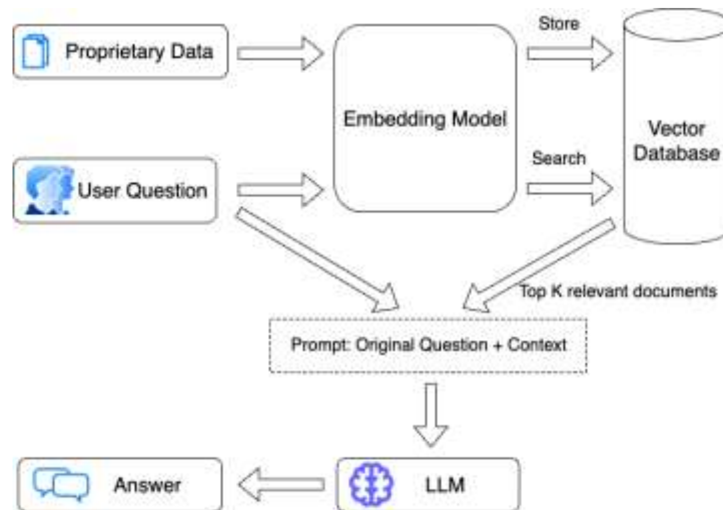

Рисунок 2.2 – Query Transformation

Ця техніка використовується для чат-ботів для широкої аудиторії (користувачі часто пишуть криві запити), а також для пошук по великих шумних базах знань, де важливо уточнювати контекст.

Query Decomposition (рис. 2.3) [24]. Особливу складність RAG-система може відчувати, коли надходить складове питання.

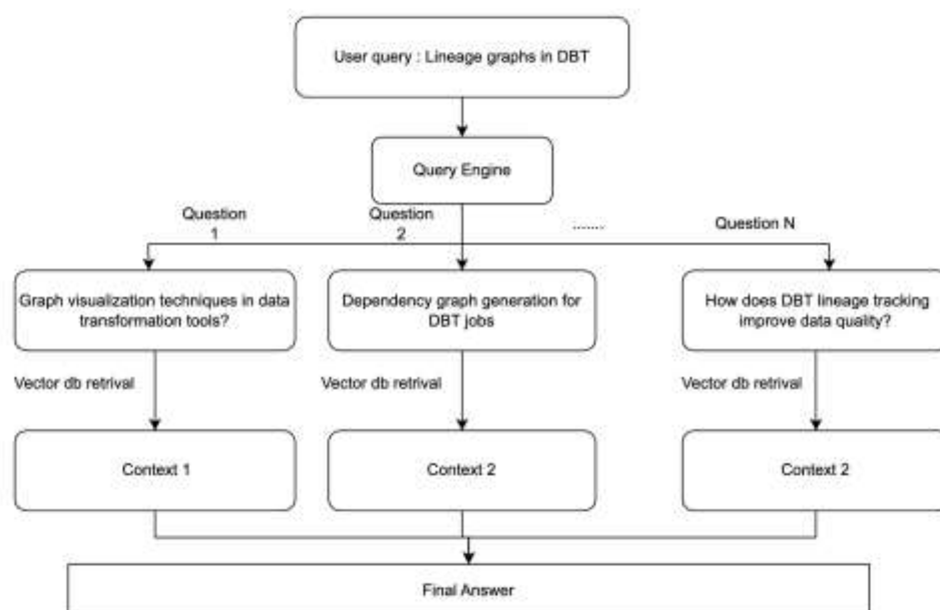

Рисунок 2.3 – Техніка Query Decomposition

Наприклад: «Як мені потрапити в офіс і о котрій я повинен там бути?» Логічним рішенням є обробка питання LLM моделлю, яка розіб'є складове питання на прості питання («Як потрапити в офіс?» і «У скільки потрібно бути в офісі?»). Далі на кожне запитання знаходиться відповідь за допомогою звичайної RAG, а наприкінці всі відповіді разом із запитанням передаються LLM, яка їх агрегує та видає фінальну відповідь.

Відповідно, плюсами є те, що дозволяє працювати зі складними, складовими запитами; дроблення на підзавдання знижує навантаження для retrieval-модуля та LLM. Недоліками є: помилки у розбитті призводять до помилок у відповіді; збільшення обчислювальних витрат – для кожного підзапиту робиться retrieval-запит та генерація. Може застосовуватись у аналітичних системах, в яких запит торкається кількох тем (багатоаспектний аналіз), а також у системах із комплексними документами (юридичні, медичні та ін.).

У більш серйозних системах Query Transformation і Query Decomposition об'єднуються, що робить RAG більш стійким до живих питань користувача, далеким від ідеальних запитів. У випадках, коли баз з документами декілька, можна закинути в контекст LLM, що відповідає за розширення/декомпозування запитів, назви цих баз та їх опис (це вже крок до routing RAG). В такому випадку, LLM розумітиме предметну область і зможе ефективніше обробити запит.

Hypothetical Document Embedding (HyDE) – ще один спосіб розширення запиту (рис. 2.4) [25].

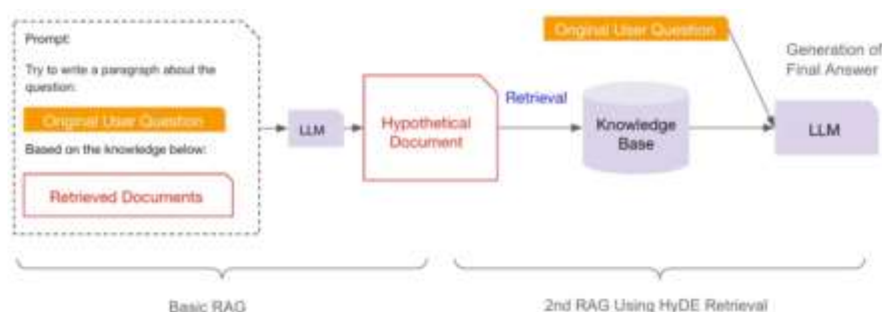

Рисунок 2.4 – Hypothetical Document Embedding

Однак у даній техніці LLM на підставі запиту користувача генерує потенційний документ, який дає відповідь на запит. Цей документ служить мостом між запитом та реальними документами, знижуючи семантичний розрив. Це через те, що з погляду векторизатора питання «Хто вперше отримав Нобелівську премію з фізики?» та відповідь «Вільгельм Конрад Рентген вперше отримав нобелівську премію з фізики» хоч і близькі, але все одно мають семантичну різницю. В цілому, така техніка покращує релевантність, особливо під час розриву між формулюванням питання та текстами документів. В той же час, імовірні помилки у згенерованому документі породжують сміття у retrieval, а також неефективно для складних багатопредметних питань, де один гіпотетичний документ не покриває всю інформацію. До варіантів застосування відносяться системи з точними питаннями та короткими фактологічними відповідями; ситуації з різними формулюваннями тексту у джерелах (наприклад, статті, новини, бази знань).

Варто зазначити, що деякі векторизатори (наприклад, BGE-M3) вміють працювати з подібними ситуаціями, вводячи як префікси фрази: query: – означає, що надаємо текст запиту користувача на вхід, passage: –означає, що подаємо текст документа або уривка (частина статті, нотатки, FAQ тощо). А якщо заглянути у картку моделі FRIDA, то можна знайти ще більше корисних префіксів.

Hypothetical Prompt Embeddings (HyPE) – а що якщо не генерувати щоразу гіпотетичну відповідь, а в момент занесення чанків до бази даних згенерувати до них гіпотетичні питання? Саме так і працює HyPE:

1. До кожного чанка генеруємо питання.

2. Пошук у векторній базі даних відбувається лише з згенерованих питань (отримуємо семантичну близькість, без необхідності додаткових генерацій у момент звернення).

3. Як контекст підтягуємо чанк, за яким було поставлене питання.

Приклад: «Вільгельм Конрад Рентген отримав Нобелівську премію з фізики 1901 року». Генерований гіпотетичний питання: Хто став першим

лауреатом Нобелівської премії з фізики? Тепер пошук по векторній базі відбувається з цього питання, що полегшує порівняння із запитом користувача, а чанк підтягується вже після. Також ми отримуємо економію часу щодо HyDE для користувачів, адже тепер не потрібно чекати, доки згенерується гіпотетична відповідь. Однак, процес внесення текстів до бази даних істотно затягується і стає набагато дорожчим. Тому в даному дослідженні метод HyPE не тестуватиметься.

Relevant Segment Extraction (RSE). Однією з головних проблем retrieval модуля у попередніх техніках є фіксована довжина чанка. Давайте уявімо графік, у якому осі X йдуть чанки, а осі Y – їх семантична близькість якомусь конкретному запиту – рис. 2.5. На графіку видно різкі перепади. Якщо просто візьмемо топ семантично близьких чанків, то мало того, що всі вони перемішаються, так ще й загубляться важливі фрагменти інформації.

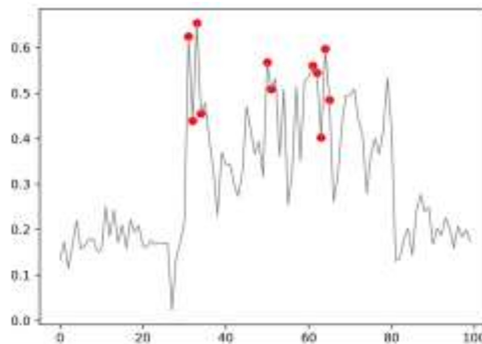

Рисунок 2.5 – Приклад роботи Relevant Segment Extraction
(червоним виділено чанки, які потраплять у контекст)

Наприклад, якщо подивитися на дві найрелевантніші чанки (це чанки під номером 31 і 33, які мають близькість до запиту ~0.6), то між ними можна помітити ще один чанк, який може нести в собі корисний для відповіді контекст. І, крім того, ці три чанки більше схожі на єдиний смисловий фрагмент, який виявився розбитим через фіксовану довжину чанків. Застосовуючи метод RSE, ми можемо поєднувати подібні чанки в єдиний смисловий фрагмент. Працює метод наступним чином.

1. Генеруються усі можливі сегменти. Це можуть бути будь-які вікна довжини не більшої за певну максимальну довжину вікна.

2. До кожного сегмента вважається його цінність. Як цінність виступає комбінована метрика, яка враховує максимальне значення релевантності, середнє значення релевантності на сегменті та бонус за довжину (щоб віддавати перевагу цільним шматкам).

Вибір функції цінності дуже впливає якість роботи. Так, якщо використовувати просту суму релевантностей (як це зроблено в репозиторії, який я згадував спочатку), то перевагу майже завжди віддаватиме довгим сегментам із середньою релевантністю (чанки під номерами 50-80), а відокремлені, але релевантні фрагменти ігноруватимуться. Якщо взяти просто середнє, то перевага навпаки віддаватиметься лише маленьким фрагментам з високими значеннями релевантностей.

3. З усіх кандидатів вибираються оптимальні сегменти так, щоб вони не перетиналися та у сумі не перевищували загальний ліміт довжини.

4. На останньому кроці сегменти можна злегка розширити і об'єднати близькі один до одного, щоб замість дрібних фрагментів вийшло менше, але більш довгих і зв'язкових сегментів.

Таким чином, RSE вирішує два ключових слабких місця простої розбивки на чанки: розрив смислових одиниць (коли один логічний фрагмент тексту виявляється порізаним); втрата релевантного контексту між чанками. Перевагами є таке: усуває проблему розбитих смислових фрагментів тексту; зменшує проблему фіксованої довжини чанків; гнучкість (можна налаштувати функцію цінності під конкретні завдання); підвищує якість retrieval-модуля без необхідності змінювати векторизатор чи модель. До недоліків відноситься те, що для формування сегментів потрібно отримати всі чанки з БД, що спричиняє додаткові обчислення при формуванні сегментів, а також залежність від функції цінності. Тобто якщо вибрати невдалу метрику, сегменти будуть занадто довгі (і засмічувати контекст), або занадто дрібні (і втрачати зв'язок). Ця техніка орієнтована на QA-системи, де

відповідь залежить від кількох близьких за змістом фрагментів документа, або аналітичні завдання (звіти, резюме довгих текстів), де важлива зв'язність контексту.

Semantic Chunking. Ще одним методом, як уникнути фіксованої довжини чанків, є розбиття чанків на семантично близькі блоки. У даній техніці чанки б'ються за допомогою SemanticChunker із бібліотеки langchain_experimental. Принцип роботи наступний [26].

1. Розбиваємо текст на речення.

2. Для кожної пропозиції отримуємо його векторне подання.

3. Вважаємо косинусну відстань між сусідніми пропозиціями. Наприклад, між першим і другим реченнями, між другим і третім, третім та четвертим тощо.

4. Складаємо розподіл відстаней. Якщо відстань потрапляє в 95-й перцентиль (евристика і її можна змінювати в залежності від бажаної довжини чанків), то вважаємо, що ці пропозиції не мають смислового зв'язку і між ними ставимо точку розриву.

5. Точки розриву показують де закінчується чанк і починається новий.

6. Чанки завантажуються у векторну базу даних.

7. Далі принцип роботи не відрізняється від простого RAG.

Сильним сторонами слід вважати: автоматичне розбиття тексту за смисловими одиницями (це дає менше розриву контексту); підходить для документів різної структури та довжини; кожен чанк несе цілісну інформацію. А от недоліками слід вважати те, що обчислювально дорожче за фіксований чанкінг (потрібно будувати векторні уявлення всіх пропозицій і вважати косинусну відстань); евристика не завжди ідеально працює (іноді розрізаються логічно пов'язані пропозиції чи поєднуються різні теми. До сфер застосування відносяться QA-системи, де тексти містять логічні блоки різної довжини. Також варто врахувати, що ця техніка добре комбінується з RSE. Semantic Chunking формує смислові чанки, а RSE поєднує кілька близьких за змістом чанків в один сегмент.

Розширення чанків – Context Enrichment Window [27]. Дана техніка будується на тому, що для кожного релевантного чанка додаються сусідні чанки (рис. 2.6). Робиться це для того, щоб зберегти логічну цілісність інформації, тому що один релевантний чанк може бути безглуздим без сусідніх речень чи абзаців. Якщо інформація для векторної бази даних береться з документів, які мають сторінки (PDF, DOCX та ін.), то гарною практикою є збереження для кожного чанка номера сторінки, з якої він взятий. Це дозволяє спочатку отримати релевантні чанки, а потім з даних чанків підтягнути цілі сторінки документів (видавши дублікати). Ця техніка підвищує повноту та якість контексту. Однак є збільшення розміру контексту та ризик додати нерелевантну інформацію, якщо сусідні чанки не мають потрібного сенсу. Сферами застосування є RAG системи, інформація для яких береться з документів із абзацами або сторінками (PDF, Word, звіти, підручники).

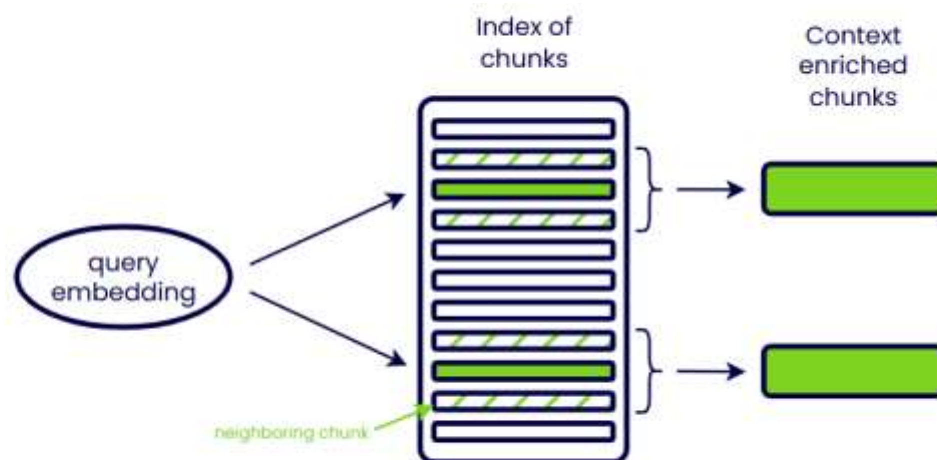

Рисунок 2.6 – Context Enrichment Window

Fusion Retrieval. При збільшенні документів моделі впираються в стелю своєї місткості (capacity) і можуть впоратися навіть із простими завданнями. Чого немає у повнотекстових моделей пошуку (наприклад, BM-25). Тому резонним є використання двох retrieval модулів: embedding-моделі для семантичного пошуку та моделі повнотекстового пошуку для підбору за

ключовими словами. Отримані з них чанки збираються та очищаються від дублікатів. Далі вони або реранжуються, або відразу подаються LLM. Отже, вдається зменшити недоліки цих двох методів. Як наслідок, комбінуються сильні сторони як семантичного пошуку, і повнотекстового, а також знижується ефект «стелі місткості» ембеддерів у великих БД.

З іншого боку, з'являється ризик дублювання, тому потрібна ретельна дедублікація та реранжування. В основному, ця техніка рекомендується для великих корпоративних баз знань із великою кількістю документів, а також QA-систем, де важливо покрити усі можливі варіанти формулювання відповіді.

Reranking (рис. 2.7) [19, 28] – потужний метод, який дозволяє оцінити релевантність отриманих чанків та (або) змінити порядок їхнього прямування в контексті.

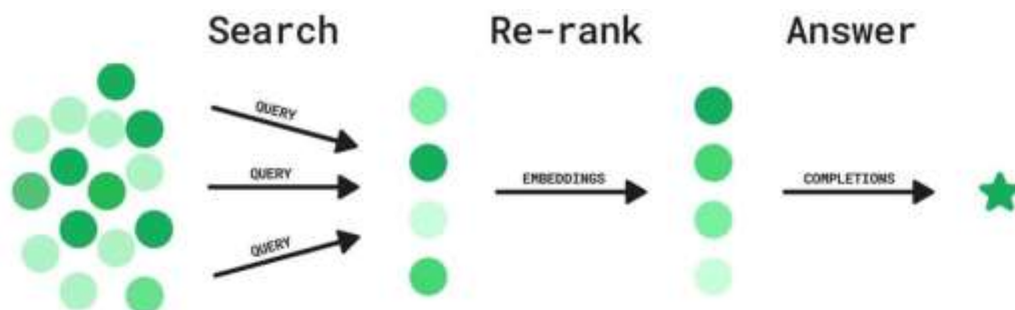

Рисунок 2.7 – Reranking

Також цей метод дозволяє позбутися обмежень, пов'язаних із фіксованою кількістю чанків. Адже, щоб дати відповідь на деякі питання, іноді вистачає 1-2 чанків, а всі інші чанки, що не належать до теми, не тільки додають шуму в контекст, а й роздмухують його. А, як ми знаємо, складність роботи механізму уваги зростає квадратично щодо розміру контексту. Тому всі нерелевантні чанки можна видалити за допомогою ранкеру. Однак у разі видалення релевантного чанка вся RAG-система втратить ефективність. І щоб цього не відбувалося, потрібно приділити багато часу його налаштуванню. Реранжування можна або за допомогою LLM, або за

допомогою крос-енкодерів. У разі роботи з LLM ми просимо оцінити конкретний чанк запиту. Причому важливо прописати конкретні критерії оцінювання та шкалу. За допомогою Structured Output отримуємо оцінки і вже за ними ранжуємо документи. У разі роботи з крос-енкодерами подаємо в модель запит та конкретний чанк, а на виході отримуємо оцінку релевантності між ними, за якою відбувається ранжування. Перевагами вважаються: видалення шуму з контексту LLM; дозволяє використовувати змінну кількість чанків; покращує якість генерації рахунок більш правильного порядку релевантних фрагментів. З іншого боку, неправильно налаштований реранкер може видаляти релевантні чанки; кожен чанк проходить через LLM або cross-encoder (це збільшує обчислювальні витрати); потребує правильної шкали оцінювання та критеріїв релевантності для LLM.

Dartboard RAG. У випадках, коли документи перетинаються, у контексті з'являється надмірність інформації. Різні чанки несуть у собі те саме. Щоб цього уникнути, запроваджується функція, яка оцінює як релевантність, так і різноманітність чанків. Оцінка функцією допомагає отримати більш всебічний відповідь, вилучивши з контексту схожі за змістом чанки. Працює наступним чином: обчислюється косинусну відстань між запитом та кожним документом (робиться автоматично під час роботи з векторними базами даних); обчислюється косинусну відстань між кожною парою документів (виходить матриця відстаней); вибирається найрелевантніший чанк; у циклі вибирається чанк, який має найкращу оцінку, яка залежить від його близькості до запиту та віддаленості від уже обраних чанків. Має переваги: зменшує дублювання інформації в контексті LLM; дозволяє отримати більш всебічну та різноманітну відповідь на запитання. В той же час, не завжди потрібний для малих баз, де дублювання мало; з'являється складність налаштування параметрів схожості та різноманітності. Може застосовуватись у RAG-системах, з великою кількістю документів, що перетинаються.

Hybrid RAG (рис. 2.8) – це розширення базового RAG-підходу, в концепції якого для пошуку знань використовуються кілька різних джерел та стратегій вилучення.

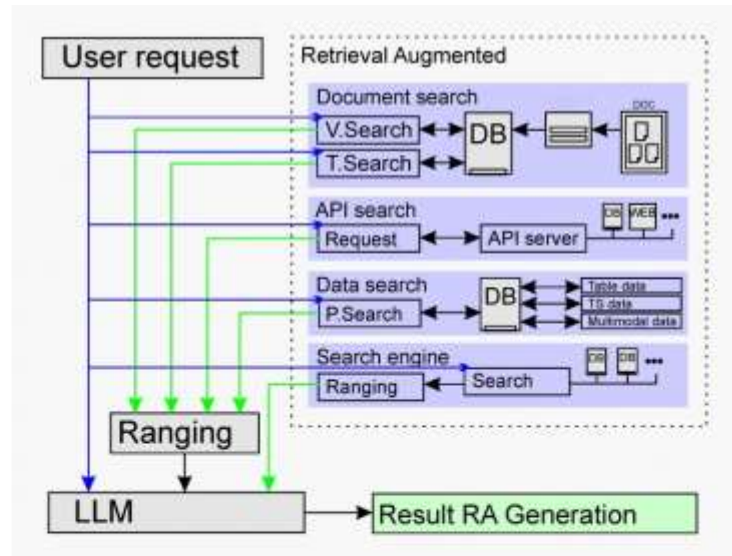

Рисунок 2.8 – Архітектура Hybrid RAG

Основна ідея – комбінувати способи пошуку та вилучення інформації для доповнення запиту. Тоді як RAG системи засновані виключно на даних, що доповнюються витягнутих з документів, Hybrid RAG пропонує підхід, комбінування і складного ранжування результатів пошуку доповнюваних даних. В основі архітектури HybridRAG лежить базова архітектура RAG доповнена альтернативними пошуковими методами, такими як Sparse retrieval, Dense Retrieval. Пошукові движки спеціалізовані API, бази даних з табличними, часовими або мультимодальними даними, графові БД.

2.4. Концепція Agentic RAG

Agentic RAG – це концепція створення розумніших AI-систем, які розуміють контекст. RAG-конвеєри виглядають переконливо в теорії, але швидко ламаються на практиці. Аналіз галузі показує: 87 % корпоративних

впроваджень RAG не дають очікуваного ефекту. Причини – надто широка індексація, статичні шляхи вибірки та методи оцінки, що не відображають реальне середовище виконання завдань. Замість фіксованого ретривера тут працюють агенти: вони самі вирішують, що саме запитувати, як це робити і коли зупинитись. Вибірка перетворюється на ітеративний процес, що враховує контекст. Класичний RAG передбачає: один раз нарізати дані на шматки, запустити векторний пошук та отримати результат. Але насправді нарізка, вибір джерел та формулювання запитів повинні адаптуватися під завдання та дані. Agentic RAG сприймає вибірку як активну дію (рис. 2.9).

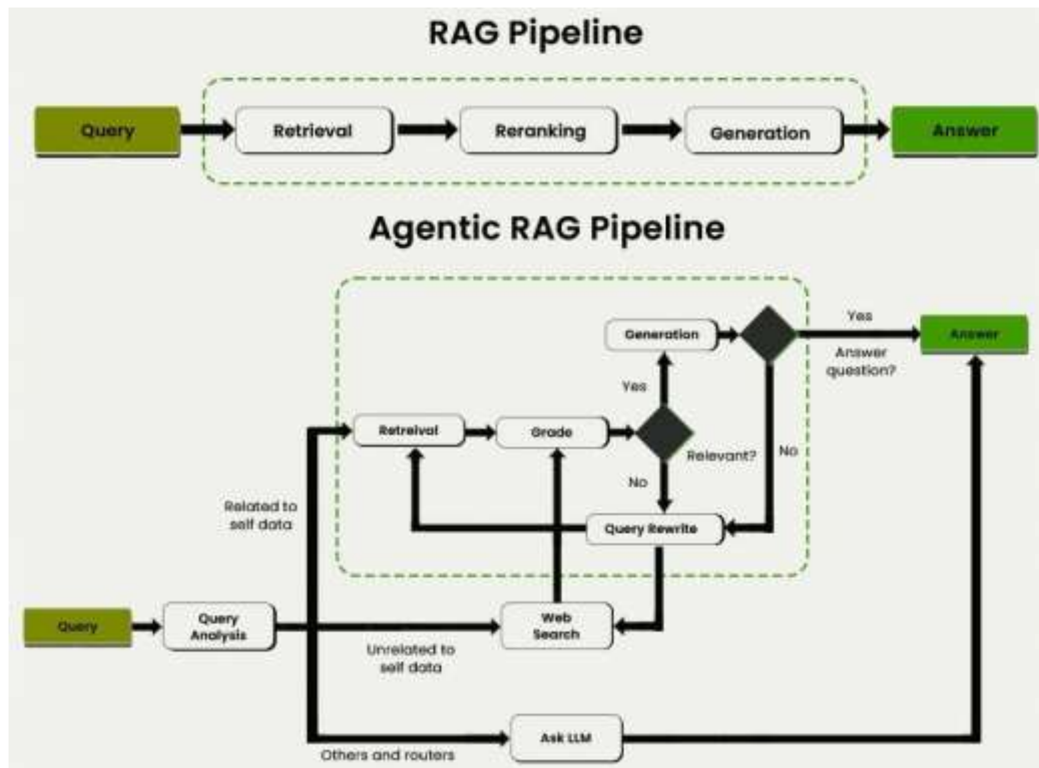

Рисунок 2.9 – Формування Agentic RAG

Агент має мету, він оцінює знайдене і вирішує, що робити далі: переформулювати запит, змінити джерело або завершити роботу, якщо контексту достатньо. Agentic RAG об'єднує генерацію за допомогою вибірки та автономне прийняття рішень. На відміну від класичного RAG, де вибірка фіксована і виконується за один прохід, Agentic RAG це динамічний ітеративний процес, керований агентом з конкретною метою.

Агент планує, як шукати, коригує стратегію за проміжними результатами та зупиняється лише тоді, коли зібрано достатній контекст. У звичайному конвеєрі RAG (верхня частина) процес лінійний: запит проходить вибірку, переранжування та генерацію, відразу надаючи відповідь. Це фіксована одноразова схема. Agentic RAG працює за циклічною моделлю: спочатку аналізує запит і вирішує, звертатися до внутрішніх даних або зовнішніх джерел. Отримані результати оцінюються релевантність, і якщо вони недостатні, агент переписує запит і запускає процес знову. Цикл продовжується, доки не буде достатньо інформації для надійної відповіді.

Архітектури Agentic RAG можуть бути від найпростіших до складних модульних. Суть у тому, що вони уникають жорсткої вибірки в один прохід і переходять до динамічної системи, де агенти приймають рішення виходячи із завдань та доступних інструментів. Найчастіше зустрічаються два базові підходи. Найпростіший варіант Agentic RAG – один агент, який приймає рішення. Такий агент працює як «розумний маршрутизатор»: отримує питання, оцінює, де краще шукати відповідь, і вибирає інструмент – векторне сховище, документну базу чи, наприклад, API на кшталт Slack чи пошуковика. Важливим є не сам вибір бази, а те, що агент динамічно вирішує, де саме шукати контекст. У складніших системах одного агента недостатньо. Multi-Agent RAG розподіляє завдання: головний агент відповідає за весь процес та делегує підзавдання спеціалізованим агентам, кожен з яких працює з певним джерелом. Наприклад, один агент обробляє технічну документацію, інший – листи та чати, третій – інформацію з інтернету. Головний агент координує роботу, щоб кожне джерело додало контекст до вихідного завдання.

Така багаторівнева структура дає модульність, гнучкість та ізоляцію помилок. Особливо корисна при роботі з корпоративними системами, де дані розкидані між публічними, приватними та напівструктурованими джерелами.

Висновки до розділу 2

Реалізація RAG вимагає врахування низки інженерних нюансів: вибору оптимального розміру та перекриття чанків, налаштування комбінованого пошуку, використання мультиплікації запитів, а також сумаризації великих текстів. Якість роботи системи визначається точністю ретрівера та релевантністю знайдених фрагментів, тому важливими є тестування, оцінка метрик та коректне формування системного промпта. За умови правильного налаштування RAG забезпечує стабільну, точну та передбачувану роботу в реальних застосуваннях.

Розкрито принципи формування ембедингів та їх використання у векторних базах даних, які забезпечують ефективний семантичний пошук у RAG-системах. Показано, що саме операції над векторами – пошук найближчих сусідів, фільтрація метаданих і гібридні методи – дозволяють швидко опрацьовувати великі масиви даних. Наведено класифікацію індексів, що визначають компроміс між точністю, швидкістю та споживанням ресурсів. У результаті доведено, що векторні БД є фундаментальним компонентом сучасних RAG-архітектур, забезпечуючи гнучкий та високопродуктивний доступ до семантичної інформації.

Систематизовано основні техніки побудови RAG-систем, що підвищують якість отримання релевантного контексту та компенсують обмеження базового підходу. Розглянуто методи покращення запитів, оптимізації структури даних та комбіновані стратегії вилучення знань. Показано, що успішність retrieval-модуля значною мірою залежить від коректного вибору векторизатора, схеми чанкінгу та механізмів реранжування. Застосування цих технік зменшує семантичний розрив між запитом і джерелами, усуває дублювання та підвищує точність генерації. Таким чином, комплексне використання наведених підходів дозволяє створювати гнучкі, стійкі та високопродуктивні RAG-архітектури. Концепція Agentic RAG демонструє еволюцію класичних RAG-систем у напрямі

автономного, контекстно залежного пошуку, що забезпечує більш точну та адаптивну роботу моделей. На відміну від статичних конвеєрів, де вибірка виконується один раз, агентні підходи реалізують динамічні цикли формування запитів і добору джерел. Це дозволяє мінімізувати помилки, спричинені надто широкою індексацією та нерелевантною інформацією. Agentic RAG підвищує ефективність завдяки здатності агентів планувати, оцінювати проміжні результати та змінювати стратегію пошуку. У результаті система забезпечує гнучке отримання знань та формування відповідей, максимально наближених до реальних потреб користувача.

РОЗДІЛ 3
РЕКОМЕНДАЦІЇ ЩОДО ВИКОРИСТАННЯ ТЕХНІК ПОШУКОВО-ПІДКРІПЛЕНОЇ ГЕНЕРАЦІЇ ДЛЯ ІНТЕГРАЦІЇ

3.1 Формування рекомендацій щодо вибору векторної БД

На даний час, існує багато публічних бенчмарків для вибору векторних БД. Але в публічних бенчмарках від самих розробників баз часто виявляється, що їхнє рішення – найкраще. Загалом, швидше за все, там ніхто не вводить в оману, але у всіх різні дані, різне апаратне забезпечення та дещо різні способи підрахунку. Як наслідок, був виконаний пошук такого інструменту. Ним виявився VectorDBBench (рис. 3.1) [29].

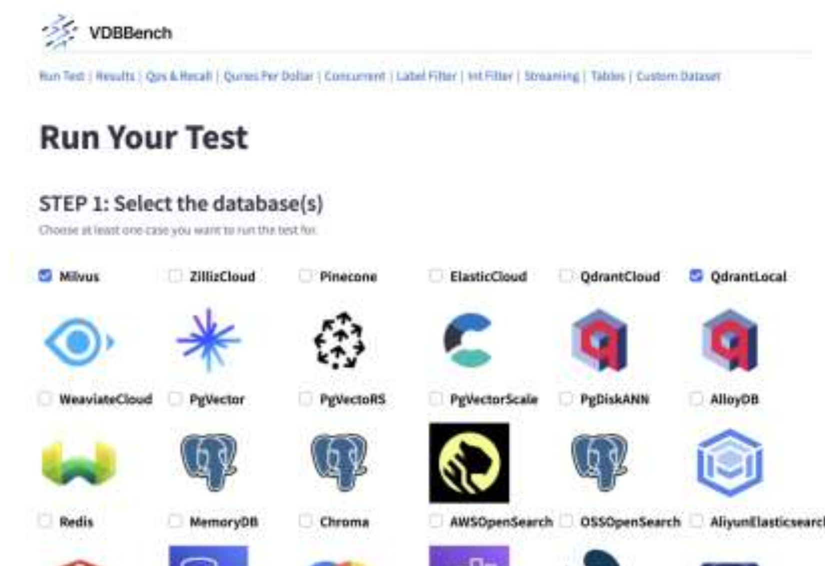

Рисунок 3.1 – ПЗ VectorDBBench для тестування векторних БД

Він встановлюється локально. Можна вибирати датасет (свій або публічний). Він обчислює всі метрики типу QPS, Latency (P50/P95/P99), Recall, Indexing time, які є досить популярними. Хоча його вигадали розробники однієї популярної векторної БД, але наявність офіційних контриб'юторів з інших БД все ж таки вселяє довіру. До того ж, всі тести треба робити на своїй інформації, своїх завданнях і своїх даних. Це є

запорукою реальної картини. На даний час, у VectorDBBench підтримуються наступні типи тестів.

1. Search Performance Test – чистий векторний пошук без фільтрів. (базова метрика для всіх БД).

2. Int-Filter Search Performance Test – векторний пошук + числові фільтри (показує деградацію при фільтрації).

3. New-Int-Filter Search Performance Test – складні комбінації числових фільтрів, (він імітує production-запити).

4. Label-Filter Search Performance Test – векторний пошук + категоріальні фільтри (category= «tech», status= «active»), виявляє post-filtering vs pre-filtering.

5. Capacity Test – максимальна кількість векторів до критичної деградації, показує реальні ліміти масштабованості.

6. Streaming Test – продуктивність пошуку при безперервній вставці нових даних, критично для систем з постійними оновленнями.

7. Custom Search Performance Test – тестування на наших даних із нашими патернами запитів.

З розгляду виключено всі БД, які фактично можна розгорнути лише в хмарі певного вендора, оскільки у 2025 р. це надзвичайно важливий аспект при виборі БД (табл. А.1).

1. Milvus [30] – відкрита розподілена векторна БД підтримує масштабування кластерів, GPU-прискорення, множину індексів (HNSW, IVF, PQ та ін). Спроектована для управління великими обсягами векторів (мільярди точок) і підтримує гібридні сценарії. Має гарну масштабованість підтримує різні режими індексів та оптимізації (GPU, диск-індекси), гнучкість для самостійного хостингу (ліцензія Open Source Software (OSS) Apache 2.0). Вона підходить для корпоративних RAG з великим обсягом даних. До мінусів відносимо такі пункти: вища складність налаштування та експлуатації (особливо кластер); потребує серйозного інфраструктурного ресурсу при великих навантаженнях; може бути оверкілом, якщо обсяг даних

невеликий і вимоги прості. Вона добре підходить для сценаріїв, коли обсяг даних великий (мільйони-мільярди векторів), потрібна висока пропускна спроможність та гнучкість (наприклад, мультимодальні дані, відео/зображення + текст), можливо власний хостинг, і коли команда готова керувати кластером.

2. Qdrant [31] – це OSS векторна БД з акцентом на payload-фільтри (метадані). Є підтримка індексів, фільтрації, гібридних запитів, оптимізації під реальні програми. Має гарний баланс між продуктивністю та гнучкістю. Сильна підтримка фільтрації за метаданими + векторів, що важливо для RAG. Масштабується, але поступається рішенням за very large scale сценаріями (наприклад, точок більше мільярду) у плані масштабу або функціональної екосистеми, а також управління кластером все ще потребує зусиль. Ця векторна БД добре підходить для проектів середнього масштабу, де потрібен семантичний пошук + метадані-фільтри, наприклад, RAG-сценарії, логіка агентів (історія, пам'ять) з кількістю точок у діапазоні мільйонів-десятків мільйонів.

3. Weaviate [32] – це OSS-векторна БД з підходом «knowledge graph/schema», тобто дозволяє задавати схеми, модулі вбудовування, гібридний пошук (ключові слова + вектори). Пропонує GraphQL API + модуль для генерації ембедингів усередині (опціонально), що спрощує інтеграцію. В той же час, може бути складною у налаштуванні, якщо просто хочете «зберігати та шукати вектори» без схеми. При дуже великих обсягах та надвисоких вимогах щодо пропускної спроможності може поступатися спеціально оптимізованим рішенням. Можлива надмірність функціоналу, якщо вимоги прості. Її варто використовувати для сценаріїв, де дані мають багату структуру (наприклад, знання, графи, складні зв'язки): RAG системи з метаданими. В цілому, характеристики векторних БД систематизовано у табл. А.1. Для вибору однієї з них можна використовувати декілька наступних рекомендацій. Якщо створюється RAG на 6M документів, то доцільно орієнтуватись на Qdrant в разі розгортання з нуля (просте

розгортання, відмінна фільтрація). Але якщо вже PostgreSQL (конкурентний з Qdrant на 10M-100M), то тоді – pgvectorscale. У ChromaDB [33] низький ліміт 1M на документи, а Milvus – складно. Якщо необхідно опрацювати певну ідею зі створення RAG, то тут стане в нагоді ChromaDB. Це рішення оптимальне для PoC, MVP, експериментів та demo за 1-2 дні. Після опрацювання ідеї формуванні >500K векторів, можна мігрувати на Qdrant або pgvectorscale. Якщо є PostgreSQL/Redis та виникає питання чи потрібна окрема векторна база даних, то тут залежить від кількості векторів. Наприклад, PostgreSQL + pgvectorscale [34] годиться якщо <100M векторів, а також команда, що знає Postgres. Окрема БД – якщо >100M, затримка <20ms критична, повільна індексація не підходить. Redis Stack [35] годиться якщо <100M векторів, критична затримка <10ms, дані розміщуються в RAM. Окрема БД, якщо дані не влазять у пам'ять (дорого) або потрібні >100M векторів. Гібридний патерн (типовий production) – структуровані дані в PostgreSQL/Redis, вектори Qdrant/Milvus, зв'язок по ID. Тепер слід розглянути коли потрібна квантизація для локальних рішень. Це стосується ситуації, якщо: >1M векторів, не вистачає RAM, розмірність ≥768D. Вона не потрібна, якщо: <500K векторів або максимальна точність критична. Гібридний пошук (вектори + BM25) потрібний, якщо точні терміни важливі (коди товарів, номери помилок, назви), запити містять рідкісні слова. Він не потрібен, якщо загальні для мови запити, точні збіги не критичні. Типова вага: 70 % вектор + 30 % BM25. Найкраща підтримка: Weaviate, ElasticSearch [36], Vespa [37].

3.2 Оцінка ефективності технік пошуково-підкріпленої генерації

Natural Questions [38] – це датасет із реальними пошуковими запитами користувачів Google. Найчастіше ці питання в датасеті є неповними, неоднозначними і потребують обробки довгого контексту. Як контекст

виступають статті з Вікіпедії. Цей датасет дуже добре лягає під сферу застосування RAG-систем: запитання написані реальними користувачами, а не є синтетикою; є великий набір документів (статті з Вікіпедії), які, як не крути, несуть багато шуму, з якого потрібно виділити релевантну інформацію; на деякі запитання немає відповідей; система повинна явно розуміти, коли відповіді немає і давати користувачеві зворотний зв'язок; для кожного питання даються фрагменти статті, які найповніше відповідають питанням (це дозволяє перевіряти релевантність як витягнутого контексту, так і фінальної відповіді); датасет містить ~300k прикладів у навчальній вибірці та ~7k прикладів у валідаційній та тестовій вибірках.

Для оцінки використовується фреймворк RAGAS [39], а також метрики BertScore [40] та ROUGE-2 [41] для аналізу релевантності вилучених чанків та фінальних відповідей. RAGAS – один із найпопулярніших фреймворків, що дозволяє проводити end-to-end оцінки роботи RAG систем. Має підтримку великої кількості необхідних метрик, яких вистачає для розуміння слабких та сильних сторін системи. У цьому дослідженні розглянемо такі метрики, як Faithfulness, Context Precision, Context Recall і Correctness.

Для оцінки RAG-технік використовуємо вибірку із 200 валідаційних питань. Таке число було вибрано як компроміс між інформативністю оцінки та практичними обмеженнями (вартість запитів до LLM через API). Як емпіричне обґрунтування рішення візьмемо 500 відповідей стандартної RAG-системи та оцінимо їх на вірність за бінарною шкалою (1 – відповідь системи рівносильна еталонній відповіді, 0 – інакше). Отримані результати наведено в табл. А.2, а також на графіках, що наведено на рис. 3.2.

Перша метрика – Faithfulness. Вона показує узгодженість між відповіддю LLM і даним контекстом. Грубо кажучи, якщо відповідь містить лише ті факти, які є в контексті, то дана метрика буде високою. Багато техніки лежать у діапазоні 0,66-0,73. Особливо вибивається з цього діапазону, як і у всіх метриках, Proposition Chunking. Також особливо високу оцінку має Query Decomposition, але це можна пояснити конструктивною

особливістю. Контекстом цієї техніки є відповіді на підзапити. Proposition Chunking добре працюватиме на невеликих обсягах даних, у яких користувачам будуть потрібні короткі факти. А поточний датасет не зовсім підходить для Proposition Chunking техніки, оскільки у статтях Вікіпедії багато шуму.

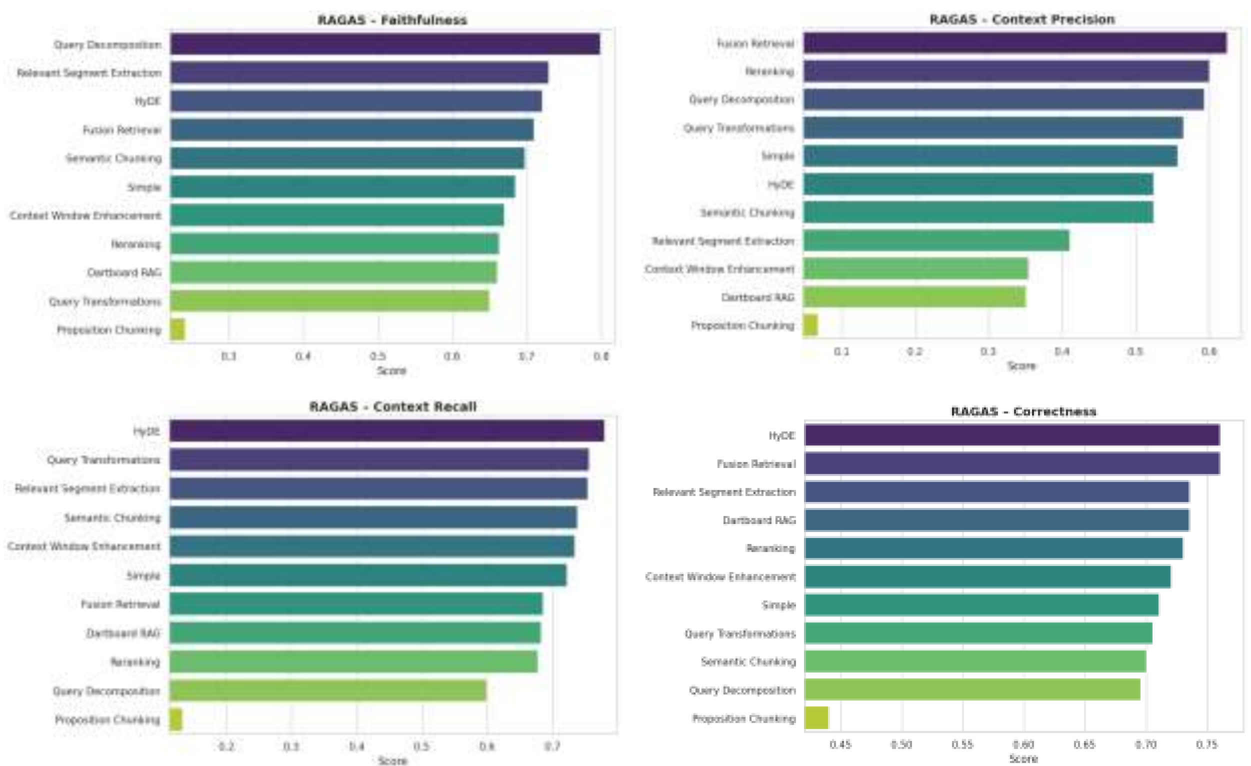

Рисунок 3.2 – Метрики для оцінки технік RAG

Наступна метрика – Context Precision. На ній видно просідання в таких техніках, як Dartboard RAG, Context Window Enhancement і Relevant Segment Extraction. Цей факт свідчить, що у контексті багато непотрібної інформації. Таке просідання легко пояснити: у Dartboard RAG контекст формується з чанків, які не лише семантично схожі на запит, а й семантично відмінні від вже відібраних чанків; у Relevant Segment Extraction можуть додаватись чанки, які знаходяться між релевантними, що додає зайвої інформації; у Context Window Enhancement ми завжди додаємо сусідні чанки, що не завжди додає релевантну інформацію (такий метод хороший при роботі з документами або звітами, в яких на одній сторінці найчастіше розміщується

близька за змістом інформація). У середньому техніки мають оцінку, близьку до 0,5-0,6, що не зовсім добре.

Щоб не покладатися тільки на оцінки RAGAS, які залежать від роботи LLM, скористаємося ще однією метрикою – BertScore, за допомогою якої оцінимо релевантність контексту і відповіді. Дана метрика працює на передбачуваній мовній моделі, яка також, як і LLM, здатна вловлювати семантичні зв'язки. У датасеті Natural Questions крім Short Answer є також Long Answer. Тому братимемо кожен отриманий чанк і порівнюватимемо з Long Answer за допомогою BertScore. Підсумкову відповідь порівнюватимемо, як і в RAGAS, з Short Answer (рис. 3.3).

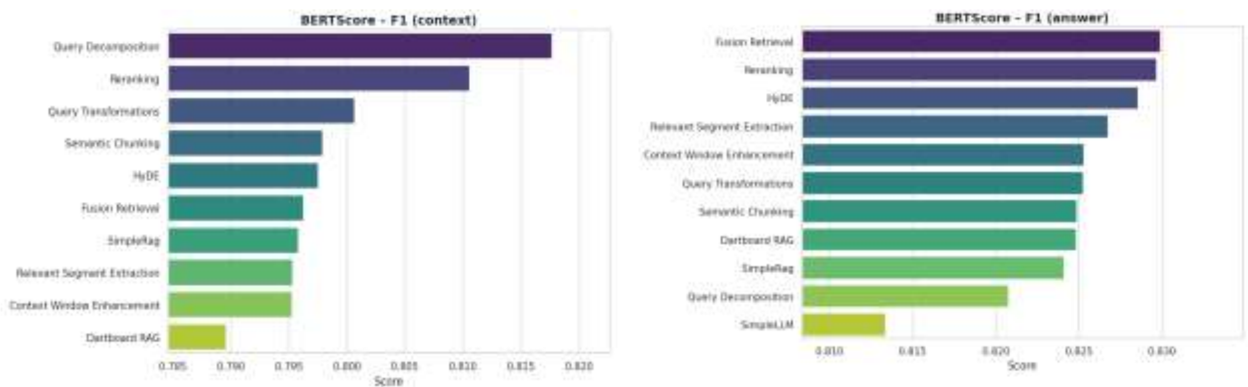

Рисунок 3.3 – Метрика BertScore

Як бачимо, для BertScore метрики ранжування залишаються схожими. Знову Fusion Retrieval, Reranking RAG та HyDE показали добрі результати. Тепер розглянемо ROUGE-2 (рис. 3.4). Це класична метрика, яка порівнює біграми між згенерованим текстом та еталоном. Ця метрика проста та інтерпретована: що більше слів збіглося, то краще. Але вона не враховує семантику та чутлива до перефразування та помилок. Як і в минулих метриках, можна виділити 4 метрики, які відпрацювали краще за інших: Fusion Retrieval, Reranking RAG і HyDE. Як і передбачалося, кореляція між метриками є. Між метриками контексту кореляція слабша. Механізм оцінки релевантності з метриками BertScore та ROUGE був побудований на оцінці чанків щодо еталонного уривка тексту, тому можна помітити кореляцію з

Context Precision та зворотну кореляцію з Context Recall. Надалі наведемо коротку оцінку кожної техніки RAG, що досліджувалась.

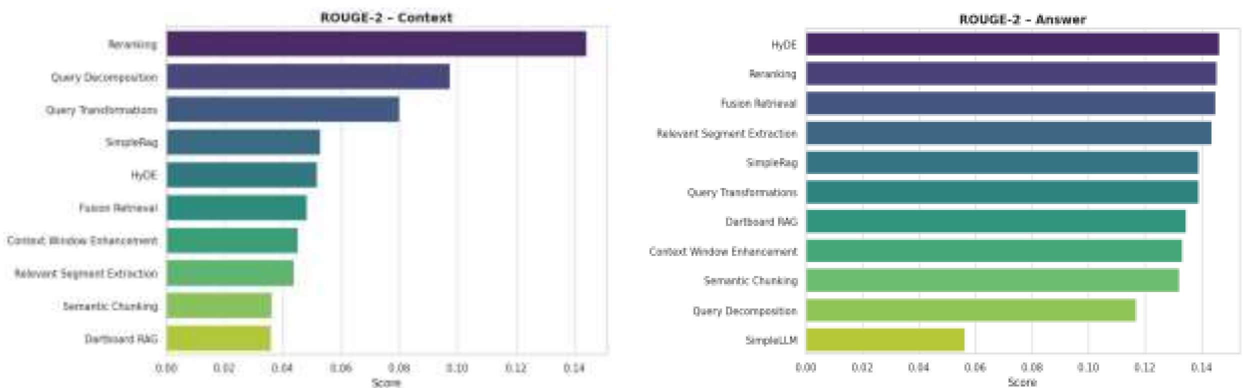

Рисунок 3.4 – Метрика BertScore

Simple RAG працює помітно краще за просту LLM (доля вірних відповідей з 52 збільшилася до 70 %). Просто реалізується, за часом працює майже так само, як і проста LLM (0,4 секунди на запит). «З коробки» може показувати гарний результат.

Query Transformation. По суті, це той самий простий RAG, тільки питання користувача трохи коригується, якщо це необхідно. Власне тому можна помітити невелике (з урахуванням похибок) поліпшення релевантності контексту (особливо помітно на метриці ROUGE-context +60 %), але якість відповіді залишається приблизно однаковою, це пов'язано з тим, що з'являється слабка ланка у вигляді LLM, що перетворює запит. Деякі питання LLM може неправильно переформулювати.

Query Decomposition за характеристиками ця техніка близька до попередньої. Відмінно працює на складних питаннях, але доводиться розплачуватися дорожнечею. Щодо звичайного RAG, час роботи збільшується майже на 60 % (з 1,6 до 2,8 секунд на запит). На даному датасеті техніка відпрацювала не найкращим чином (результати близькі до простого RAG) через прості питання, які не вимагають розбиття на підзапити. Також варто розуміти, що дана система працюватиме гірше через свою паралельність.

HyDE входить у топ найкращих технік у цьому дослідженні. Причиною може бути те, що пропадає семантичний розрив між питанням та оповідально-викладеними чанками. Час роботи порівняно з простою RAG збільшується приблизно на 40 %, але й суттєво збільшується частка вірних відповідей (0,7 → 0,76) та ROUGE-answer метрика (0,06 → 0,14).

Relevant Segment Extraction також добре себе показала. Майже на всіх графіках видно покращення якості відповідей (correctness: 0,7 → 0,74), проте гірше, ніж HyDE. Але RSE не потребує додаткових викликів LLM. Однак, складність з'являється при пошуку чанків, тому що для цього потрібно отримати їх усі та обробити. Тому необхідно проводити оптимізацію, щоби це не займало стільки часу, скільки у мене.

Context Window Enhancement як і RSE, дана техніка додає шум у контекст, що видно за метрикою RAGAS-Precision, але цей шум може нести корисну інформацію (що видно по Recall), яка збільшує якість відповідей (на метриках RAGAS і BertScore). За часом CWE працює так само, як і проста RAG. Але якщо не зменшити кількість чанків, які спочатку дістаються, то можна в 3 рази збільшити обсяг контексту, порівняно з простою RAG, що може впливати на якість роботи LLM і здорожчення однієї відповіді. Тому в цьому дослідженні кількість одержуваних чанків була зменшена в 3 рази (10 → 3).

Semantic Chunking не дало особливої переваги через будову документів. Статті на Вікіпедії часто досить добре структуровані та кожен абзац несе у собі нову думку. З цієї причини добре спрацьовує звичайне рекурсивне розбиття (Recursive Character Text Splitter). Налаштувати ж рекурсивне розбиття було складно через те, що в текстах бувають занадто марні ділянки (довгі перерахування або джерела з величезною кількістю рядків). Тому при збільшенні значення точки розбиття (breakpoint_threshold_type) виходили величезні фрагменти тексту, в якому розмивалася семантика, а при зменшенні – збільшувалась загальна кількість чанків, що призводило до ситуації з Proposition Chunking.

Fusion Retrieval – одна з кращих на датасеті технік, яка має 76 % вірних відповідей і працює зі швидкістю звичайного RAG (отже, і споживає токенів приблизно стільки ж). Fusion Retrieval поєднує в собі переваги семантичних пошукачів з повнотекстовими, що критично при великих базах даних. Однак, для цієї техніки часто необхідно додавати Reranker-модуль (змішування технік у цьому дослідженні зачіпати вже не будемо), який допоможе впоратися з шумом.

Reranking RAG через фільтрацію чанків має хороші показники метрик, пов'язаних із контекстом. Але за метриками, які оцінюють відповідь, є невелике відставання від лідера (0,73 на відміну від 0,76 у лідера). Складним є процес налаштування промпту та процесу оцінки, щоб релевантні чанки не видалялися, а нерелевантні не потрапляли до контексту.

Dartboard RAG у середньому відпрацювала трохи краще простого RAG з точки зору RAGAS і BertScore, хоча через внесення різноманітності в контекст видно, що просідають метрики, пов'язані з контекстом, що говорить про зашумленість. Тому добре було б додавати Reranker-модуль для фільтрації.

В цілому, вдалося розібрати підходи щодо застосування техніки, порівняти їх на практиці, показати їх сильні та слабкі сторони, а також ті виклики, що виникають під час застосування. Багато деталей виявляються важливішими, ніж здається на перший погляд: зміна шкали реранкера помітно впливає на результат, зміна функції в RSE змінює саму логіку роботи. Метрики в дослідженні були використані як інструмент для виявлення цих особливостей.

3.3 Економічне обґрунтування прийнятих рішень

Інтеграція технік RAG та LLM з базою знань компанії дозволяє автоматизувати обробку інформації, скоротити час пошуку документів та

зменшити навантаження на експертів. У сучасних компаніях обсяг внутрішньої документації щорічно зростає, що збільшує операційні витрати, пов'язані з комунікацією, підтримкою співробітників та прийняттям рішень. Впровадження RAG-технологій забезпечує суттєве підвищення ефективності за рахунок: скорочення часу доступу до релевантних знань; зниження вартості експертних консультацій; оптимізації бізнес-процесів; підвищення точності рішень завдяки актуальним даним із корпоративної бази знань.

Надалі проведемо приблизний аналіз витрат на впровадження запропонованої системи. Прямі витрати на впровадження системи пошуково-підкріпленої генерації охоплюють кілька ключових складових, пов'язаних із технічною інфраструктурою, програмним забезпеченням, інтеграційними роботами та підготовкою персоналу. Передусім значну частку становлять витрати на інфраструктуру, які включають оренду або використання локальних серверних ресурсів, необхідних для забезпечення стабільної роботи системи, а також організацію зберігання векторної бази даних на таких платформах, як Weaviate, Pinecone чи Milvus. Крім того, для функціонування великих мовних моделей потрібні суттєві обчислювальні ресурси – GPU- або CPU-кластери, або ж доступ через API, що передбачає додаткові фінансові витрати. Другу групу прямих витрат становить програмне забезпечення. До них належать ліцензії на використання спеціалізованих платформ векторних баз даних, а також оплата роботи LLM-моделей у разі використання зовнішніх API, де вартість розраховується залежно від обсягу оброблених токенів. Важливою статтею витрат є також розробка та інтеграція: робота інженерів з побудови RAG-pipeline, налаштування процесів індексації корпоративних документів та інтеграція рішення з існуючими внутрішніми IT-системами компанії. Окремо слід враховувати витрати на навчання персоналу, оскільки ефективна робота із системою потребує проведення спеціалізованих тренінгів та оновлення технічної документації для користувачів і адміністраторів. Поряд із прямими витратами у процесі впровадження виникають і непрямі витрати. До них

належать тимчасове зниження продуктивності співробітників під час адаптації до нових інструментів, витрати на аудит і підготовку даних перед індексацією, а також потенційні додаткові витрати, пов'язані з підвищенням рівня інформаційної безпеки, що є особливо актуальним у разі роботи з конфіденційними корпоративними знаннями. У сукупності ці витрати формують економічну базу для розрахунку окупності та доцільності впровадження системи.

1. Скорочення часу пошуку інформації. За статистикою, середній спеціаліст витрачає до $T_1 = 20\ldots30\%$ робочого часу на пошук внутрішньої інформації. Якщо RAG скорочує ці витрати хоча б на $\Delta T_1 = 50\%$, економічний ефект для компанії з 50 співробітниками буде:

$$E_{час} = 50 \cdot 20\% \cdot 0{,}5 \cdot C_{год}, \tag{3.1}$$

де $C_{год}$ – середня вартість години працівника.

2. Скорочення витрат на експертні консультації. RAG дозволяє автоматизувати відповіді на повторювані питання, зменшуючи потребу у залученні висококваліфікованих спеціалістів:

$$E_{експерт} = (N_{запитів} \cdot C_{консультації}) \cdot k, \tag{3.2}$$

де k – частка автоматизованих відповідей (зазвичай, 0,4…0,7).

3. Підвищення точності рішень. Покращення якості відповідей LLM завдяки корпоративним даним зменшує ризик неправильних рішень. Неможливо оцінити точно $E_{точність}$, але у більшості компаній втрати від помилок складають 2…5 % операційних витрат. Зниження помилок на 30…40 % вже приносить відчутний економічний результат.

Надалі розрахуємо рентабельність (ROI).

$$ROI = \frac{E_{час} + E_{експерт} + E_{точність} - B_{загальні}}{B_{загальні}} \cdot 100\%, \tag{3.3}$$

де $B_{загальні}$ – сумарні витрати на впровадження та підтримку.

Розглянемо типовий приклад розрахунку для умовної компанії з 50 працівниками. Згідно табл. 3.1, річний ефект складає 510000 грн, а річні

витрати – 120000 грн. Відповідно до (3.3) *ROI* дорівнює 325 %. Тобто, система окупається менш ніж за 5 місяців, що підтверджує економічну доцільність інтеграції RAG-рішень.

Таблиця 3.1 – Економічні показники впровадження RAG

Стаття	Сума, грн
Витрати на впровадження (одноразово)	180000
Щорічна підтримка	120000
Економія часу співробітників	350000
Зменшення експертних витрат	90000
Підвищення точності рішень	70000

Економічний аналіз демонструє, що інтеграція технік пошуково-підкріпленої генерації з базою знань компанії є фінансово виправданим рішенням. Система забезпечує значне скорочення операційних витрат, зменшує навантаження на персонал і підвищує ефективність бізнес-процесів. Завдяки високому рівню окупності та потенціалу масштабування технології RAG можуть стати ключовим елементом цифрової трансформації компанії.

Висновки до розділу 3

В роботі систематизовано підходи до оцінки та вибору векторних баз даних, що дозволило визначити їх реальні можливості в умовах різних навантажень і сценаріїв використання. Використання інструмента VectorDBBench дало змогу об’єктивно порівняти продуктивність рішень та оцінити вплив фільтрації, масштабованості та потокових оновлень на ефективність пошуку. Аналіз Milvus, Qdrant, Weaviate та інших БД продемонстрував, що вибір оптимальної системи значною мірою залежить від обсягу даних, вимог до швидкодії та можливостей інфраструктури. Окремо підкреслено важливість квантизації, гібридного пошуку та інтеграції з традиційними СУБД у високонавантажених системах. Сформульовані

рекомендації забезпечують практичні орієнтири для побудови ефективних RAG-рішень і підкреслюють необхідність тестування на власних даних як ключової передумови коректного вибору архітектури.

проведено комплексну оцінку ефективності різних технік пошуково-підкріпленої генерації, що дозволило визначити їх сильні та слабкі сторони в умовах роботи з реальними даними. Використання датасету Natural Questions та метрик RAGAS, BertScore і ROUGE забезпечило об'єктивне зіставлення якості витягнутого контексту та фінальних відповідей. Дослідження показало, що техніки HyDE, Fusion Retrieval та Reranking RAG демонструють найвищі результати, тоді як інші підходи мають обмеження через шум у контексті або конструктивні особливості. Виявлено також, що зміна логіки формування контексту, параметрів реранкінгу чи способу сегментації суттєво впливає на підсумкову точність. Загалом результати підтверджують важливість ретельного підбору технік RAG відповідно до структури даних та вимог системи, що є критичним для побудови надійних і продуктивних рішень.

Проведений економічний аналіз підтвердив доцільність інтеграції технік пошуково-підкріпленої генерації в корпоративні системи управління знаннями. Розрахунки показали, що впровадження RAG-рішень значно скорочує витрати часу на пошук інформації, зменшує потребу в експертних консультаціях і підвищує точність прийняття рішень. Врахування прямих та непрямих витрат дозволило сформувати реалістичну оцінку інвестицій, необхідних для запуску такої системи. При цьому високий рівень рентабельності та швидке повернення інвестицій свідчать про економічну ефективність технології навіть для компаній середнього масштабу. Використання RAG-технологій є не лише технологічно обґрунтованим, а й фінансово вигідним компонентом цифрової трансформації бізнесу.

ВИСНОВКИ

Ефективність RAG значною мірою залежить від коректного налаштування технічних елементів, зокрема параметрів чанкінгу, методів пошуку та прийомів роботи з великими текстами. Важливу роль відіграють точність ретривера й релевантність вибраних фрагментів, що потребує системного тестування та аналізу метрик. За умови правильної конфігурації RAG-система демонструє стабільну й передбачувану роботу у практичних застосуваннях.

Як наслідок, узагальнено принципи формування ембедингів та показано їх ключову роль у семантичному пошуку. Операції над векторами забезпечують швидке опрацювання даних, а вибір типу індексу визначає баланс між точністю та швидкодією. Доведено, що векторні бази даних є основою сучасних RAG-проєктів, оскільки надають гнучкий і продуктивний доступ до знань. Систематизація технік побудови RAG показала, що якість отриманого контексту залежить від ефективності векторизації, схеми сегментації даних і механізмів реранжування. Методи покращення запитів та комбіновані стратегії пошуку зменшують семантичний розрив між питанням і джерелами та підвищують точність генерації. Agentic RAG продемонстрував подальший розвиток підходу, забезпечивши динамічний добір інформації та адаптивну поведінку системи відповідно до потреб користувача.

Узагальнено підходи до вибору векторних баз даних та показано, що їхня ефективність визначається масштабом даних, вимогами до швидкодії й доступною інфраструктурою. Тестування за допомогою VectorDBBench дозволило об’єктивно оцінити вплив фільтрації, потокових оновлень та індексації на продуктивність систем. Підкреслено важливість гібридного пошуку, квантизації та інтеграції з реляційними СУБД у складних RAG-сценаріях. Запропоновані рекомендації допомагають коректно обрати архітектуру та підтверджують необхідність тестування на власних даних.

Аналіз технік RAG продемонстрував відмінності в їхній результативності та чутливість до параметрів сегментації, реранжування і способів обробки контексту. Метрики RAGAS, BertScore і ROUGE показали переваги підходів HyDE, Fusion Retrieval і Reranking RAG, тоді як інші методи виявили обмеження, пов’язані з шумом або структурою даних. Результати підкреслюють, що правильний вибір техніки та конфігурації ретривера є ключовою умовою надійної роботи RAG-систем.

Економічна оцінка довела, що впровадження RAG-рішень є фінансово виправданим і сприяє суттєвому скороченню витрат на пошук інформації та експертну підтримку. За умови врахування витрат на інфраструктуру та інтеграцію такі системи демонструють високу рентабельність і швидко окуповуються (менш ніж за 5 місяців). У підсумку встановлено, що RAG-технології є ефективним інструментом цифрової трансформації та підвищення продуктивності бізнес-процесів.

Таким чином, результатами роботи є систематизація технік пошуково-підкріпленої генерації для інтеграції з базою знань компанії; оцінка ефективності використання технік пошуково-підкріпленої генерації; рекомендації щодо використання технік пошуково-підкріпленої генерації для інтеграції з базою знань компанії. Вони можуть бути використані для застосування в системах підтримки прийняття рішень, службах технічної підтримки, консультаційних сервісах та в проектах цифрової трансформації та подальших досліджень за даною тематикою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Силантьєв В.М., Шишлін В.О. Інтеграція генеративного AI і RAG для вирішення питань логістики. *Сучасні аспекти та перспективні напрямки розвитку науки: матеріали X Міжнародної студентської наукової конференції* (жовтень 2025 р. м. Луцьк), 2025. С. 235, 236.

2. Raschka S. Beyond Standard LLMs. URL: https://magazine.sebastianraschka.com/p/beyond-standard-llms (дата звернення: 27.11.2025).

3. OpenAI. URL: https://openai.com (дата звернення: 27.11.2025).

4. Google Gemini. URL: https://gemini.google.com/app (дата звернення: 27.11.2025).

5. Bhoyar V. RAG Models Decoded: From Theory to Practice. Independently published, 2024. 96 p.

6. Ferrer J. Fine-Tuning LLMs: A Guide With Examples. URL: https://www.datacamp.com/tutorial/fine-tuning-large-language-models (дата звернення: 27.11.2025).

7. Slyusar V.I. Applications of Large Language Models in the Military Sphere. Artificial Intelligence: Achievements and Recent Developments. *Series in Computing and Information Science and Technology*. River Publishers, 2025. P. 53-82.

8. LoRA (Low-Rank Adaptation). URL: https://huggingface.co/learn/llm-course/chapter11/4 (дата звернення: 27.11.2025).

9. Mohan N.S. Finetuning Large language models using QLoRA. URL: https://chatgpt.com/c/693a2b1e-1c98-832b-8572-daf6b6f6eae3 (дата звернення: 27.11.2025).

10. Enen M., Saad S., Nazmy T. A survey on retrieval-augmentation generation (RAG) models for healthcare applications. *Neural Computing and Applications*. 2025. 37(33). P. 28191-28267. DOI: 10.1007/s00521-025-11666-9.

11. Sage E.W. Retrieval-Augmented Generation for NLP Practitioners. Independently Published, 2024. 390 p.

12. Kimothi A. A Simple Guide to Retrieval Augmented Generation. Manning Publications, 2025. 256 p.

13. Vector embeddings. URL: https://platform.openai.com/docs/guides/embeddings (дата звернення: 27.11.2025).

14. Zhu C. The Development Status of Retrieval-Augmented Generation (RAG) Technology. *Applied and Computational Engineering*. 2025. 207(1). P. 125-134. DOI: 10.54254/2755-2721/2026.TJ30058.

15. Llama 4: Leading intelligence. Unrivaled speed and efficiency. URL: https://www.llama.com (дата звернення: 27.11.2025).

16. The new standard for complex document processing. URL: https://www.llamaindex.ai (дата звернення: 27.11.2025).

17. Engineer reliable agents. Ship agents to production with LangChain's comprehensive platform for agent engineering. URL: https://www.langchain.com/ (дата звернення: 27.11.2025).

18. Tok E.B. Chunking in LLMs (Large Language Models). *Medium*. URL: https://medium.com/@elifbeyzatok/chunking-in-llms-large-language-models-450687c4378a (дата звернення: 27.11.2025).

19. Шишлін В. Застосування механізму Re-Ranking для завдань пошуку на основі LLM у корпоративних базах знань. *Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій: матеріали XXII щорічного міждисциплінарного семінару* (листопад 2025 р., м. Полтава), 2025. С. 114, 115.

20. Aggarwal A. RAG Made Simple: A Beginner’s Journey to Smarter AI Applications. Part 1. *Medium*. URL: https://medium.com/@arushiagg04/rag-made-simple-a-beginners-journey-to-smarter-ai-applications-8f68354d97f7 (дата звернення: 27.11.2025).

21. Chunking Strategies for LLM Applications. *Pinecone*. URL: https://www.pinecone.io/learn/chunking-strategies/ (дата звернення: 27.11.2025).

22. BGE-M3. URL: https://huggingface.co/BAAI/bge-m3 (дата звернення: 27.11.2025).

23. Query Transformations. *LangChain Blog*. URL: https://blog.langchain.com/query-transformations/ (дата звернення: 27.11.2025).

24. Bhat V.N. Query Decomposition: Understanding the User's Perspective. *Sahai*.URL: https://sahaj.ai/query-decomposition-understanding-the-users-perspective/ (дата звернення: 27.11.2025).

25. Nigam G. A Complete Guide to Implementing HyDE RAG. Medium. URL: https://medium.com/aingineer/a-complete-guide-to-implementing-hyde-rag-82492551f3d8 (дата звернення: 27.11.2025).

26. Nayak P. Semantic Chunking for RAG. *Medium*. URL: https://medium.com/the-ai-forum/semantic-chunking-for-rag-f4733025d5f5 (дата звернення: 27.11.2025).

27. Mindek F. RAG Strategies - Context Enrichment. RAG strategies. *Pixion*. URL: https://pixion.co/blog/rag-strategies-context-enrichment (дата звернення: 27.11.2025).

28. Yadav A., Yadav B. Re-ranking in Retrieval Augmented Generation: How to Use Re-rankers in RAG. Chitika. URL: https://www.chitika.com/re-ranking-in-retrieval-augmented-generation-how-to-use-re-rankers-in-rag/ (дата звернення: 27.11.2025).

29. VectorDBBench. URL: https://github.com/zilliztech/VectorDBBench (дата звернення: 27.11.2025).

30. The High-Performance Vector Database Built for Scale. URL: https://milvus.io (дата звернення: 27.11.2025).

31. High-Performance Vector Search at Scale. *Qdrant*. URL: https://qdrant.tech (дата звернення: 27.11.2025).

32. For AI engineers who think big. *Weaviate*. URL: https://weaviate.io (дата звернення: 27.11.2025).

33. Abdel-Aziz N. Build your first RAG using Python / ChromaDB / OpenAI. *Medium*. URL: https://medium.com/@nermeen.abdelaziz/build-your-first-python-rag-using-chromadb-openai-d711db1abf66 (дата звернення: 27.11.2025).

34. Rathore R. Postgres Vector Search with pgvector: Benchmarks, Costs, and Reality Check. *Medium*. URL: https://medium.com/@DataCraft-Innovations/postgres-

vector-search-with-pgvector-benchmarks-costs-and-reality-check-f839a4d2b66f (дата звернення: 27.11.2025).

35. Your App is about to get faster. *Redis*. URL: https://redis.io (дата звернення: 27.11.2025).

36. The open source platform that powers search, observability, security, and more .... *Elastic*. URL: https://www.elastic.co (дата звернення: 27.11.2025).

37. Vespa. URL: https://vespa.ai/solutions/visual-retrieval-augmented-generation/ (дата звернення: 27.11.2025).

38. Natural Questions. URL: https://github.com/google-research-datasets/natural-questions (дата звернення: 27.11.2025).

39. Ragas. URL: https://docs.ragas.io/en/stable/ (дата звернення: 27.11.2025).

40. RAG Benchmarking: Comparing RAGAS, BERTScore, and Giskard for AI Evaluation. *Giskard*. URL: https://www.giskard.ai/knowledge/rag-benchmarking-for-ai-evaluation (дата звернення: 27.11.2025).

41. ROUGE (metric). *Wikipedia*. URL: https://en.wikipedia.org/wiki/ROUGE_(metric) (дата звернення: 27.11.2025).