Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Інформаційних управляючих систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів формування запитів до великих мовних моделей з використанням векторної бази даних

(тема)

Виконав:
здобувач 2 року навчання,
групи ІУСТм-24-1
Сухоруков Данило Артурович
(прізвище, ім'я, по батькові)

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

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформаційні управляючі системи та технології
(повна назва освітньої програми)

Керівник: проф. каф. ІУС Чалий С.Ф.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ІУС _____ Петров К.Е.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Інформаційних управляючих систем

Рівень вищої освіти другий (магістерський)

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

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформаційні управляючі системи та технології
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

" 24 " листопада 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Сухорукову Данилу Артуровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів формування запитів до великих мовних моделей з використанням векторної бази даних

затверджена наказом по університету від " 24 " листопада 2025 р. № 1055Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії " 15 " листопада 2025 р.

3. Вихідні дані до роботи науково-технічні публікації; джерела інтернету, науково-технічна література, що стосується теми кваліфікаційної роботи

4. Перелік питань, що потрібно опрацювати у роботі Аналіз властивостей великих мовних моделей; аналіз методів векторного пошуку та формування контексту до великих мовних моделей; аналіз існуючих методів удосконалення запитів; постановка задачі дослідження; підхід до вирішення задачі; розробка удосконаленого методу формування запитів; опис інформаційної технології; імплементація інформаційної технології; опис програмного модуля; опис експериментальних даних.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз властивостей великих мовних моделей	24.11.2025	Виконано
2	Аналіз методів векторного пошуку та формування	26.11.2025	Виконано
	контексту до великих мовних моделей		
3	Аналіз існуючих методів удосконалення запитів	28.11.2025	Виконано
4	Постановка задачі дослідження	30.11.2025	Виконано
5	Підхід до вирішення задачі	01.12.2025	Виконано
6	Розробка удосконаленого методу формування	03.12.2025	Виконано
	запитів		
7	Опис інформаційної технології	06.12.2025	Виконано
8	Імплементація інформаційної технології	07.12.2025	Виконано
9	Опис програмного модуля	08.12.2025	Виконано
10	Опис експериментальних даних	09.12.2025	Виконано
11	Оформлення пояснювальної записки	10.12.2025	Виконано
12	Здача роботи для перевірки на нормоконтроль	11.12.2025	Виконано
13	Підготовка презентації	14.12.2025	Виконано
14	Попередній захист	15.12.2025	Виконано
15	Захист кваліфікаційної роботи в екзаменаційній	17.12.2025	Виконано
	комісії		

Дата видачі завдання 24 листопада 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. каф. ІУС Чалий С.Ф.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 120 с., 13 рис., 12 табл., 2 дод., 38 джерел.

ВЕКТОРНА БАЗА ДАНИХ, ВЕЛИКІ МОВНІ МОДЕЛІ, ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, КОМПРЕСІЯ КОНТЕКСТУ, ПОВТОРНЕ РАНЖУВАННЯ, СЕМАНТИЧНИЙ ПОШУК, ФОРМУВАННЯ ЗАПИТІВ, RETRIEVAL-AUGMENTED GENERATION.

У роботі виконано аналіз сучасного стану об’єкта дослідження. Оглянуто існуючі варіанти задачі формування запитів до великої мовної моделі. Запропоновано модифікацію метода формування контексту до запиту користувача шляхом використання підсумків попередніх діалогів.

Об’єктом дослідження кваліфікаційної роботи є процес формування запитів до великих мовних моделей.

Предметом дослідження є методи побудови запитів із використанням векторного контексту.

Метою роботи є розробка метода формування запитів до LLM з автоматичною інтеграцією релевантного контексту з векторної бази для підвищення якості відповідей.

Для досягнення даної методи необхідно вирішити наступні питання:

– аналіз методів формування запитів та RAG-підходів;

– розробка методу інтеграції результату векторного пошуку з попередніх бесід у запит користувача до великої мовної моделі;

– розробка інформаційної технології формування запитів до великих мовних моделей;

– експериментальна перевірка розробленого методу.

ABSTRACT

Master’s thesis: 120 pages, 13 figures, 12 tables, 2 appendices, 38 sources.

CONTEXT COMPRESSION, INFORMATION TECHNOLOGY, LARGE LANGUAGE MODELS, PROMPT ENGINEERING, RE-RANKING, RETRIEVAL-AUGMENTED GENERATION, SEMANTIC SEARCH, VECTOR DATABASE.

The paper analyzes the current state of the research object. Existing options for forming queries to a large language model are reviewed. A modification of the method for forming context for a user query by using the results of previous dialogues is proposed.

The object of the thesis is the process of forming queries to large language models.

The subject of the study is methods of constructing queries using vector context.

The goal of the work is to develop a method for forming queries to LLM with automatic integration of relevant context from a vector database to improve the quality of responses.

To achieve this, the following issues must be addressed:

– analysis of query formation methods and RAG approaches;

– development of a method for integrating vector search results from previous conversations into a user query to a large language model;

– development of information technology for forming queries to large language models;

– experimental verification of the developed method.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

СУБД – система управління базами даних

API – application programming interface

BERT – bidirectional encoder representations from transformers

BM25 – best match 25

FAISS – facebook artificial intelligence similarity search

GPT – generative pre-trained transformer

HTML – hypertext markup language

IR – information retrieval

LLM – large language model

MRR – mean reciprocal rank

RAG – retrieval-augmented generation

SBERT – sentence bidirectional encoder representations from transformers

TF-IDF – term frequency-inverse document frequency

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням обсягів неструктурованих текстових даних, які накопичуються в корпоративних сховищах та домен-орієнтованих інформаційних системах. Класичні пошукові технології, орієнтовані переважно на прості статистичні моделі, виявляються недостатньо ефективними для побудови відповідей природною мовою. Поява та швидкий розвиток великих мовних моделей (Large Language Models, LLM) змінили уявлення про можливості автоматизованої обробки тексту, проте практичне використання таких моделей у реальних прикладних системах виявило низку обмежень, пов’язаних із розміром контекстного вікна.

Актуальність даної кваліфікаційної роботи зумовлена проблемою LLM зберегти лише обмежений обсяг контексту та не мати доступу до історії взаємодій користувача поза межами певної кількості повідомлень у поточної сесії. Використання векторних баз даних як механізму розширення контексту надає можливість інтегрувати у запит фрагменти попередніх діалогів, що підвищує релевантність відповіді.

Метою роботи є розробка метода формування запитів до LLM з автоматичною інтеграцією релевантного контексту з векторної бази для підвищення якості відповідей.

Для досягнення даної методи необхідно вирішити наступні питання:

– аналіз методів формування запитів та Retrieval-Augmented Generation (RAG)-підходів;

– розробка методу інтеграції результату векторного пошуку з попередніх бесід у запит користувача до великої мовної моделі;

– розробка інформаційної технології формування запитів до великих мовних моделей;

– експериментальна перевірка розробленого методу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз властивостей великих мовних моделей

LLM – це різновид генеративних нейронних мереж, здатних продукувати зв’язний текст у відповідь на текстовий запит природною мовою [1]. Взаємодія користувача або зовнішньої системи з LLM здійснюється через формування запиту (prompt), де модель отримує на вхід певний текст – питання, інструкцію чи інший контекст – і генерує на його основі продовження або відповідь. Отже, процес побудови запиту є центральним механізмом спілкування з моделлю, визначаючи на що модель реагуватиме.

Правильно сформульований запит надає моделі необхідну інформацію та інструкції, тоді як некоректний або розпливчастий запит може бути інтерпретований неправильно. Іншими словами, якість відповіді LLM значною мірою залежить від того, як користувач сформулював свій запит [2]. Це підтверджується науковим дослідженням, яке показує, що один і той самий мовний модуль може давати різну точність і доречність відповіді залежно від чіткості та повноти інструкції у prompt [2].

Сучасні LLM базуються на трансформерній архітектурі з механізмом самоуваги, запропонованим Васвані у 2017 році [1]. Механізм дає змогу моделі враховувати контекст: при обробці кожного слова (токена) модель обчислює ваги уваги до інших слів у ввідній послідовності, визначаючи, які саме фрагменти тексту є найрелевантнішими для поточного прогнозу. Завдяки цьому модель будує внутрішні контекстуалізовані подання: вектори, які кодують значення кожного токена з урахуванням оточення [1].

Перед тим як застосувати механізм уваги, текст запиту проходить через етап токенізації. Модель розбиває вхідне речення на дрібніші фрагменти – токени, які можуть бути словами, частинами слів або символами. Як правило, використовуються алгоритми підготовки словника

на основі підслів, зокрема метод byte-pair encoding. Такий підхід дозволяє кодувати текст різними мовами і справлятися з неологізмами: будь-яке слово розбивається на знайомі підчастини, тож модель не залишає жоден фрагмент запиту невідомим. Кожен токен зі словника має навчене векторне представлення, яке вводиться на вхід трансформера. Далі за допомогою багатошарової самоуваги модель обчислює нові представлення, насичені контекстом сусідніх токенів, і зрештою генерує відповідь, декодуючи ці представлення назад у слова.

Важливою особливістю LLM є обмежена довжина контексту – так зване контекстне вікно моделі. Це максимальна кількість токенів включно із токенами запиту та згенерованої відповіді, яку модель може опрацювати за один раз. Історично контекстне вікно Generative Pre-Trained Transformer (GPT) поступово зростало: перші трансформери типово мали ліміт ~512 токенів, GPT-2 збільшив його до 1024, GPT-3 – до 2048 токенів. Сучасні найбільші моделі мають ще ширший контекст; зокрема, GPT-4 здатний обробляти контекст до 32 768 токенів (близько 50 сторінок тексту). Обмеження довжини запиту означає, що надто великий обсяг тексту неможливо подати моделі одразу – зайві токени будуть відкинуті або ігноровані. Більше того, дослідження вказують, що продуктивність моделей починає помітно падати при дуже довгих запитах задовго до досягнення граничної довжини. Це пов’язано з явищем «розфокусування уваги»: коли prompt включає надто багато інформації, модель може розгубитися щодо того, що є головним, і втратити частину релевантного контексту. Тому ефективні запити мають балансувати між наданням достатнього контексту і зайвою довжиною, яка перевантажує модель [3-5].

Формулювання prompt’у істотно визначає релевантність, точність і стабільність відповіді LLM. Якщо запит чіткий, конкретний і містить всю необхідну інформацію, модель, як правило, генерує більш точні й доречні відповіді. Навпаки, неоднозначні інтерпретації призводять до невизначеності моделі: вона може домислювати деталі або розуміти

питання не так, як того хотів користувач. Наукові роботи зазначають, що двозначні чи оманливі запити підвищують ризик упереджених або неправильних результатів [2].

З іншого боку, спеціальні техніки формулювання запиту допомагають підвищити якість відповіді. Зокрема, було показано, що коли модель отримує в prompt'і приклади розв'язання задач або додаткові підказки до того, як дати остаточну відповідь, результат значно поліпшується [6].

За останні роки виробилося кілька стратегій, як саме подавати завдання моделі у prompt'і, щоб досягти оптимального результату [2]:

– zero-shot prompting (нульовий приклад);

– few-shot prompting (з кількома прикладами);

– chain-of-thought prompting (ланцюжок міркувань);

– role prompting (задання ролі).

Zero-shot prompting – простий запит-інструкція без жодних демонстрацій. Модель отримує лише формулювання задачі або питання і повинна на основі своїх знань згенерувати відповідь. Zero-shot режим зручний для простих запитів, але на складних завданнях може давати субоптимальні результати, оскільки модель не завжди розуміє, який формат чи підхід очікується.

Few-shot prompting – техніка, коли до інструкції додають невеликі приклади виконання подібного завдання [2]. Зазвичай у prompt включаються одна або кілька пар «вхід – очікуваний вихід» як зразок. Цей метод, вперше масштабно продемонстрований у GPT-3 [6], суттєво покращує результати на нових задачах: модель навчається з контексту розуміти формат відповіді і підхід до розв'язання. Дослідження показали, що few-shot приклади особливо корисні для складних або вузькопредметних завдань [2] – вони дозволяють моделі активувати релевантні патерни знань.

Chain-of-thought prompting – метод, що спонукає модель явно розписувати кроки міркування перед остаточною відповіддю. Замість того щоб одразу дати результат, модель спочатку генерує послідовність

проміжних логічних кроків або аргументів так званий ланцюжок думок, а вже потім – висновок. Експерименти з LLM показали, що Chain-of-thought значно підвищує точність на арифметичних і логічних задачах – модель менше помиляється, якщо думає вголос [6].

Role prompting – підхід, коли в prompt явно задається певна роль або стиль, від імені якого модель повинна сформулювати відповідь [2]. Ідея в тому, щоб надати моделі контекст щодо потрібного тону, фаху чи аудиторії відповіді. Такі настанови скеровують модель говорити голосом певного експерта чи персонажа, що часто підвищує релевантність і достовірність відповіді у заданій сфері. Role prompting фактично зменшує необхідність для моделі здогадуватися про контекст – він явно задається користувачем. Наукові огляди зазначають, що рольові інструкції є одним з базових інструментів prompt engineering, нарівні з few-shot та chain-of-thought [2].

Перелічені методи можна поєднувати. Наприклад, користувач може задати моделі роль і навести кілька прикладів запитань-відповідей юридичного характеру, а також попросити обґрунтувати відповідь крок за кроком. В сучасних дослідженнях з'являються й інші варіації, наприклад, Tree-of-thoughts (дерево можливих кроків міркувань) чи автоматичний підбір оптимальних prompt'ів, але всі вони базуються на згаданих фундаментальних підходах [2].

Незважаючи на успіхи підходів вище, prompt engineering залишається творчим процесом, пов'язаним із низкою труднощів. Користувач може ненавмисно сформулювати запит так, що він містить неясності або двозначності. Модель не має справжнього розуміння намірів і може інтерпретувати неоднозначний prompt не тим чином, як очікувалося. Неоднозначні або неконкретні prompt'и підвищують ризик отримати нерелевантну чи помилкову відповідь [2]. Інша схожа проблема – контекстні пастки, коли користувач пропускає в запиті важливий контекст чи припускається фактичної помилки. Модель покірно використовує даний їй текст і може розвинути цю помилку далі. Таким чином, якість prompt'у

безпосередньо впливає на якість відповіді, і одне з ключових правил – робити запити максимально однозначними, точними та самодостатніми.

При багатокроковій взаємодії, коли користувач веде діалог з LLM, задаючи серію запитань виникає проблема підтримання контексту. Модель пам'ятає лише те, що вміщується в її контекстне вікно – решта історії відсікається. Якщо на початку розмови були задані важливі деталі, але потім діалог триває і перевищує ліміт пам'яті, модель може забути ключову інформацію, надану раніше. Це призводить до того, що відповіді на пізніші запити можуть втрачати зв'язок з попередніми уточненнями. Наприклад, обговорюючи технічний проект через 20 повідомлень, модель може не згадати початкові вимоги, якщо вони вже випали з контексту. Тому розробники часто змушені штучно рекапітулювати контекст у нових запитах або використовувати зовнішнє збереження стану діалогу. Оптимізація цього аспекту – активна тема досліджень, зокрема розробка моделей із розширеною пам'яттю та методів, що дозволяють динамічно вибирати, яку частину контексту зберегти в prompt.

Один з найбільш відомих викликів – це тенденція LLM вигадувати неправдиві або не підтверджені факти, коли вони не впевнені у відповіді [7]. Такі помилки дістали назву «галюцинації». В контексті формування запитів це означає, що модель може заповнювати прогалини у prompt'і власними здогадками. Якщо у запиті бракує потрібних даних, або питання виходить за межі знань моделі, LLM все одно згенерує відповідь – але вона може не мати опори в реальності. Наприклад, якщо спитати про біографію маловідомої особи, модель може вигадати правдоподібні, але хибні факти замість того, щоб зізнатися в незнанні. Частково зменшити галюцинації можна за рахунок більш релевантних prompt'ів: надаючи моделі контекст або джерела знань у запиті, ми зменшуємо її схильність вигадувати [7]. Також важливо формулювати питання так, щоб вони не заохочували модель до спекуляцій. Проте повністю позбутися цього явища лише інженерією prompt'у важко – додатково потрібні вдосконалені моделі чи інтеграція

перевірки фактів.

Як зазначалося, надто довгі або складні запити можуть негативно вплинути на модель. Перевантаження prompt'у зайвою інформацією ускладнює моделю виявлення суті питання [5]. Модель може розпорошити увагу на другорядні деталі і дати менш чітку відповідь. Більше того, великий prompt наближає до межі контекстного вікна, що не тільки технічно обмежує довжину відповіді, а й може призвести до часткової втрати раніше введеної інформації. Наприклад, коли користувач намагається «втиснути» в один запит декілька завдань чи дуже детальні дані, є ризик, що модель відповість лише на частину з них або зосередиться не на тому. З цієї причини рекомендується давати структуровані та лаконічні prompt'и: розбивати складне завдання на кілька послідовних запитів або видаляти з prompt другорядні фрагменти, які можна опустити. У наукових експериментах було виявлено, що продуктивність моделей знижується вже при довжині вводу понад кількасот токенів, навіть якщо формально це ще вписується в межі контексту [4].

Процес підготовки prompt'у відіграє критичну роль в архітектурі сучасних інформаційних систем на основі LLM. Зокрема, технологія Retrieval-Augmented Generation (RAG) передбачає, що перед подачею запиту в модель відбувається етап пошуку зовнішньої інформації, релевантної запиту [7].

Формування prompt'у у RAG включає злиття двох компонентів: запиту користувача та знайдених даних з бази знань. По суті, система автоматично розширює оригінальний запит, додаючи до нього контекст, – наприклад, найсвіжіші новини, статті або записки, що стосуються питання [7]. Потім об'єднаний prompt надходить до мовної моделі, і модель генерує відповідь, спираючись як на своє треноване розуміння мови, так і на надані факти. Таким чином, prompt стає точкою інтеграції між нейронною моделлю та традиційними базами знань. Якість сформованого prompt'у особливо важлива: якщо додати нерелевантний або занадто об'ємний текст

із бази даних (БД), це може заплутати модель. Навпаки, влучно підібраний фактичний контекст у prompt’і різко знижує ризик галюцинацій і підвищує достовірність відповіді [7]. RAG-підхід вже успішно застосовується для створення чат-ботів, що надають користувачам відповіді з посиланням на джерела: модель генерує текст, а поруч виводяться документи, з яких було взято інформацію. Це підвищує довіру до системи, адже користувач може перевірити першоджерела.

Методи формулювання запитів до LLM становлять ключовий елемент ефективної взаємодії з LLM, оскільки саме спосіб подання завдання визначає точність, релевантність і стабільність відповіді. У науковій літературі виокремлюють низку стратегій prompt engineering, що дозволяють керувати поведінкою моделі, зменшувати неоднозначність інтерпретації та підвищувати інформативність вихідних даних. Крім того, сучасні системи все частіше інтегрують зовнішні джерела знань (наприклад, у RAG-архітектурах), що розширює функціональні можливості prompt’ів і знижує ризики галюцинацій. Порівняльну характеристику основних методів промптингу, яка систематизує їхні характеристики, сильні сторони та обмеження наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика основних методів промптингу

Метод	Сутність	Переваги	Недоліки
Zero-shot prompting	Метод формує запит без прикладів; спирається на загальні знання та індуктивні патерни, активовані лише формулюванням інструкції. Структура: лаконічна інструкція або питання.	Лаконічність і відсутність додаткових прикладів зменшують когнітивне й контекстне навантаження; метод зберігає релевантність у стандартних задачах завдяки опорі на універсальні патерни.	Обмежена структура prompt'у не актуалізує специфічні патерни міркування; зростає ризик некоректного трактування формату або підходу, особливо в складних і вузьких задачах.
Few-shot prompting	У структуру запиту додаються зразки виконання схожих завдань, які задають очікуваний стиль, логіку та формат відповіді.	Приклади задають явний шаблон міркування, що спрямовує метод використовувати патерни з вищою релевантністю у складних завданнях.	Великий обсяг прикладів збільшує витрати контексту; якість відповіді стає залежною від точності й доречності наданих зразків; можливе перенесення артефактів зі зразків.

Продовження таблиці 1.1

Метод	Сутність	Переваги	Недоліки
Chain-of-thought prompting	Метод навмисно активує покрокове пояснення, фіксуючи проміжні міркування як частину структури відповіді.	Примусове розгортання кроків міркування, що покращує прозорість процесу та релевантність у логічних багатокрокових завданнях; полегшує виявлення помилок у ході думки.	Розширена форма відповіді збільшує витрати контексту; детальність не усуває ризику хибних висновків, а лише робить їх довшими.
Role prompting	У prompt'і окреслюється роль, тон і цільова аудиторія, що задає стиль міркування та рівень деталізації.	Рольова рамка спрямовує метод на добір релевантних патернів мовлення й структур пояснення; підвищується адаптованість відповіді до визначеної аудиторії.	Нечітко окреслена роль створює надмір свободи інтерпретації й може змінити очікувану глибину або стиль відповіді; спосіб подання може підсилювати приховані упередження.

Кінець таблиці 1.1

Метод	Сутність	Переваги	Недоліки
RAG-орієнтоване формування prompt’у	Метод поєднує користувацький запит із релевантними фрагментами з векторної бази знань; джерела явно інтегруються у структуру prompt’у.	Вставлені фрагменти зменшують ймовірність галюцинацій, адже метод спирається на актуальний корпус; прозорість джерел підвищує верифікованість і релевантність відповіді у складних інформаційних задачах.	Релевантність залежить від якості відбору фрагментів; можливе перенасичення контексту або введення другорядної інформації; збільшується складність побудови системи.

1.2 Аналіз методів векторного пошуку та формування контексту до LLM

З розвитком сучасних моделей штучного інтелекту та стрімким зростанням обсягів неструктурованих даних все більшої ваги набувають векторні БД – спеціалізовані системи управління базами даних (СУБД), що зберігають інформацію у вигляді високорозмірних числових векторів [8]. Кожен такий вектор є семантичним представленням певного об’єкта: тексту, зображення, аудіо тощо, отриманим шляхом перетворення сирих даних за допомогою моделі або алгоритму, який виконує відображення у векторне представлення (пер. з англ. embedding) [8-9]. На відміну від традиційних реляційних чи документо-орієнтованих СУБД, що оперують чітко структурованими даними та точним співставленням значень, векторні БД призначені для роботи з неструктурованими даними і забезпечують швидкий пошук за змістовною подібністю. Замість точного порівняння рядків чи полів, пошук здійснюється шляхом обчислення відстаней або міри схожості між векторами у багатовимірному просторі [8, 10]. Такий семантичний пошук дозволяє знаходити інформацію, релевантну запиту за змістом, навіть якщо вона не містить прямих збігів за ключовими словами. Векторна БД слугує пам’яттю для семантичних ознак: вона зберігає вектори, що кодують значення та контекст об’єктів, і дозволяє виконувати операції порівняння цих векторних представлень.

Embedding – це числовий вектор, компоненти якого містять закодовану інформацію про зміст і контекст текстового фрагмента. Мета такого відображення – розмістити семантично подібні тексти близько один до одного у векторному просторі. Для побудови embedding-векторів використовуються сучасні моделі глибокого навчання. Наприклад, модель Bidirectional Encoder Representations from Transformers (BERT) генерує контекстуальні представлення слів і речень, враховуючи оточення кожного

слова в тексті [9].

На основі BERT було створено спеціалізовані моделі для sentence-embedding – зокрема архітектура Sentence-BERT (SBERT), що поєднує дві копії мережі BERT у сіамську мережу. SBERT перетворює ціле речення у вектор розміру, наприклад, 768 чи 384, який є змістовним представленням цього речення; при цьому мережу навчено так, щоб близькі за змістом речення мали вектори, близькі за косинусною мірою [11]. Завдяки цьому пошук найбільш схожих речень у колекції з десятків тисяч документів, який за допомогою вихідного BERT вимагав би величезних обчислювальних затрат, із SBERT виконується за частки секунди [11].

Окрім моделей на основі BERT, активно застосовуються інші трансформерні моделі Sentence Transformers, наприклад, Family of Models від HuggingFace або MiniLM, які теж генерують sentence- embedding. OpenAI пропонує комерційні моделі для embedding, наприклад, text-embedding-ada-002, навчені на величезних обсягах різноманітного тексту; такі вектори добре відображають семантичні зв’язки і стали де-факто стандартом для багатьох застосувань, зокрема у системах retrieval-augmented generation. Незалежно від конкретної моделі, результатом є векторне представлення, що стискає зміст тексту у наборі чисел. Головна перевага полягає в тому, що ці вектори можна порівнювати між собою математично, знаходячи семантично близькі тексти через метрики відстані.

Щоб визначити, наскільки два embedding-вектори схожі, використовують кілька поширених метрик: евклідова відстань, косинусна подібність та скалярний добуток.

Евклідова відстань – це геометрична відстань між точками в просторі, що враховує різницю між значеннями всіх координат; вона залежить і від напрямку, і від величини векторів [12].

Для двох векторів евклідова відстань визначається за формулою:

$$d(\vec{a}, \vec{b}) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ \dots \ + (a_n - b_n)^2},$$

де $d(\vec{a}, \vec{b})$ – евклідова відстань між векторами $\vec{a}$ та $\vec{b}$.

a_i – i-та компонента вектора $\vec{a}$;

b_i – i-та компонента вектора $\vec{b}$;

n – розмірність векторного простору.

Косинусна подібність вимірює кут між векторами і визначається як їх скалярний добуток, поділений на добуток довжин (норм) цих векторів [12]:

$$\cos(\vec{a}, \vec{b}) = \frac{\vec{a} * \vec{b}}{\|\vec{a}\| * \|\vec{b}\|}, \quad (1)$$

де $\cos(\vec{a}, \vec{b})$ – косинусна міра подібності між векторами;

$a_i * b_i$ – скалярний добуток векторів $\vec{a}$ та $\vec{b}$;

$\|\vec{a}\|$ – довжина (норма) вектора $\vec{a}$;

$\|\vec{b}\|$ – довжина (норма) вектора $\vec{b}$.

Значення косинусної подібності лежить у діапазоні від -1 до 1: значення 1 відповідає нульовому куту (вектори співнаправлені і максимально схожі за змістом), 0 – вектори ортогональні (семантично не пов’язані), -1 – вектори протилежно спрямовані [12].

Важливо, що косинусна метрика звертає увагу лише на напрям векторів, ігноруючи їх довжину. Це зручно, оскільки при нормуванні всіх embedding-векторів до однакової довжини результуюче значення подібності залежить тільки від семантичного вмісту, а не від масштабів чисел.

Скалярний добуток двох векторів – це сума попарних добутків їх компонент. Він пропорційний косинусній подібності, але також враховує довжини векторів [12]:

$$\vec{a} * \vec{b} = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n,$$

де $\vec{a} * \vec{b}$ – скалярний добуток двох векторів $\vec{a}$ та $\vec{b}$;

a_i – i-та компонента вектора $\vec{a}$;

b_i – i-та компонента вектора $\vec{b}$;

n – розмірність векторного простору.

Якщо вектори нормалізовано (довжина кожного дорівнює 1), скалярний добуток чисельно співпадає з косинусною мірою.

Формула для визначення довжини (норми) вектора:

$$\|\vec{a}\| = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2},$$

де $\|\vec{a}\|$ – довжина (норма) вектора $\vec{a}$;

a_i – i-та компонента вектора $\vec{a}$;

n – кількість координат (розмірність простору).

У практичних реалізаціях вибір метрики часто залежить від того, з якою функцією втрат навчалася модель, що породжує embeddings. Як правило, для моделей, оптимізованих під косинусну подібність, такий самий критерій застосовують і при пошуку (альтернативно – виконують нормалізацію векторів і використовують скалярний добуток, що еквівалентно) [12].

Аналогічно, якщо модель навчалася максимізувати скалярний добуток між схожими прикладами або мінімізувати евклідову відстань, то доцільно використовувати ту саму міру і на етапі пошуку. Загалом, евклідова відстань інтерпретується як абсолютна міра різниці, корисна, коли важлива величина ознак, косинусна подібність – як міра збіжності напрямків фокусується на самих патернах ознак, ігноруючи їх масштаб, а скалярний добуток поєднує обидва аспекти.

Для завдань семантичного пошуку тексту найпоширенішою є косинусна подібність, оскільки вона добре відображає схожість тематики чи сенсу двох текстів, не зважаючи на їхню довжину [12].

Початкову основу для sparse-пошуку становить векторна модель документів з вагуванням термів за схемою Term Frequency-Inverse Document Frequency (TF-IDF), вперше системно проаналізована в роботах Дж. Сантана [13].

Нехай корпус містить N документів, документ d представлено як багатошарову множину термів, а $f_{(t,d)}$ позначає кількість входжень терма t у документі d. Одна з класичних формул TF-IDF має вигляд

$$tfidf(t,d) = tf(t,d) * idf(t),$$

де $tfidf(t,d)$– зважена оцінка важливості терміна tу документі d;

$tf(t,d)$ – частотність терміна tу документі;

$idf(t)$ – міра унікальності терміна в корпусі документів;

t – термін, для якого виконується оцінка;

d – документ, у якому розглядається термін.

Частотність терміна визначає, скільки разів термін зустрічається в документі:

$$tf(t,d) = f_{(t,d)},$$

де $tf(t,d)$– частотність терміна tу документі d;

$f_{t,d}$ – кількість входжень терміна tу документ d;

t – термін у тексті;

d – документ, для якого обчислюється частотність.

Зворотна частотність документа (Inverse Document Frequency) показує, наскільки рідкісним є термін tу всьому корпусі:

$$idf(t) = log\frac{N}{n_t},$$

де $idf(t)$– міра рідкісності терміна t;

N – загальна кількість документів у корпусі;

n_t – кількість документів, що містять термін t;

t – термін, для якого оцінюється рідкісність.

Така вага зростає зі збільшенням частоти терма в документі та зменшується, якщо терм часто зустрічається в корпусі, що забезпечує підсилення дискримінативних термів і придушення стоп-слів. У векторній моделі схожість між запитом q та документом d часто визначається за косинусною мірою.

Подібна схема вагування забезпечує високу лексичну точність: документи, що містять точні збіги ключових термів із запитом, отримують високі бали, а індексування через інвертовані списки дає змогу досягати високої швидкості пошуку навіть у масштабних колекціях.

Подальший розвиток probabilistic relevance framework привів до появи ранжувальної функції Best Match 25 (BM25), яка на сьогодні є стандартом де-факто для sparse-пошуку в інформаційно-пошукових системах. Робертсон і Сарагоса виводять BM25 як аппроксимацію до двопуассонівської моделі терм-частот із урахуванням насичення частоти терма та нормалізації за довжиною документа [14].

У класичній формі оцінка релевантності документа d відносно запиту q записується як:

$$BM25(q,d) = \sum_{t \in q} idf_{BM25(t)} \frac{f_{t,d}(k_1 + 1)}{f_{t,d} + k_1(1 - b + b\frac{|d|}{\langle d \rangle})}$$

де $BM25(q,d)$ – оцінка релевантності документа d запиту q;

q – текст запиту;

d – документ, для якого обчислюється оцінка релевантності;

t – термін, що входить до складу запиту q;

$idf_{BM25}(t)$ – зворотна частотність документа для терміна t;

$f_{t,d}$ – частота входження терміна tу документ d;

k_1 – параметр BM25, що визначає ступінь згладжування частоти;

b – параметр BM25, що контролює вплив довжини документа;

$|d|$ – довжина документа d або кількість термінів документа d;

$\langle d \rangle$ – середня довжина документа в колекції.

Формула, що реалізує модифіковану оцінку інверсної документної частоти, більш стійку до крайніх випадків:

$$idf_{BM25(t)} = log\frac{N - n_t + 0.5}{n_t + 0.5}$$

де $idf_{BM25}(t)$– зворотна частотність документа для терміна t;

N – загальна кількість документів у колекції, од.;

n_t – кількість документів, що містять термін t, од.

Нелінійний компонент:

$$\frac{f_{t,d}(k_1 + 1)}{f_{t,d} + k_1(1 - b + b\frac{|d|}{\langle d \rangle})}$$

де $f_{t,d}$ – частота входження терміна t у документ d;

d – документ, для якого обчислюється оцінка релевантності;

k_1 – параметр BM25, що визначає ступінь згладжування частоти;

b – параметр BM25, що контролює вплив довжини документа;

$|d|$ – довжина документа d або кількість термінів документа d;

$\langle d \rangle$ – середня довжина документа в колекції.

Нелінійний компонент моделює насичення: додаткові входження терма після певного порогу дають дедалі менший приріст ваги, а параметр b контролює ступінь нормалізації на довжину документа. Така конструкція

дозволяє зберігати високу швидкість пошуку завдяки інвертованому індексу, забезпечує стійкість до варіацій довжини документів і демонструє стабільні результати на класичних колекціях Text Retrieval Conference, де BM25 досі слугує сильним базовим підходом, з яким порівнюють сучасні нейронні моделі [15].

Sparse-методи характеризуються трьома ключовими властивостями. По-перше, висока швидкість досягається завдяки компактному інвертованому індексу, де складність першого етапу пошуку наближено оцінюється як

$$\mathcal{O}(|q|\ log\ N)$$

де $\mathcal{O}$ – асимптотична складність;

$|q|$ – довжина запиту;

N – кількість елементів у колекції.

при ефективній організації списків. По-друге, лексична точність забезпечує чутливість до точних входжень ключових термів, що особливо важливо для доменів із стабільною термінологією. По-третє, обмежене розуміння семантики зумовлене припущенням «bag-of-words» та відсутністю моделювання порядку слів і контексту; документи, які формулюють ту саму інформацію за допомогою синонімів або перефразувань, можуть отримувати низькі оцінки, що прямо відображається на якості відповідей LLM у RAG-сценаріях.

Розвиток нейронного інформаційного пошуку продемонстрував, що семантичні щільні представлення (dense embeddings) значно підвищують якість на задачах запитання-відповідь і passage retrieval, проте не витісняють повністю лексичні сигнали. У роботах Ліна та Ліна запропоновано dense-фреймворк для одночасного врахування лексичного та семантичного збігу, де комбінуються високорозмірні sparse-вектори та компактні dense-вектори у єдину схему оцінювання [16]. Гібридний пошук (hybrid search) у

загальному випадку можна формалізувати як:

$$s_{hyb}(q,d) = \lambda\, s_{sparse}(q,d) + (1-\lambda)\, s_{dense}(q,d)$$

де $s_{hyb}(q,d)$– гібридна оцінка релевантності документа d запиту q;

q – текст запиту;

d – документ, для якого обчислюється оцінка релевантності;

λ – ваговий коефіцієнт комбінування;

$s_{sparse}(q,d)$ – розріджена (BM25 або інша) оцінка релевантності;

$s_{dense}(q,d)$ – щільна (векторна) оцінка релевантності.

Деякі сучасні системи замість простої лінійної комбінації використовують більш складні агрегатори, наприклад, спільне навчання нейронних енкодерів, які оптимізують сумарну втрату за sparse- і dense-сигналами [17].

Таке комбінування дає змогу зберігати сильну чутливість до рідкісних термів, характерну для BM25, і одночасно покривати випадки семантичного розриву, коли формулювання запиту та документа істотно відрізняються. Експериментальні дослідження на колекціях ad-hoc-пошуку демонструють, що гібридні системи перевершують як чисто sparse-, так і чисто dense-ретрівери за показниками nDCG@10 та Mean Reciprocal Rank (MRR@10), особливо в багатомовних і крос-доменних сценаріях [18]. Водночас гібридні методи мають підвищену складність конфігурації: потрібна узгоджена нормалізація різнорідних оцінок, налаштування параметра λ, вибір і оптимізація індексних структур (інвертований індекс для sparse-частини і наближений пошук найближчих сусідів для dense-частини), а також контроль латентності при об’єднанні кількох каналів пошуку.

У Retrieval-Augmented Generation процес пошуку завершується формуванням контексту, який передається на вхід LLM. Нехай $R(q)$ повертає впорядковану множину топ-k документів:

$$R(q) = \{d_1, d_2, \dots, d_k\},$$

де $R(q)$– множина документів, отриманих у відповідь на запит q;

q – текст запиту;

d_i – i-й документ у цій множині;

k – кількість документів.

Ця множина впорядковується за мірою релевантності:

$$s(d_1) \geq s(d_2) \geq \cdots \geq s(d_k),$$

де $s(d_i)$ – оцінка релевантності документа d_i;

d_i – i-й документ у ранжованому списку;

k – кількість документів у списку.

LLM має обмежене контекстне вікно розміром L токенів, тому дизайн механізмів формування контексту безпосередньо впливає на точність відповіді та обчислювальну вартість.

Найпростішою стратегією є пряма конкатенація топ-k результатів. У цьому випадку формується контекст

$$C_{concat}(q) = trunc_L(d_1 \oplus d_2 \oplus \dots \oplus d_k) \quad (2)$$

де $C_{concat}(q)$– конкатенований контекст для запиту q;

q – текст запиту;

$trunc_L$ – оператор обрізання до довжини L;

$concat$ – операція послідовного об’єднання документів;

d_i – i-й документ у множині;

k – кількість документів;

L – максимальна довжина сформованого контексту.

Такий підхід використано як базовий у початкових реалізаціях RAG-моделей, де на вхід генеративному трансформеру подавалися кілька

найрелевантніших пасажів з BM25- або dense-рітрівера [19]. Простота реалізації, відсутність додаткових моделей і мінімальна затримка роблять конкатенацію привабливою як стартове рішення. Водночас неоптимальне використання контекстного вікна проявляється у двох аспектах: по-перше, довгі документи з високим рейтингом можуть повністю заповнити вікно й витіснити інші релевантні фрагменти; по-друге, нерелевантні або слаборелевантні частини тексту потрапляють у контекст і створюють шум, що підвищує ризик галюцинацій моделі й погіршує точність відповіді. Сучасні оглядові роботи з RAG-систем демонструють, що навіть при незмінному рітрівері оптимізація стратегії відбору й організації контексту дає суттєве зростання точності на задачах відкритого запитання-відповіді.

Семантичне перевпорядкування (re-ranking) розв'язує частину зазначених проблем за рахунок додаткового етапу оцінювання топ-k документів більш потужною моделлю, зазвичай cross-encoderом на базі BERT-подібного трансформера. У типовій схемі sparse- або гібридний рітрівер формує кандидатний список $R(q)$, після чого для кожної пари (q, d_i) обчислюється семантична оцінка:

$$s_{ce}(q, d_i) = f_\theta([q, d_i])$$

де $s_{ce}(q, d_i)$– оцінка релевантності, отримана моделлю зі спільного кодування;

q– текст запиту;

d_i – i-й документ, для якого обчислюється оцінка релевантності;

f_θ – модель зі параметрами θ;

$[q, d_i]$ – об'єднане подання запиту q та документа d_i.

Документи перевпорядковуються за $s_{ce}(q, d_i)$, і вже перші $k' \leq k$ передаються в контекст LLM. Ногейра та Чо показали, що використання BERT-cross-encoder'а для перевпорядкування BM25-кандидатів на колекції Microsoft Machine Reading Comprehension дає відносний приріст MRR@10

приблизно на 27 % порівняно з найкращими на той момент підходами, що підкреслює ефективність re-ranking'у як другого етапу [15]. Подальші дослідження в галузі нейронного Information Retrieval (IR) підтвердили, що багатоступеневі архітектури «легкий рітрівер + важкий re-ranker» забезпечують оптимальний компроміс між латентністю й якістю, особливо коли re-ranker застосовується лише до десятків або сотень кандидатів [15].

Використання re-ranking'у в RAG-підсистемах формування контексту призводить до помітного покращення узгодженості між питанням і відібраними фрагментами. У термінах контексту можна вважати, що функція $R(q)$ замінюється композицією

$$R_{rerank}(q) = Top_{k'}\left(ReRank\big(R(q)\big)\right)$$

де $R_{rerank}(q)$– підмножина документів після повторного ранжування;

q – текст запиту;

$ReRank$ – процедура повторного ранжування;

$Top_{k'}$ – оператор вибірки k'елементів;

k' – кількість документів, що відбираються після повторного ранжування.

Важливим технологічним аспектом є зрозумілість причин за яких був обран фрагмент контекста. Непрозорі методи ранжування можуть знижувати довіру користувача до системи, за причини недостатьої аргументації відбору. Така проблема інтерпритованості звичайна для LLM моделей та re-rank`у на основі LLM [20].

Окремий клас методів формування контексту пов'язаний із компресією (context compression), метою якої є зменшення кількості токенів, що подаються на вхід LLM, при збереженні максимальної кількості корисної інформації. Логічним формалізмом служить оператор компресії, який будує коротший текст так, що зберігається інформація, релевантна до

запиту. Узагальнений контекст тоді набуває вигляду згідно з формулою (2).

У роботі Token Compression Retrieval-Augmented Large Language Model запропоновано дві комплементарні стратегії: абстрактивну «summarization compression» на базі T5-моделі, що генерує скорочені резюме документів, та семантичну компресію, яка вилучає слова з низьким впливом на семантичний зміст [21]. Експерименти показують, що узгоджена компресія дає змогу зменшити обсяг контексту приблизно на 65 % без погіршення, а подекуди й із невеликим покращенням точності відповіді, тоді як більш агресивне скорочення токенів (приблизно на 80 %) призводить до контрольованого, але помітного падіння показників точності. У роботі EXIT запропоновано контекст-орієнтовану екстрактивну компресію, де кожне речення з retrieved-документів класифікується як «залишити» або «видалити» з урахуванням глобального контексту документа і запиту, причому рішення приймаються паралельно [22]. Такий підхід демонструє, що правильне видалення нерелевантних речень здатне покращити і точність відповідей, і швидкодію системи порівняно з некорованим контекстом, оскільки LLM фокусується на більш щільній, інформативній підмножині даних.

Компресія контексту у практичних RAG-системах забезпечує економію токенів і, відповідно, зменшення вартості та латентності запитів до LLM, однак супроводжується неминучим ризиком втрати критичних доказів, особливо для багатокрокових запитів чи задач, що потребують строгих посилань на першоджерела. Оглядові роботи з RAG-reasoning підкреслюють, що компресійні механізми мають проєктуватися у зв’язці з рітрівером і генератором, інакше розрив між прихованими припущеннями різних компонентів може призводити до нових типів галюцинацій.

Таблиця 1.2 демонструє, що вибір конкретної комбінації методів пошуку та формування контексту має базуватися на компромісі між обчислювальною вартістю, вимогами до латентності та цільовими показниками якості відповіді LLM.

Таблиця 1.2 – Порівняльна характеристика основних методів пошуку

Назва методу	Сутність	Основні переваги	Основні недоліки
TF-IDF / BM25	Лексичний пошук на основі інвертованого індексу з оцінюванням релевантності через ваги термінів.	Низька обчислювальна вартість завдяки операціям зі списками постингів; інтерпретованість і стабільність результатів; висока ефективність для корпусів із чітко визначеною термінологією.	Лексична природа методу не дає змоги врахувати семантичні еквіваленти чи перефразування; релевантність різко знижується в міждоменних або контекстно складних запитах.
Гібридний пошук	Поєднання лексичних та семантичних сигналів, отриманих від sparse- і dense-ретриверів.	Синергія dense- та sparse-компонентів забезпечує стійкішу релевантність, особливо в складних запитах; комбінування різних сигналів збільшує покриття варіативних формулювань.	Вищі обчислювальні витрати, оскільки потрібні дві моделі та операції інтеграції; додаткові вимоги до пам’яті й калібрування ваг між сигналами.

Кінець таблиці 1.2

Назва методу	Сутність	Основні переваги	Основні недоліки
Конкатенація top-k	Передавання у моделний запит простого списку k відібраних фрагментів без подальшої обробки чи фільтрації.	Мінімальні обчислювальні витрати: метод передбачає лише вибір і послідовне з'єднання фрагментів.	Висока ймовірність шуму метода: серед поданих фрагментів можуть міститися відомості з низькою релевантністю, що призводить до витіснення важливої інформації.
Semantic re-ranking	Додаткове ранжування відібраних top-k фрагментів через cross-encoder, що оцінює семантичну відповідність запиту.	Cross-encoder забезпечує детальніше семантичне зіставлення, що підвищує якість добору фрагментів та, релевантність RAG-відповідей.	Висока обчислювальна вартість методу: кожна пара запита та фрагмента потребує повного проходження моделі; метод погано масштабується з ростом k.
Компресія контексту	Скорочення або агрегування відібраних фрагментів перед переданням у запит, з метою зменшення обсягу контексту.	Метод зменшує використання контекстного вікна: зниження кількості токенів без значної втрати; включає більше джерел у запит.	Додатковий етап обробки збільшує обчислювальні витрати; існує ризик втрати змістових деталей у разі надмірної компресії.

1.3 Аналіз існуючих методів удосконалення запитів

Функціонування систем на основі LLM у режимі доступу до зовнішньої документальної бази визначається двома фундаментальними обмеженнями: фіксованим розміром контекстного вікна моделі та, дуже великим обсягом корпусу документів. Для реалістичних корпоративних чи галузевих сховищ вимірюється сотнями тисяч або мільйонами фрагментів, тоді як контекстне вікно навіть для сучасних LLM не перевищує десятків тисяч токенів. У дослідженнях, присвячених порівнянню підходів довгого контексту і RAG, підкреслюється, що саме обмежене контекстне вікно створює обмеження для вбудовування зовнішньої інформації, а отже, робить якість формування запиту та структурування корпусу ключовими факторами продуктивності системи [23].

У загальному випадку запит користувача описується як текстова послідовність, а корпус документів представляється або у вигляді сирих текстів, або у вигляді множини фрагментів (чанків). Задача етапу пошуку в RAG-конвеєрі полягає у виборі підмножини фрагментів, що максимізує релевантність до запиту при обмеженні на сумарну довжину

$$\sum_{c \epsilon \mathcal{C}} \ell(c) \leq L_{ctx} - \ \ell(q) - \ell(prompt_{\text{системи}})$$

де c – фрагмент тексту;

$\mathcal{C}$ – множина всіх контекстних фрагментів, які планується включити в prompt;

$\ell(c)$ – довжина фрагмента c у токенах;

$\sum_{c \epsilon \mathcal{C}} \ell(c)$ – сумарна кількість токенів, що займають усі фрагменти контексту;

L_{ctx} – максимальна допустима довжина контекстної частини prompt’а,

визначена системними або модельними обмеженнями;

$\ell(q)$ – довжина запиту користувача q;

$\ell(prompt_{\text{системи}})$ – довжина системного prompt'а (наприклад, інструкцій або preamble).

Для сучасних векторних методів кожному запиту q та фрагменту c зіставляється векторна репрезентація, а релевантність оцінюється, косинусною подібністю згідно з формулою (1).

За прямого застосування векторного пошуку до всіх фрагментів C обчислювальна складність оцінки релевантності масштабується як

$$\mathcal{O}(|\mathcal{C}| * d) \approx \mathcal{O}(N * m * d)$$

де $\mathcal{O}$ – асимптотична складність;

$|\mathcal{C}|$ – кількість елементів у множині контекстів $\mathcal{C}$;

d – розмірність векторного подання;

N – кількість документів;

m – середня кількість токенів на документ.

Для великих $|D|$ таке рішення виявляється надто затратним, тому формування запитів до LLM в умовах великого корпусу та обмеженого вікна контексту неможливо розглядати ізольовано від структурування самого пошуку. У сучасній літературі виділяється низка спеціалізованих методів, які змінюють як сам запит, так і подання корпусу перед взаємодією з LLM [24].

Ієрархічні схеми векторного пошуку пропонують розділити задачу на два рівні: спочатку виконується відбір релевантних документів, а далі – більш тонкий пошук релевантних фрагментів всередині вже звуженої множини кандидатів. У роботах з Dense Hierarchical Retrieval та його гібридних модифікацій описано архітектуру, у якій застосовується окремий документний та пасажний ретривери, що спільно формують багаторівневий простір подання [25].

Формально кожний документ d_j перетворюється у вектор, використвоуючи функцію отримання векторного подання:

$$v_j^{doc} = g(d_j) \in \mathbb{R}^{d_{doc}}$$

де v_j^{doc}– векторне подання j-го документа;

g – функція отримання векторного подання;

d_j – j-й документ;

$\mathbb{R}^{d_{doc}}$ – простір векторів розмірності d_{doc};

d_{doc} – розмірність векторного подання документа.

На першому етапі запит кодується у вектор після чого знаходиться множина індексів документів з найбільшою подібністю

$$J_q = TopK_{k_{doc}} \left\{ s_{doc}(q, d_j) = \frac{\langle f_{doc}(q), v_j^{doc} \rangle}{\|f_{doc}(q)\| \left\| v_j^{doc} \right\|} \right\}$$

де J_q– множина документів, відібраних для запиту q;

$TopK_{k_{doc}}$ – оператор вибірки перших k_{doc}документів із найбільшою оцінкою;

$s_{doc}(q, d_j)$ – оцінка подібності між запитом qі документом d_j;

$f_{doc}(q)$ – векторне подання запиту;

v_j^{doc} – векторне подання j-го документа;

k_{doc} – кількість документів, що відбираються.

На другому етапі для кожного документа відібраного для запиту, документ розбивається на фрагменти, для яких обчислюються вектори. Далі проводиться ранжування фрагментів, а до контексту LLM потрапляють фрагменти з найбільшими оцінками.

Така двоетапна схема використовує макрорівневу семантику документа на першому кроці та мікрорівневу семантику фрагментів на

другому [25].

Обчислювальна вигода ієрархічного підходу проявляється у зміні асимптотики, для якої складність визначається як:

$$\mathcal{O}(N * d_{doc}) + \mathcal{O}(k_{doc} * m * d_{frag})$$

де $\mathcal{O}$– асимптотична складність;

N – кількість документів;

d_{doc} – розмірність векторного подання документа;

k_{doc} – кількість вибраних документів;

m – середня кількість фрагментів у документі;

d_{frag} – розмірність векторного подання фрагмента.

Емпіричні дослідження демонструють, що така стратифікація дає змогу одночасно скоротити час відповіді та підвищити Recall@k порівняно з суто пасажним ретривером, зокрема в задачах відкритого запитально-відповідного пошуку [25].

Водночас двоетапний характер пошуку породжує систематичний ризик втрати релевантних фрагментів. Якщо документний ретривер помилково відкидає документ, що містить критично важливий пасаж, то навіть ідеальний фрагментний ретривер не зможе його відновити, оскільки відповідний документ не потрапив до множини документів відяібраних для запиту q. У роботах з гібридного ієрархічного пошуку пропонується пом’якшувати цю проблему через комбінування щільних та лексичних ознак вже на документному рівні, наприклад шляхом лінійної або навчуваної комбінації BM25-оцінки та векторної подібності, що дозволяє краще узгоджувати випадки, де первинна семантична близькість не супроводжується лексичним перетином термінів.

Для формування запиту до LLM та вибору контексту такий ієрархічний підхід означає, що конструктор промпта оперує не одиничними фрагментами, а структурованими сутностями: списком документів-свідків

із відповідними фрагментами та, за потреби, їх метаданими. У результаті контекст передається моделі не як довільна конкатенація чанків, а як напівупорядкована множина, що покращує інтерпретованість відповіді й дозволяє виконувати додаткові операції, наприклад, агрегацію або фільтрацію за джерелами.

Інший напрямок оптимізації формування запитів до LLM в умовах великого корпусу пов'язаний із розширенням початкового запиту перед векторним пошуком. У сучасних роботах пропонується використовувати LLM як генератор додаткових текстових репрезентацій запиту – синонімів, перефразувань, потенційних відповідей або псевдодокументів, які потім слугують основою для побудови багатших векторів запиту [26].

Нехай початковий запит кодується у вектор. LLM, використовуючи внутрішні параметричні знання та заданий промпт, генерує множину текстів, що інтерпретуються як розширення запиту та задають відносну довіру до початкового формулювання та окремих розширень:

$$\tilde{f}(q) = \frac{a\, f(q) + \sum_{i=1}^{M} \beta_i f(e_i)}{a + \sum_{i=1}^{M} \beta_i}$$

де $\tilde{f}(q)$– скориговане векторне подання запиту q;

a – ваговий коефіцієнт базового подання запиту;

$f(q)$ – векторне подання запиту;

β_i – ваговий коефіцієнт для прикладу e_i;

$f(e_i)$ – векторне подання прикладу e_i;

M – кількість прикладів, що використовуються в корекції.

У щільних ретриверах подібні лінійні або ядрові комбінації векторів забезпечують більш плавне заповнення семантичного проміжку між запитом і документами, які використовують відмінну термінологію [26].

У класичних задачах інформаційного пошуку query expansion традиційно підтримувався через псевдорелевантний зворотний зв'язок або

орієнтацію на термінологію з початкової вибірки документів. Сучасні дослідження демонструють, що LLM-орієнтований підхід суттєво підвищує Recall@k і nDCG як для лексичних, так і для щільних ретриверів, оскільки модель генерує не лише словоформи, а й фрази, які відображають приховану інтенцію користувача [27]. Зокрема, у роботах, які досліджують багатотекстову інтеграцію розширень, показано, що використання декількох незалежних семплів LLM з подальшою агрегацією підвищує стійкість до генерацій нерелевантних відповідей і покращує якість пошуку як у доменних, так і в позадоменних наборах даних [27].

Разом з тим розширення запиту LLM-методами має низку суттєвих недоліків, релевантних саме сценаріям з обмеженим контекстним вікном. По-перше, кожен виклик LLM для генерації збільшує латентність системи, а при використанні складних промптів – і вартість обчислень. По-друге, розширений запит інколи приводить до введення нерелевантних термінів, що у векторному просторі проявляється як додатковий шум вектора запиту, зменшення кута між скоригованим векторним поданням та широкою множиною документів і, як наслідок, падіння точності при фіксованому k. У деяких дослідженнях зафіксовано, що надмірно агресивне розширення, зокрема через генерацію довгих псевдодокументів, покращує покриття, але погіршує ранжування верхньої частини списку [28].

У контексті формування запиту до LLM така стратегія використовується двома способами. У першому варіанті розширення застосовується лише на етапі побудови векторної репрезентації, а власне текст запиту, який бачить LLM-генератор, залишається компактним, що дозволяє зекономити токени у вікні контексту. У другому варіанті частина згенерованих розширень явно включається у промпт у вигляді альтернативних формулювань питання, що полегшує для моделі інтерпретацію інформаційної потреби, проте збільшує розмір запиту і зменшує доступний обсяг для релевантних фрагментів. Оптимальний баланс між цими двома режимами залежить від домену, розміру корпусу та

вартості обчислень.

Третім ключовим компонентом, що безпосередньо впливає на формування запитів і вибір контексту, є стратегія сегментації документів. У простих реалізаціях RAG документи розбиваються на фіксовані за довжиною шматки, що добре узгоджується з вимогами до довжини дліни контекста, але часто руйнує семантичну цілісність. Унаслідок цього фрагменти, релевантні одному логічному твердженню, можуть потрапляти до різних чанків, а LLM змушена відновлювати цілісну картину з частково несумісних фрагментів.

Adaptive chunking, або адаптивна семантична сегментація, пропонує будувати межі чанків не за фіксованим числом токенів, а з урахуванням семантичної зв'язності сусідніх речень чи абзаців. У низці робіт показано, що стратегія, яка враховує локальну семантичну подібність між сусідніми реченнями, істотно покращує як точність ретривера, так і кінцеві метрики запитально-відповідних систем [28].

Нехай документ представляється послідовністю речень, а кожному реченню відповідає вектор який належить до множини векторів БД. Визначається функція подібності, наприклад косинусна, а межа чанка встановлюється в позиції t, якщо

$$\sigma(u_t, u_{t+1}) < \theta$$

де $\sigma(u_t, u_{t+1})$– міра подібності між елементами u_tта u_{t+1};

u_t – поточний елемент послідовності;

u_{t+1} – наступний елемент послідовності;

t – індекс позиції елемента у послідовності;

θ – порогове значення подібності.

Таким чином кожен чанк містить послідовність речень, між якими зберігається висока внутрішня семантична когерентність:

У роботах, присвячених впливу документної сегментації на RAG,

демонструється, що навіть проста евристика на основі подібності сусідніх речень здатна підвищити F1 і EM на низці наборів даних, оскільки релевантна інформація рідше залишається між чанками [28].

У більш просунутих підходах адаптивне chunking поєднується з ієрархічною сегментацією та графовою кластеризацією. Прикладом є підхід, у якому спочатку нейронна модель сегментації визначає межі базових сегментів, а потім ці сегменти розглядаються як вершини графа. Ребро між вершинами i та j додається, якщо їх подібність перевищує адаптивний поріг

$$\tau = \mu + k\sigma$$

де τ– порогове значення;

μ – середнє значення вибірки;

k – масштабний коефіцієнт;

σ – стандартне відхилення.

Подальший пошук максимальних кліків і їх об’єднання у кластери дозволяє утворювати чанки, які одночасно зберігають семантичну єдність і послідовний порядок у документі [29].

Такий підхід створює низку переваг у контексті обмеженого L_{ctx}. По-перше, кожен чанк містить логічно завершений фрагмент знання, що зменшує потребу включати в промпт зайві сусідні частини для підтримки релівантності. По-друге, ієрархічна структура чанків (від англ. chunk) (наприклад, «розділ → підрозділ → абзац») дає можливість комбінувати adaptive chunking з ієрархічним векторним пошуком: спочатку обираються великі сегменти-кандидати, потім уточнення виконується на рівні менших підчанків, які краще вкладаються у контекстне вікно. Нарешті, семантично орієнтована сегментація знижує ризик змішування тем діалогу, коли один чанк містить фрагменти, релевантні різним запитам, що особливо критично для медичних та юридичних доменів з високими вимогами до точності відбору доказів [30].

Складність adaptive chunking полягає у значних обчислювальних витратах на попередню обробку корпусу та у неоднорідності розміру чанків. У випадку динамічного визначення меж окремі чанки можуть бути суттєво довшими за середнє значення, що ускладнює подальшу оптимізацію розміщення декількох фрагментів у межах дліни контексту. Для компенсації використовуються додаткові обмеження на максимальну довжину чанка або вторинні операції компресії [28].

Узгоджене застосування ієрархічного векторного пошуку, LLM-орієнтованого query expansion та adaptive chunking формує цілісний підхід до побудови запитів в умовах великого корпусу й обмеженого контекстного вікна. Ієрархічний пошук зменшує обсяг фрагментів, серед яких здійснюється відбір, і таким чином знижує обчислювальну складність без істотної втрати покриття. Query expansion з LLM розширює семантику запиту, підвищуючи ймовірність потрапляння релевантних документів до множини кандидатів. Adaptive chunking, у свою чергу, забезпечує подання документів у вигляді фрагментів, які краще відповідають природній структурі знань і більш ефективно використовують обмежений обсяг контексту.

У подальшому підрозділі доцільно узагальнити наведені методи у вигляді порівняльної таблиці, де для кожного з них буде вказано основний принцип дії, ключові параметри, типові умови застосування (розмір корпусу, вимоги до латентності, домен), а також характерні недоліки, зокрема ризик втрати релевантних фрагментів на документному етапі для ієрархічного пошуку, складність реалізації й неоднорідність розміру чанків для adaptive chunking. Узагальнені порівняльні характеристики розглянутих методів формування запитів у RAG-системах наведено у таблиці 1.3.

Таблиця 1.3 – Порівняльна характеристика методів формування запитів у RAG-системах

Назва методу	Сутність	Типові умови застосування в RAG та LLM-системах	Вплив методу на формування запитів та роботу контекстного вікна
Ієрархічний векторний пошук	Метод застосовує двоетапний ретривал: спочатку відбираються релевантні документи як цілісні одиниці, а далі працює з їхніми внутрішніми чанками, звужуючи простір пошуку до підкорпусу.	Метод особливо ефективний у дуже великих корпусах та в системах, де суворі вимоги до латентності не дозволяють виконувати повний dense-пошук. Він також добре працює з документами, що мають виразну ієрархічну структуру	Метод формує запит у два логічні контури: верхній рівень визначає корпус для подальшого аналізу, а нижній рівень самі фрагменти документів. Контекстне вікно LLM наповнюється лише структурно пов’язаними та взаємно узгодженими фрагментами. Нерелевантні документи метод усуває ще до етапу формування контексту, що робить контекст щільним та більш керованим.

Продовження таблиці 1.3

Назва методу	Сутність	Типові умови застосування в RAG та LLM-системах	Вплив методу на формування запитів та роботу контекстного вікна
Query expansion з використанням LLM	Метод утворює множину варіантів запиту, синонімічних формулювань чи псевдодокументів та об'єднує їх у агрегований вектор, який підсилює семантичне охоплення області пошуку.	Метод найчастіше використовують у доменах з підвищеною вимогою до повноти: у сферах із великою термінологічною варіативністю, зі сталими стилістичними відмінностями між джерелами або там, де формулювання запитів суттєво змінюється залежно від користувача.	У векторному режимі метод тримає текст запиту компактним: розширення відбувається у векторному просторі й не збільшує вартість токенів. У текстовому режимі метод частково вбудовує розширення в prompt, збільшуючи контекстний обсяг, але формує запит більш деталізованим та семантично збалансованим, що збільшує релевантність вибірки.

Кінець таблиці 1.3

Назва методу	Сутність	Типові умови застосування в RAG та LLM-системах	Вплив методу на формування запитів та роботу контекстного вікна
Adaptive chunking	Метод сегментує документи на чанки змінної довжини, використовуючи семантичну когерентність, структурні межі чи результати кластеризації, щоб кожен чанк репрезентував логічно завершену одиницю змісту.	Метод добре працює з корпусами, де внутрішня структура документів нерівномірна. Метод особливо ефективний у системах, де індексація виконується офлайн і дозволяє оптимізувати сегментацію наперед.	Метод наповнює контекстне вікно інформативними й семантично цілісними блоками. Через це контекст містить меншу кількість фрагментів, але кожен із них передає значно більше змісту.

1.4 Постановка задачі дослідження

Актуальність дослідження зумовлена тим, що сучасні великі мовні моделі, попри істотне зростання кількості параметрів і продемонстровані можливості few-shot та zero-shot навчання, залишаються жорстко обмеженими за розміром контекстного вікна та не мають вбудованих механізмів довготривалої пам’яті. Як показано у роботах, присвячених моделям типу GPT-3, масштабування параметричної частини моделі суттєво покращує узагальнювальну здатність, однак не усуває залежності якості відповіді від повноти й коректності сформульованого запиту та поданого контексту, особливо у знаннєво-інтенсивних задачах. За відсутності доступу до історії попередніх взаємодій із користувачем короткі або слабо структуровані запити призводять до поверхових, частково релевантних або галюцинаторних відповідей, оскільки модель оперує лише тією частиною інформації, яка поміщається в поточне вікно уваги.

У цьому контексті концепція RAG розглядається як формальний підхід до поєднання параметричної пам’яті мовної моделі з зовнішньою непараметричною пам’яттю, організованою у вигляді векторного індексу текстових фрагментів. Векторні бази даних у такій архітектурі виконують функцію семантичного сховища знань, де документи та фрагменти попередніх діалогів представлені у вигляді щільних embedding -векторів, що дає змогу здійснювати пошук за семантичною подібністю, а не лише за лексичним збігом. Порівняльні дослідження щільних методів пошуку демонструють їх принципову перевагу над класичними розрідженими моделями на основі TF-IDF та BM25 за показниками якості відбору релевантних контекстів для відкритих задач запитання–відповідь. При цьому методи TF-IDF та BM25, що історично є базовими для текстового пошуку, продовжують демонструвати обмежену спроможність до відображення семантичних еквівалентів, варіативності формулювань і

міждисциплінарних зв’язків.

У RAG-системах проблема релевантності загострюється тим, що якість відповіді великої мовної моделі жорстко детермінована точністю й структурованістю набору фрагментів, включених до промпту. Надмірний, частково нерелевантний або слабо структурований контекст призводить до перевантаження контекстного вікна другорядною інформацією й зменшує частку інформативних токенів, тоді як недостатній або занадто агресивно відфільтрований контекст не дозволяє активувати наявні в параметричній частині моделі доменні знання. Відсутність повноцінної довготривалої пам’яті, що системно враховує зміст попередніх діалогів з конкретним користувачем, додатково зумовлює втрату контекстно значущих нюансів предметної області, які можуть істотно змінювати інтерпретацію запиту. Унаслідок цього виникає завдання формалізації процесу формування запиту як керованого механізму адаптивної інтеграції релевантного векторного контексту, зокрема контексту з попередніх сесій, за умови мінімізації шуму, надлишковості та структурної неузгодженості включених фрагментів.

Об’єктом дослідження є процес формування запитів до великих мовних моделей, який розглядається як впорядкована послідовність операцій аналізу, нормалізації, трансформації та збагачення початкового тексту користувача з урахуванням доменної специфіки, поточної інформаційної потреби та накопиченої історії взаємодій. Предметом дослідження виступають методи побудови запитів із використанням векторного контексту, а саме принципи організації зовнішньої семантичної пам’яті на основі векторної бази даних, алгоритми векторного пошуку та ранжування, а також правила відбору, узгодження й структурованого включення обраних фрагментів у промпт до LLM за умов обмеженого розміру контекстного вікна. Об’єкт дослідження у формалізованому вигляді можна визначити як процес формування запитів до великих мовних моделей, а методи дослідження – як сукупність методів формування запитів з використанням векторної бази даних, що забезпечують автоматизовану

інтеграцію релевантного контексту різної природи в єдину запитну конструкцію. Метою роботи є розроблення формалізованого методу формування запитів до великих мовних моделей з автоматичною інтеграцією релевантного контексту з векторної бази даних, орієнтованого на підвищення точності, повноти, логічної узгодженості та доменної релевантності згенерованих відповідей порівняно з традиційними техніками prompt engineering і базовими RAG-реалізаціями.

Наукова новизна роботи полягає у теоретичному обґрунтуванні та побудові методу інтеграції результатів векторного пошуку, здійсненого за історією попередніх взаємодій користувача, у поточний запит до великої мовної моделі. На відміну від відомих рішень, у яких зовнішня пам’ять представлена статичним набором документів або слабко персоналізованою базою знань, запропонований підхід трактує багатосесійний діалог як динамічне векторне сховище персоналізованих знань і контекстів, що оновлюється в процесі взаємодії та використовується для побудови семантично насиченого промпту. Метод задається у вигляді формалізованого конвеєра, який включає побудову embedding-представлень поточного запиту та релевантних фрагментів попередніх бесід, виконання векторного пошуку з урахуванням доменної специфіки, адаптивне ранжування знайдених фрагментів за семантичною близькістю й інформативністю, відсікання шумових і надлишкових елементів та формування деталізованого промпту зі структурованим включенням обраного контексту. У межах такого підходу контекст, що передається моделі, постає не як набір випадково або вручну підібраних фрагментів, а як формально скоординована структура, оптимізована за критеріями релевантності.

Практичне значення одержаних результатів полягає у створенні прототипу інформаційної технології формування розширених запитів до великих мовних моделей, у якій модуль векторного пошуку та векторна база даних інтегровані як невід’ємні компоненти підсистеми побудови промптів.

Така технологія реалізує функції зовнішньої довготривалої семантичної пам’яті, забезпечує персоналізовану реконструкцію контексту на основі попередніх діалогів та дає змогу підвищити стабільність і прогнозованість поведінки LLM-орієнтованих систем у прикладних сценаріях – від інтелектуальних освітніх платформ і корпоративних асистентів знань до доменних діалогових агентів зі складною міждисциплінарною предметною областю. Сформульована таким чином науково-прикладна проблема відсутності формалізованого методу формування запитів, здатного одночасно адаптивно інтегрувати релевантний векторний контекст, у тому числі з попередніх діалогів, систематично усувати шумову та надлишкову інформацію і забезпечувати стабільно високий рівень доменної точності й логічної узгодженості відповідей, визначає актуальність проведеного дослідження.

Метою роботи є розробка метода формування запитів до LLM з автоматичною інтеграцією релевантного контексту з векторної бази для підвищення якості відповідей.

Для досягнення даної методи необхідно вирішити наступні питання:

– аналіз методів формування запитів та RAG-підходів;

– розробка методу інтеграції результату векторного пошуку з попередніх бесід у запит користувача до великої мовної моделі;

– розробка інформаційної технології формування запитів до великих мовних моделей;

– експериментальна перевірка розробленого методу.

2 ТЕОРЕТИЧНІ ОСНОВИ ТА РОЗРОБКА МЕТОДУ

2.1 Підхід до вирішення задачі

Для підвищення релевантності відповідей LLM пропонується динамічно доповнювати промпт актуальною інформацією з довготривалої пам’яті діалогу. Зовнішню пам’ять реалізовано у вигляді векторної БД, де зберігаються фрагменти історичних діалогів у вигляді векторних представлень. Такий підхід належить до класу методів RAG – генерації тексту на основі зовнішніх знань, що зберігаються у диференційовано доступній пам’яті [31, 32].

Під час нового запиту модель здійснює пошук у векторному просторі за семантичною схожістю та вибирає релевантні фрагменти діалогу із бази, додаючи їх до контексту запиту. За рахунок цього штучний співрозмовник отримує доступ до фактів і деталей, які виходять за межі короткострокової пам’яті, але були раніше згадані або збережені. В результаті покращується зв’язність та точність відповіді: модель може враховувати ключові відомості з історії спілкування, підтримувати тему, пам’ятати уподобання користувача [33] та уникати суперечностей, спричинених забуванням старої інформації [34]. Таким чином, поєднання LLM з векторною базою знань слугує основою пропонованого рішення.

Embedding становлять собою векторні репрезентації текстових фрагментів, у яких кожний вимір відображає певний аспект їхньої семантичної структури. На рисунку 1.1 подано типовий процес перетворення сирих текстів у такі представлення: кожен фрагмент незалежно подається на вхід embedding-моделі, яка відображає його у багатовимірний простір фіксованої розмірності. У цьому просторі близькість між векторами корелює зі смисловою подібністю відповідних текстів, а віддаленість сигналізує про їхню концептуальну різнорідність.

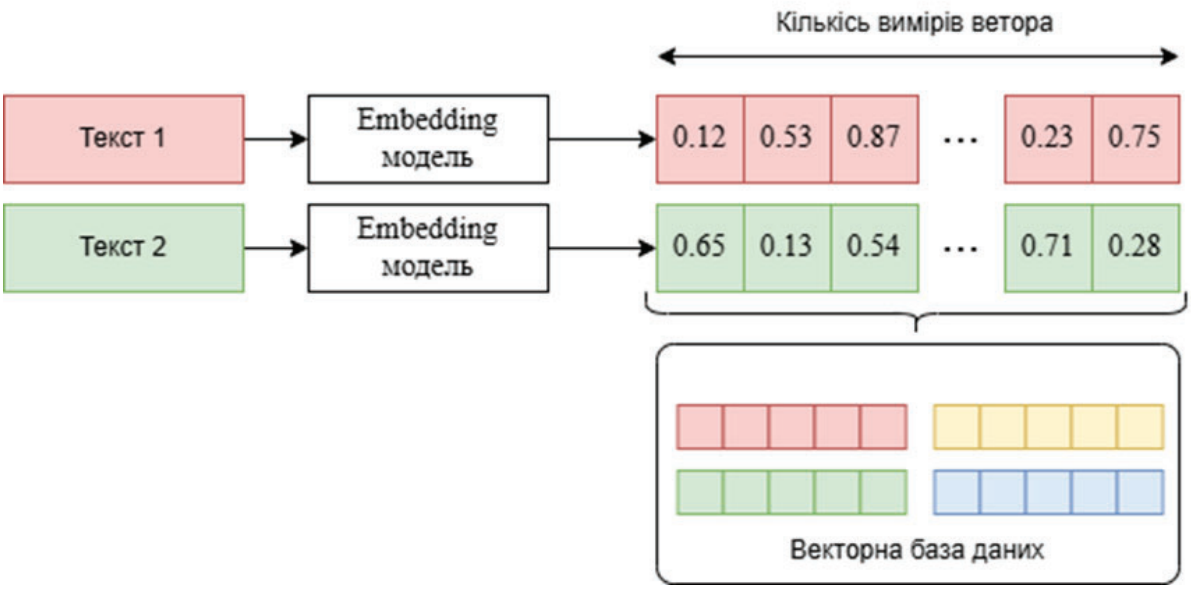

Рисунок 1.1 – Схема представлення текста у вигляді embbeding

Кожен елемент отриманого вектора є числовою ознакою, сформованою внаслідок багаторівневої трансформації тексту, що включає аналіз контексту, латентних залежностей та семантичних патернів. Таким чином, навіть тексти, які різняться лексично, але збігаються за змістом або мають спільний тематичний профіль, відображаються у векторні області, що розташовані поряд.

Після обчислення embedding система заносить їх до векторної бази даних. У цій базі кожний вектор функціонує як компактний і придатний до пошуку запис, що узагальнює семантичний зміст відповідного тексту. Це забезпечує можливість виконувати порівняння та відбір релевантних фрагментів не за поверхневими словесними ознаками, а за їхніми позиціями у високовимірному семантичному просторі. У RAG-системах така організація даних є критичною: вона дозволяє методу ефективно знаходити фрагменти, змістово наближені до запиту, навіть якщо формулювання користувача суттєво відрізняється від тих, що містяться у корпусі.

Саме завдяки embedding система отримує можливість здійснювати пошук за смислом, а не за збігами слів. Це дозволяє їй інтегрувати у контекст відповіді ті частини діалогу чи знань, що є релевантними з погляду

змісту, підтримуючи когерентність та тематичну узгодженість взаємодії зі співрозмовником.

Розроблений підхід має відповідати трьом ключовим вимогам: релевантності, масштабованості та адаптивності. Векторна БД повинна повертати саме ті фрагменти діалогу, що найбільш значущі для поточного запиту. Це вимагає використання якісної функції семантичного зіставлення та відсіювання нерелевантного контексту. Для забезпечення цього вводиться функція embedding, яка перетворює текстові дані у вектори фіксованої розмірності:

$$E: T \to \mathbb{R}^d \tag{3}$$

де E– функція, що відображає текст T у векторний простір $\mathbb{R}^d$;

T – текстовий об'єкт (рядок, документ тощо);

$\mathbb{R}^d$ – простір векторів розмірності d;

d – розмірність векторного подання.

Вектори зберігаються у БД, формуючи простір пам'яті. Векторна функція пошуку визначає підмножину найбільш близьких за змістом фрагментів з простору пам'яті на основі метрики подібності. Як метрику обрано косинусну схожість:

$$sim(v_q, v_i) = \frac{v_q * v_i}{\|v_q\| \, \|v_i\|},$$

де $sim(v_q, v_i)$– міра подібності між векторами v_qта v_i;

v_q – векторне подання запиту;

v_i – векторне подання елемента i;

$\|v_q\|$ – норма вектора запиту;

$\|v_i\|$ – норма вектора елемента.

Таким чином, релевантність досягається завдяки тому, що до

контексту включаються лише ті фрагменти, які максимально близькі до запиту у семантичному просторі [32]. Метод повинен ефективно працювати при великому обсязі накопичених діалогів. Застосування векторного індексу у базі дозволяє виконувати швидкий пошук навіть за мільйонами фрагментів. Кожен текстовий документ або діалог розбивається на сегменти фіксованої довжини, кожен сегмент векторизується і зберігається у базі [33]. Пошук у такій структурі виконується за сублінійний час, що забезпечує масштабованість рішення при зростанні даних.

Метод має динамічно оновлювати пам’ять і підлаштовуватися до нової інформації. Це означає, що після завершення діалогу система додає до бази стислий запис про нього для подальшого використання. Одночасно бажано реалізувати механізми оновлення та забування: менш корисні або застарілі спогади з часом повинні мати меншу вагу або видалятися, тоді як важлива інформація – підкріплюватися. Існують підходи, що використовують експоненційну функцію забування для поступового зменшення значущості старих спогадів і посилення часто запитуваної інформації [35]. Завдяки цьому реалізується людиноподібна пам’ять, здатна вибірково забувати другорядне і акцентувати ключові знання з урахуванням часу. Адаптивність також означає гнучкість щодо різних моделей. Система може працювати як з закритими інтерфейсами (Application Programming Interface, API) LLM, так і з відкритими LLM, інтегруючи їх через уніфікований інтерфейс [35].

Зауважимо, що запропонований підхід поєднує сильні сторони двох напрямів: векторного семантичного пошуку та автоматичного реферування тексту. Пошук у векторній базі відповідає за оперативне відтворення релевантного контексту на основі семантики запиту, а побудова текстового prompt’у з врахуванням знайдених даних дозволяє інтегрувати цей контекст у процес генерації відповіді. З іншого боку, після завершення діалогу алгоритм оновлює пам’ять через побудову узагальненого опису, що підтримує якість і актуальність знань у базі. Циклічне застосування цих

кроків забезпечує безперервне навчання системи: нові діалоги збагачують базу знань, а актуальні знання з бази підвищують якість наступних діалогів. Таким чином, дотримуються всі визначені вимоги: система надає релевантні відповіді за рахунок пошуку потрібних даних [32], зберігає продуктивність при зростанні обсягу пам’яті завдяки ефективним індексам [33], та адаптується до нової інформації шляхом динамічного резюмування і вибіркового забування [35].

Важливим аспектом є інтеграція знайденого контексту у текст запиту до моделі. Векторний пошук виступає фільтром, який зі всієї сукупності наявних даних обирає невеликий піднабір. Цей піднабір фактично репрезентує собою динамічно сформовану пам’ять для даного кроку діалогу. Далі функція формування prompt’у включає цю пам’ять у вхідне повідомлення. Можна сказати, що відображення задає композицію двох операторів – пошуку та побудови запиту, – яка перетворює початковий запит на розширений запит, що вже містить контекстні знання. Отже, результат пошуку у векторній базі безпосередньо впливає на поведінку мовної моделі: фрагменти, отримані з бази, подаються моделі у тій самій формі, що й звичайна історія діалогу, тому модель не відрізняє інформацію з пам’яті від наданої користувачем чи системою. Це дозволяє інтегрувати знання без змін у параметрах моделі – достатньо лише правильно сформувати prompt. З теоретичної точки зору, такий підхід еквівалентний використанню непараметричної пам’яті у алгоритмі роботи моделі. Параметри LLM зберігають імпліцитні знання, набуті при переднавчанні, тоді як зовнішня векторна пам’ять містить експліцитні знання у зручній для вибірки формі [31]. Комбінування цих двох джерел вхідних даних під час генерації дозволяє досягти кращої точності, прозорості та керованості: інформація в пам’яті може оновлюватися і перевірятися, а модель – оперувати більшим контекстом, ніж це визначено її фіксованим вікном. Отже, пошук у векторному просторі і побудова текстового prompt’у утворюють єдиний процес, що з’єднує блок IR з блоком Natural Language

Processing generation в рамках інтегрованого рішення. Практично це реалізується як частина програмного «ланцюжка» обробки запиту: після отримання питання користувача система викликає модуль пошуку у базі, а потім передає як prompt мовній моделі злитий текст знайденого контексту та самого запитання. Така архітектура відповідає сучасним підходам до розширення контексту LLM, зокрема, техніці RAG, де зовнішній пошуковий модуль поєднується з генеративною моделлю [32]. У результаті модель поводиться як двокомпонентна система: вона спочатку згадує потрібну інформацію з бази знань, і вже на основі цього формує підсумкову відповідь користувачу.

Підсумовуючи, запропонований метод можна подати як послідовність узгоджених етапів. Спочатку задається простір збережених знань, що складається з текстових фрагментів та їхніх векторних подань. Поточний діалог описується історією попередніх реплік, до якої додається новий запит користувача. Далі визначається процедура пошуку релевантного контексту, яка для поточного запиту обирає фрагменти, найбільш схожі на нього за мірою семантичної близькості. На основі знайдених індексів формується множина відповідних текстів, що слугують зовнішнім контекстом для моделі. Завершальним етапом є побудова розширеного запиту, у межах якого історія діалогу, відібраний контекст та поточне запитання об’єднуються в єдину текстову структуру, що передається мовній моделі для генерації остаточної відповіді. Функція формування розширеного запиту P об’єднує історію, знайдений контекст і актуальний запит у єдиний текст:

$$\tilde{q} = P(H_{n-1}, C(q), q)$$

де $\tilde{q}$– скоригований запит;

P – функція перетворення або уточнення запиту;

H_{n-1} – історія взаємодії до кроку $n-1$;

$C(q)$ – контекст, сформований на основі початкового запиту q;

q – початковий запит.

Після завершення діалогу (при отриманні спеціального сигналу завершення або неактивності користувача) застосовується функція підсумовування S:

$$m = S(H_n)$$

де H_n – повна історія діалогу;

S – функція підсумування;

m – згенерований машинний конспект.

Цей результат додається до бази тим самим оновлюючи довготривалу пам’ять системи.

2.2 Розробка удосконаленого методу формування запитів

Метою розробки запропонованого методу є подолання виявлених у розділі 1 недоліків, зокрема низької релевантності результатів та надлишкового або нерелевантного контексту у відповідях. Загальна ідея полягає в динамічному формуванні запиту шляхом поєднання семантичного векторного пошуку контексту з адаптивними шаблонами prompt’ів. Такий підхід дозволяє автоматично доповнювати початковий запит користувача найбільш доречними фрагментами інформації перед передачею його до мовної моделі, що підвищує точність і змістовність одержуваних відповідей [36, 37].

На першому етапі вхідні текстові дані проходять попередню обробку. Це включає очищення тексту від шумів (спеціальних символів, HyperText Markup Language (HTML) тегів тощо), нормалізацію (приведення до

однакового регістру, усунення зайвих пробілів), видалення стоп-слів (за потреби) та токенізацію. Попередня обробка забезпечує уніфікацію вхідного тексту і покращує подальше перетворення векторами. Результатом цього етапу є підготовлений текст запиту q_{clean}, а також корпус документів (база знань), розбитий на логічні фрагменти і очищений аналогічним чином. Належна сегментація і очищення документів дозволяє виділити самостійні контекстні фрагменти оптимальної довжини (зазвичай кілька сотень токенів) для ефективного пошуку [38].

На другому етапі кожен текстовий фрагмент перетворюється на вектор ознак фіксованої розмірності n за допомогою моделі embedding. Формально це можна подати як відображення згідно з формулою (3).

Для підготовленого запиту спершу визначається його векторне подання. Аналогічним чином для кожного документа бази знань заздалегідь сформовано набір векторів, які відповідають окремим фрагментам документації. Використання навченої модельної функції дає змогу відобразити семантичний зміст кожного текстового елемента у спільний векторний простір. Саме ці векторні подання забезпечують можливість семантичного пошуку, оскільки близькі між собою вектори відповідають фрагментам, що є подібними за змістом.

Третій етап полягає у знаходженні фрагментів бази знань, найбільш релевантних до запиту, на основі порівняння векторів. Для кожного вектора документа обчислюється міра схожості з вектором запиту. В якості такої міри використовується косинусна близькість між векторами згідно з формулою (1).

Косинусна схожість набуває значення від 0 до 1 (після нормалізації довжин), причому більш високі значення відповідають вищому семантичному збігу запиту з фрагментом. Далі виконується пошук k-найближчих сусідів – вибір тих фрагментів, чиї embedding-вектори розташовані найближче до вектора запиту за метрикою S. Алгоритм kNN є класичним підходом для вибірки схожих об’єктів за заданою метрикою

близькості, і в контексті embedding-пошуку він ефективно знаходить семантично релевантні документи за допомогою структури векторного індексу. В результаті цього етапу формується множина кандидатів на роль контексту запиту – наприклад, множина топ-k фрагментів ранжованих за спаданням міри релевантності

Четвертий етап передбачає фільтрацію отриманих кандидатів контексту за порогом схожості. Встановлюється мінімально допустиме значення τ. Усі фрагменти, для яких значення релевантності менше за τ , вилучаються зі списку кандидатів як нерелевантні. Така фільтрація запобігає включенню фрагментів, що випадково містять спільні з запитом слова, але не несуть корисної інформації за змістом. Іншими словами, поріг τ забезпечує базовий рівень релевантності: до наступного кроку потрапляють тільки фрагменти, схожість яких з запитом достатньо висока. Застосування порогового відсіву особливо важливе, оскільки простий вибір Top-K без порогу може призвести до потрапляння певної кількості нерелевантних даних; введення ж глобального порогу схожості (після відповідного масштабування оцінок) дає змогу відфільтрувати малоінформативні результати і підвищити точність вибірки. У формалізованому вигляді множину відібраних релевантних embedding-векторів можна подати як

$$R = \{e_d \in E_D \mid S(e_q, e_d) \geq \tau,$$

де R– множина відібраних елементів;

e_d – елемент із множини E_D;

E_D – множина всіх доступних елементів;

$S(e_q, e_d)$ – функція подібності між елементами e_qта e_d;

τ – порогове значення подібності.

Тобто R містить embeddings тих фрагментів, що мають косинусну схожість із запитом не меншу за поріг τ. Далі з цієї множини R обираються

k найбільш схожих фрагментів (якщо після фільтрації залишилося більше k кандидатів). Таким чином, отримується підсумковий набір контекстних фрагментів, які будуть використані для побудови запиту до мовної моделі.

На п’ятому етапі відбувається безпосереднє конструювання фінального запиту Q' для передавання в LLM. Для цього оригінальний запит користувача q доповнюється відібраними фрагментами контексту з множини C. Об’єднання здійснюється, наприклад, шляхом конкатенації тексту запиту з текстами відповідних фрагментів (з додаванням необхідних розділювачів або підказок щодо формату). Формально побудований запит можна подати так:

$$Q' = concat\left(q, TopK\left(E_D, e_q, k\right)\right),$$

де Q' – розширений запит;

$concat$ – операція конкатенації;

q – початковий запит;

$TopK(E_D, e_q, k)$ – вибір kелементів із множини E_D, найподібніших до e_q;

E_D – множина доступних елементів;

e_q – векторне подання запиту;

k – кількість елементів для включення.

Іншими словами, Q' складається з початкового запиту q і контексту, що містить k найбільш релевантних фрагментів інформації з бази. Структура prompt-шаблону при цьому є адаптивною: вона може змінюватися в залежності від кількості та змісту включених фрагментів. Зокрема, шаблон передбачає маркери або інструкції, що відділяють контекст від тексту запиту, а також можуть додаватися спеціальні вказівки моделі. Адаптивність шаблону означає, що якщо фрагментів декілька або вони різного типу, форматування запиту Q' підлаштовується відповідним чином для оптимальної взаємодії з LLM.

На завершальному етапі сформований розширений запит Q' передається до LLM. Модель обробляє запит, який вже містить додатковий контекст, і генерує відповідь A. Завдяки вбудованим у Q' зовнішнім відомостям, LLM може надати більш точну та обґрунтовану відповідь, оскільки вона бачить актуальну інформацію, якої могло бракувати у її параметричній пам’яті. Таким чином, вихід моделі є результатом динамічно побудованого запиту, що містить релевантні знання, і цей підхід відповідає парадигмі RAG.

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ФОРМУВАННЯ ЗАПИТІВ ДО ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

3.1 Опис інформаційної технології

Загальна концепція архітектури інформаційної технології полягає в тому, що перед генерацією відповіді мовною моделлю виконується пошук необхідних знань і підживлення моделі знайденими фактами, що підвищує якість та достовірність отриманих результатів.

Розглянута архітектура підтримує повний цикл обробки запиту: від формування користувацького запитання до отримання кінцевої відповіді. На рисунку 3.1 схематично зображено ключові компоненти та потоки даних у системі. Процес починається з того, що користувач вводить запит через інтерфейс. Цей запит спочатку обробляється й готується до пошуку. Далі нейронна модель для побудови embedding-представлень перетворює текст запиту на вектор, який компактно кодує його зміст у n-вимірному просторі ознак. Отриманий вектор запиту спрямовується до векторної БД, де виконується пошук найбільш подібних векторів із попередньо проіндексованого корпусу знань. На основі міри близькості знаходяться топ-k релевантних фрагментів інформації. Ці відібрані фрагменти потім додаються до початкового запиту при формуванні підсумкового prompt’у для LLM.

Іншими словами, модуль побудови запиту об’єднує текст запитання користувача з знайденими релевантними даними, а також може додати інструкції щодо формату відповіді. Таким чином, мовна модель одержує розширений prompt, що вже містить необхідні факти, і генерує фінальну відповідь, спираючись на наданий контекст. На завершальному етапі модуль аналізу результатів може перевірити відповідь та здійснює логування: зберігає пари запит-відповідь та використаний контекст для збору метрик якості.

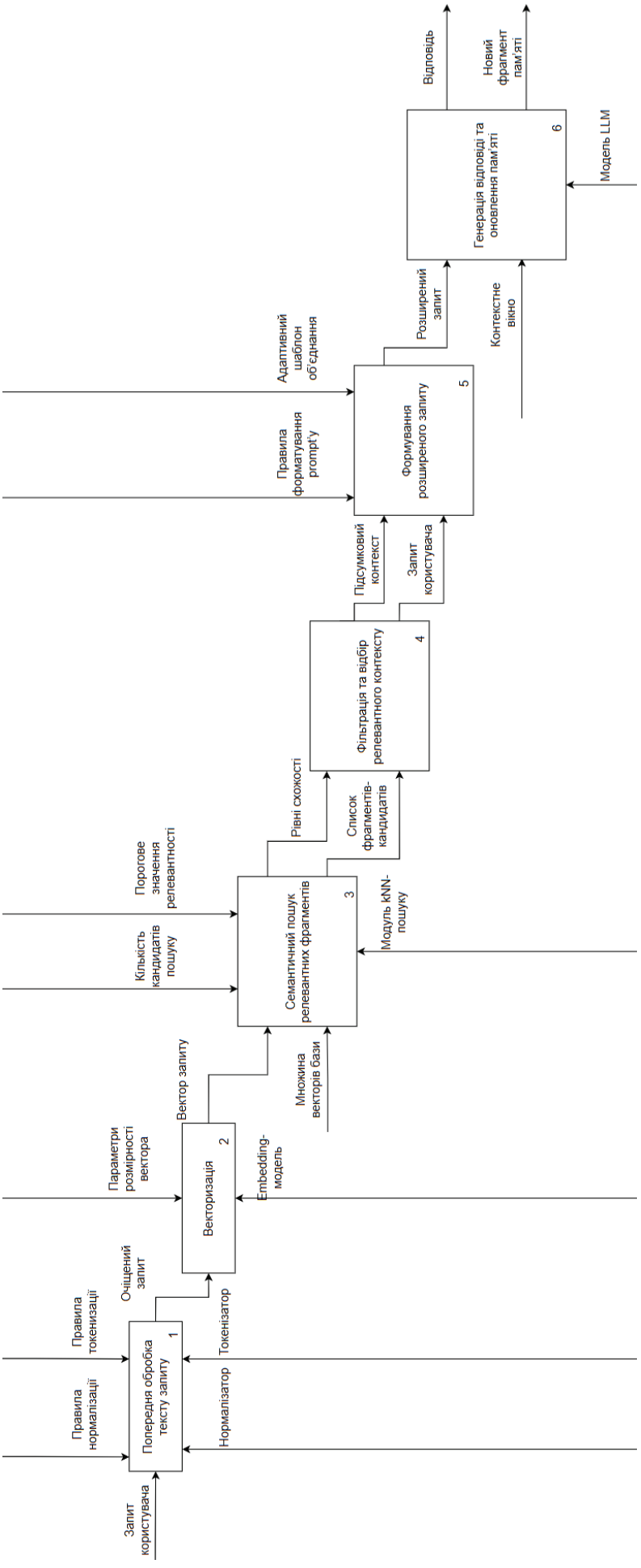

Рисунок 3.1 – Архітектурна схема системи, що реалізує метод доповнення запитів контекстом

3.2 Реалізація інформаційної технології

Розроблена система має багаторівневу архітектуру, що включає модулі фронтенду та бекенду, а також інтегровані компоненти: векторну БД знань і локальну LLM. На рисунку 3.2 надана архітектура взаємодії між компонентами системи під час формування відповіді LLM з використанням векторної БД.

Фронтенд забезпечує інтерактивний інтерфейс, через який користувач вводить запит та отримує згенеровану відповідь. Бекенд відповідає за обробку запиту та координує взаємодію між усіма компонентами. Зокрема, бекенд-модуль реалізує pipeline обробки: нормалізацію тексту запиту, обчислення векторного подання, семантичний пошук у базі знань та генерування відповіді за допомогою LLM. Векторна БД слугує довготривалою пам’яттю системи – у ній зберігаються текстові фрагменти у вигляді числових векторів ознак. LLM-модель інтегрована для генерації текстових відповідей; вона отримує розширений запит з доданим контекстом із бази знань і формує змістовну відповідь користувачеві.

Для спрощення розгортання прототипу та забезпечення портативності використано контейнеризацію. Застосовано технологію Docker, що дозволяє упакувати кожен компонент системи у відокремлений контейнер із усіма необхідними бібліотеками та налаштуваннями середовища. Таким чином досягається ізоляція процесів та відтворюваність запуску: незалежно від локального оточення користувача всі модулі стартують у стандартизованому вигляді. Наприклад, векторна БД розгорнута як окремий контейнер, що прослуховує локальний порт і обробляє запити семантичного пошуку. Бекенд-сервер розміщено в іншому контейнері на базі образу Python з необхідними залежностями.

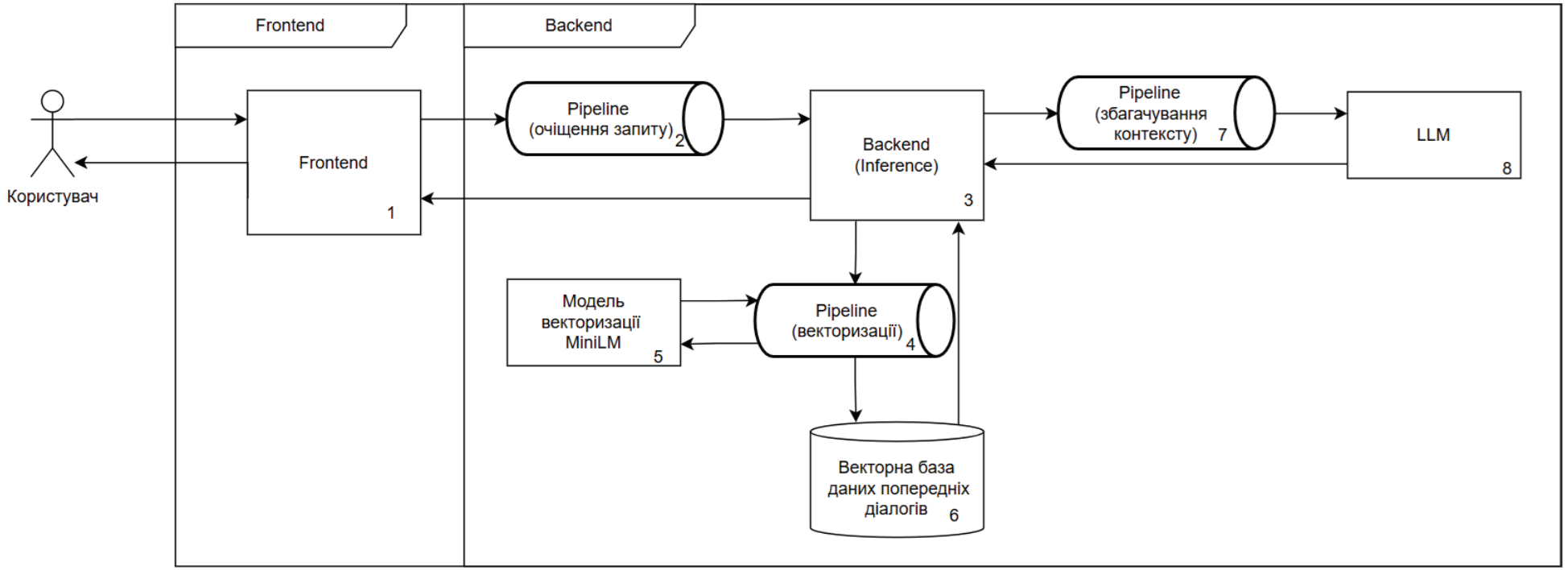

Рисунок 3.2 – Архітектура взаємодії між компонентами системи під час формування відповіді LLM з використанням векторної БД

Можливе також виділення контейнера для LLM-модуля, якщо модель потребує окремого сервісу або апаратного прискорення; у прототипі натомість використано вбудований запуск моделі в межах бекенду для спрощення. Координація кількох контейнерів здійснюється через Docker Compose: відповідний Yet Another Markup Language-файл описує склад системи і дозволяє запустити усі компоненти однією командою. Окрім того, додаткові скрипти автоматизують підготовчі кроки: ініціалізацію сховища, запуск завдання зі створення embedding для нових даних, моніторинг стану сервісів. В ході роботи система здійснює логування ключових подій – запити користувачів, результати пошуку, час відгуку моделей – що спрощує налагодження та аналіз ефективності. Логи виводяться в консоль кожного сервісу і при необхідності можуть зберігатися у файл для подальшого аналізу.

Користувач вводить текст запиту у інтерфейсі фронтенду. Після натискання кнопки відправлення запит у форматі HyperText Transfer Protocol-запиту передається на бекенд-сервер. Фронтенд не містить бізнес-логіки обробки – його роль полягає у зборі вводу, відправленні його на сервер та відображенні отриманої відповіді.

Бекенд отримує запит користувача через Representational State Transfer API і розпочинає обробку. Першим кроком застосовується модуль нормалізації тексту: видаляються зайві пробіли, специфічні символи або розмітка, за потреби приводиться до єдиного регістру або формату. Метою цього етапу є усунути можливі шуми і неоднозначності у вхідному тексті, які могли б негативно вплинути на подальше семантичне порівняння. На рисунку 3.3 надана схема етапів очищення користувацького запиту на етапі нормалізації тексту. Нормалізований та підготовлений до аналізу текст передається на наступний етап.

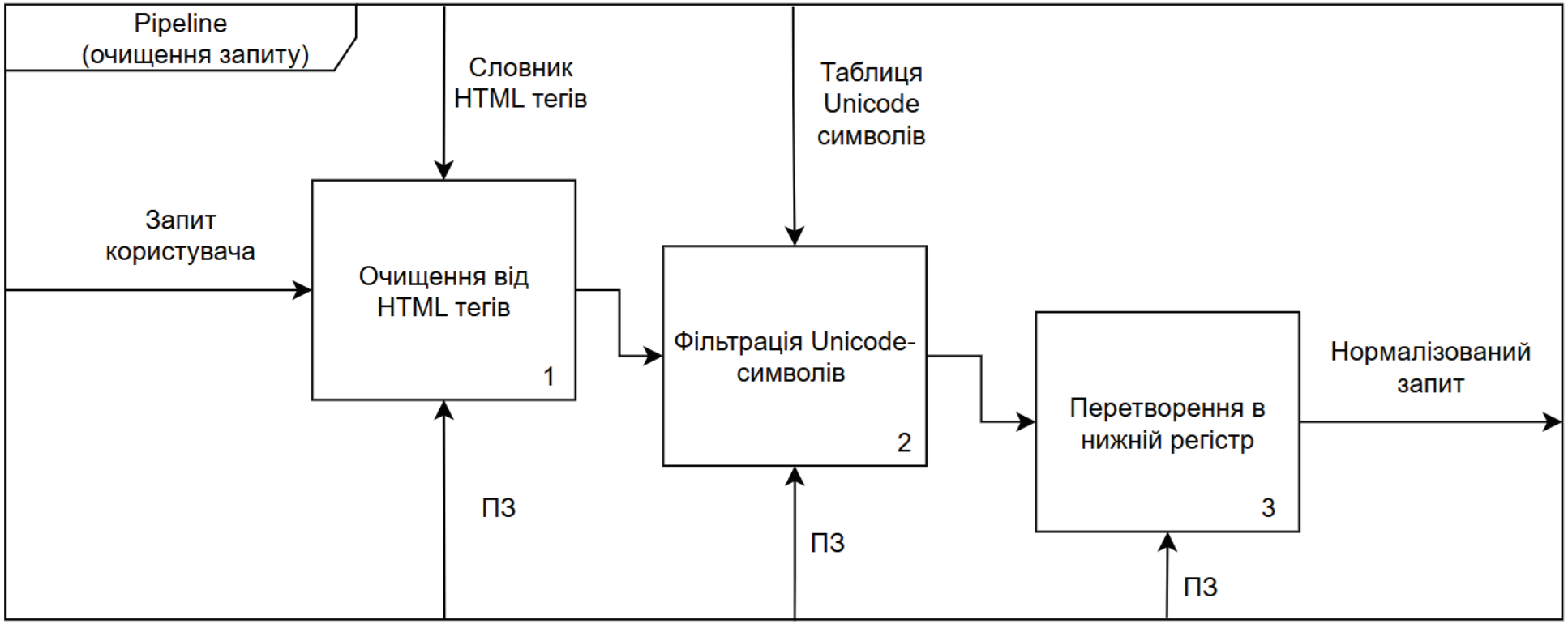

Рисунок 3.3 – Схема етапів очищення користувацького запиту на етапі нормалізації тексту

На основі очищеного запиту обчислюється його векторне представлення за допомогою заздалегідь навченого embedding-модуля. Цей модуль побудований на моделі глибокого навчання, що перетворює рядок тексту у точку в багатовимірному векторному просторі. На рисунку 3.4 надана Pipeline векторизації запиту з використанням моделі MiniLM. В прототипі використано готову модель трансформера для embedding, що повертає вектор фіксованої розмірності. Отриманий вектор характеризує семантичний зміст запиту: близькі за змістом запити матимуть розташування векторів, близьке один до одного. Важливо зазначити, що використання однакової моделі та методики нормалізації як для запитів, так і для текстів у базі знань забезпечує коректність порівняння – всі вектори знаходяться в одному просторі ознак.

Розрахований вектор запиту надходить до векторної БД, де виконується семантичний пошук – визначення найбільш схожих векторів серед тих, що збережені у сховищі знань. Бекенд надсилає запит до API БД, вказуючи вектор запиту та бажану кількість результатів k. Векторна СУБД здійснює високопродуктивний пошук найближчих сусідів: обчислює міру схожості між вектором запиту та кожним вектором у колекції. У результаті база повертає топ-k найбільш релевантних фрагментів – тобто тих, чиї вектори максимально близькі до вектора запиту. Разом з кожним знайденим вектором зберігається пов’язаний з ним контент: текст фрагмента знань та додаткова мета-інформація. Бекенд отримує список знайдених фрагментів упорядкований за спаданням схожості. Таким чином відбирається невелика кількість контекстних даних, найбільш релевантних до запиту; нерелевантні дані відсіюються і не потрапляють до контексту, що підвищує точність подальшої відповіді.

На основі отриманих з БД фрагментів знань бекенд будує розширений контекст для мовної моделі. До оригінального запиту користувача автоматично додаються знайдені релевантні тексти – цей об’єднаний текст і становить prompt для LLM.

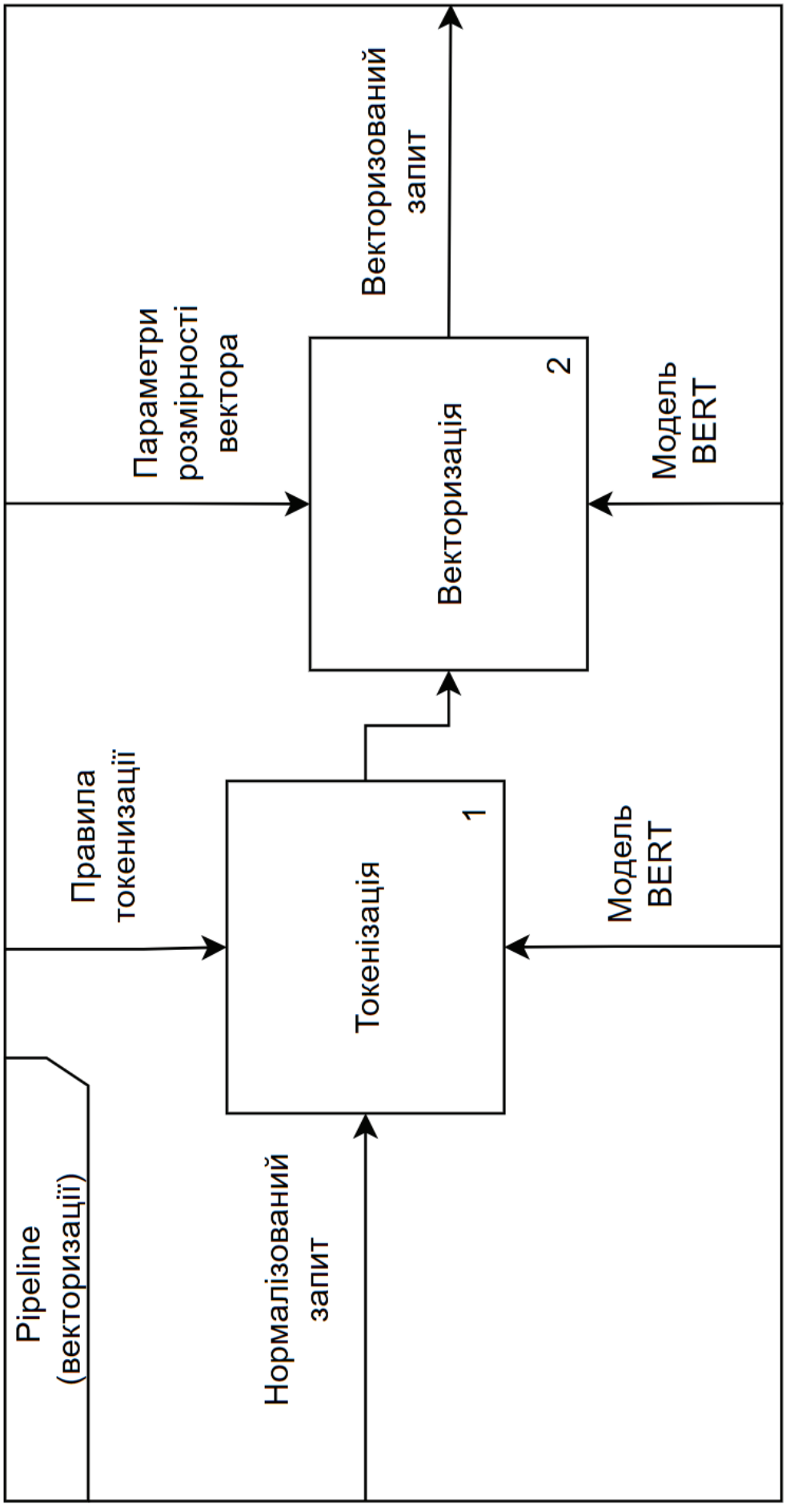

Рисунок 3.4 – Pipeline векторизації запиту з використанням моделі MiniLM

На рисунку 3.5 надана Pipeline збагачення запиту контекстом попередніх діалогів. Формування prompt'у може включати шаблонізацію: наприклад, спершу вставляються фрагменти контексту зі сховища, а далі – сам запит користувача. Такий підхід дозволяє інтегрувати зовнішні знання у процес генерації: мовна модель отримує не лише питання, а й додаткові підказки або факти, на які вона повинна спиратися. У результаті зменшується ризик так званих «галюцинацій» моделі, коли вона вигадує відповіді без опори на дані: LLM має актуальний контекст і менше схильна генерувати недоречну інформацію.

LLM отримує на вхід злитий prompt і генерує продовження у вигляді текстової відповіді. Генерація здійснюється авторегресивно токен за токеном до досягнення умов завершення. Відповідь моделі містить сформульовану природною мовою реакцію на запит користувача з урахуванням наданого контексту. Наприклад, якщо запит стосується раніше обговорюваної теми, LLM збереже наступність, використовуючи факти з витягнутих фрагментів діалогу.

Згенерована моделью відповідь надсилається бекендом назад на фронтенд у форматі HyperText Transfer Protocol-відповіді. Фронтенд отримує ці дані через веб-запит і відображає текст відповіді у інтерфейсі для користувача. На цьому цикл обробки завершується, і користувач бачить результат. Якщо діалог продовжується, система може додатково зберегти нову інформацію у базі знань: наприклад, отримане запитання-відповідь або згенерований моделью підсумок поточної розмови.

Такий механізм дозволяє поповнювати векторну базу актуальними даними по мірі взаємодії. Кожен новий запит повторює описаний pipeline, забезпечуючи динамічне врахування як поточної, так і довготривалої інформації.

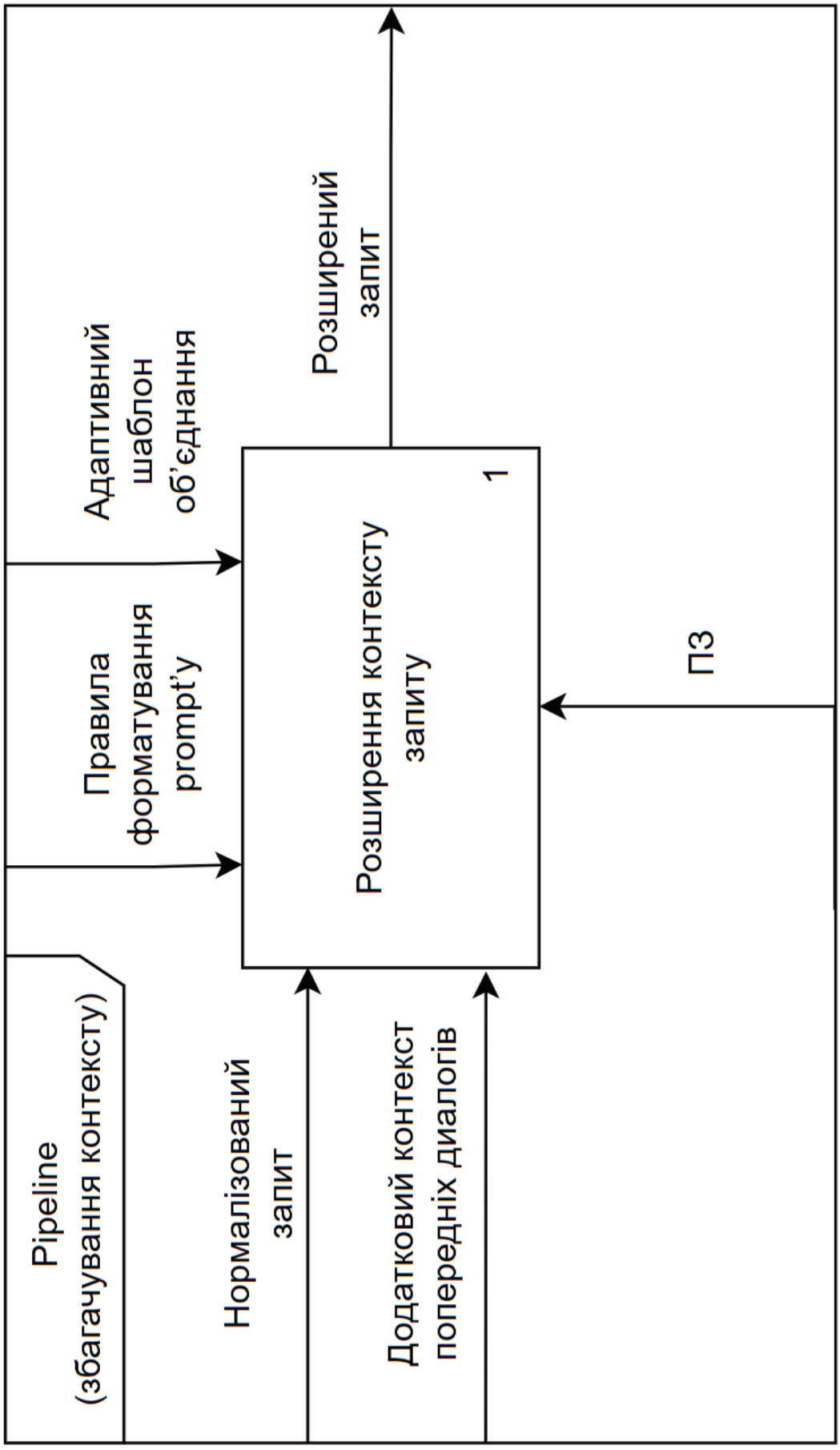

Рисунок 3.5 – Pipeline збагачення запиту контекстом попередніх діалогів

Таким чином, реалізована інформаційна технологія являє собою демонстраційний прототип, який поєднує векторне семантичне сховище знань та генеративну мовну модель. Архітектура модульна і масштабована: за необхідності фронтенд можна замінити або доповнити, модель – оновити на потужнішу, а базу знань – розгорнути на віддаленому сервері для роботи з більшими даними. Використання контейнерів Docker забезпечує легкий запуск на будь-якій машині та ізоляцію компонентів один від одного. Потокова обробка запитів гарантує, що кожна відповідь моделі максимально враховує релевантний контекст із наявних даних, підтримуючи цілісність діалогу і точність інформації. У цілому, обрані технічні рішення і структура системи підтверджують практичну здійсненність запропонованого методу формування запитів та створюють основу для його подальшого вдосконалення і масштабування.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА

4.1 Опис програмного модуля

Програмна реалізація запропонованої інформаційної технології формування запитів до великих мовних моделей організована у вигляді низки взаємопов’язаних модулів на мові Python продемонстровані у додатку А.

У складі програмного модуля, що реалізує запропоновану інформаційну технологію, ключову роль відіграє послідовний pipeline перетворення тексту запиту, взаємодії з векторною базою пам’яті та формування розширеного промпту для великої мовної моделі. Логіка цього pipeline матеріалізується у вигляді низки функцій в окремих підмодулях, які узгоджуються з формалізацією методу, поданою в теоретичних розділах роботи.

Функція preprocess_text реалізує описаний у роботі модуль нормалізації вхідного запиту й виконує попередній етап очищення тексту перед векторизацією. На вхід надходить довільний рядок, що може містити HTML-розмітку, спеціальні символи та артефакти форматування, характерні для веб-інтерфейсів. Усередині функції послідовно застосовуються регулярні вирази для видалення HTML-тегів, далі здійснюється фільтрація символів за узгодженим алфавітом: зберігаються літери, цифри, базова пунктуація та пробіли, а також символи, що належать до літерних категорій Unicode. Такий підхід дозволяє зберегти корисну текстову інформацію, одночасно усуваючи шумові елементи, які не несуть змістовного навантаження, але здатні впливати на embedding-представлення. Наступний крок полягає у нормалізації пропусків: множинні пробіли та переведення рядків замінюються одним пробілом, крайні пробіли видаляються. Завершальним етапом є приведення тексту до нижнього

регістру, що робить подальший семантичний пошук інваріантним до регістру та узгоджує поведінку із підходами, описаними в теоретичній частині під час аналізу нормалізації запиту. У підсумку функція повертає стандартизований рядок, який виступає стабільною основою для подальшої векторизації та порівняння у векторному просторі.

Наступна група функцій пов’язана з формуванням векторних представлень сумаризованих спогадів і їхнім розміщенням у векторному сховищі. Функція create_embedding_from_summary приймає на вхід текст резюме діалогу й передає його до embedding-моделі типу sentence-transformers, яка в прототипі відповідає попередньо навченій моделі all-MiniLM-L6-v2. Модель перетворює текст у вектор фіксованої розмірності 384, що узгоджується з описом структури експериментальних даних: підсумки діалогів у роботі також зберігаються разом з embedding-координатами цієї розмірності. Вектор повертається у вигляді масиву типу NumPy, що використовується як вхід для індексації у векторній базі.

Функція add_memory_to_vector_store реалізує інтеграцію об’єкта довготривалої пам’яті до Facebook Artificial Intelligence Similarity Search (FAISS)-індексу, який у роботі розглядається як основний інструмент семантичного пошуку найближчих сусідів. На першому етапі одновимірний embedding перетворюється у двовимірне представлення розміру 1×d та приводиться до типу float32 відповідно до вимог FAISS. Далі застосовується L2-нормалізація, що забезпечує одиничну довжину вектора. Така нормалізація узгоджується з теоретичним положенням про те, що для нормалізованих векторів скалярний добуток та евклідова відстань дають еквівалентні ранжування відносно косинусної подібності, яка використовується як базова міра семантичної близькості у роботі.

Після нормалізації вектор додається до FAISS-індексу плоского типу, а поточна позиція в індексі зберігається у двосторонніх асоціативних структурах: від ідентифікатора MemoryUnit до індексу й навпаки. Така мапа забезпечує можливість швидко відновити текстовий спогад за індексом,

отриманим під час пошуку, та підтримує узгодженість між текстовою та векторною репрезентаціями пам’яті.

Функція search_relevant_memories реалізує семантичний пошук релевантних одиниць пам’яті для поточного запиту користувача.

На вхід надходить текст запиту, який нормалізується функцією preprocess_text, що забезпечує одноманітність обробки запитів і сумаризованих спогадів. Нормалізований текст кодується embedding-моделлю у вектор запиту, який, приводиться до типу float32 та L2-нормалізується. Отриманий вектор подається до FAISS-індексу для виконання операції пошуку топ-k найближчих сусідів. У відповідь індекс повертає матриці подібностей і індексів елементів, де кожне значення подібності вже відповідає косинусній мірі схожості в діапазоні від −1 до 1. Індекси переводяться у ідентифікатори MemoryUnit через попередньо сформовану мапу, після чого формується список пар об’єкт пам’яті та числове значення подібності. Така реалізація відтворює формальний механізм вибору множини фрагментів $C(q)$, описаний у теоретичній частині, і виступає основою для побудови розширеного запиту.

Подальшим кроком у pipeline є формування промптів для великої мовної моделі з урахуванням знайденого контексту. Функція generate_query_with_memory приймає нормалізований текст запиту та список релевантних спогадів із векторної бази. Всередині функції формується текстовий блок контексту: кожний спогад супроводжується його порядковим номером, значенням подібності, заокругленим до двох десяткових знаків, та текстом підсумку, що відповідає властивості summary об’єкта MemoryUnit. Така форма представлення контексту робить pipeline прозорим з погляду відлагодження та аналізу, оскільки дозволяє явно бачити, які саме спогади було активовано для даного запиту. Далі контекстні фрагменти конкатенуються у єдиний текстовий блок, який разом із нормалізованим запитом включається до повідомлення користувача у форматі, сумісному з API LLM. Паралельно формується системне

повідомлення з інструкцією для моделі: явно зазначається, що подається додатковий контекст з попередніх діалогів, який слід використовувати за наявності релевантності. Така двокомпонентна структура промпту фактично реалізує функцію $P(H_{n-1}, C(q), q)$, описану у формальній постановці методу, де історія діалогу відображена через узагальнені спогади з векторної бази.

Функція generate_query_without_memory реалізує базовий сценарій без залучення довготривалої пам'яті. На відміну від попереднього варіанту, у системному промпті LLM задається інструкція ігнорувати будь-який зовнішній контекст, а в повідомленні користувача передається лише текст вихідного запиту. Порівняння результатів роботи двох функцій дозволяє кількісно оцінити внесок векторної пам'яті в релевантність відповідей.

Окремий функціональний блок присвячений підсумовуванню завершених діалогів і створенню одиниць пам'яті. Функція summarize_dialog отримує екземпляр Dialog, у якому збережено повну історію взаємодії користувача й асистента. Через метод get_text_content діалог перетворюється на лінійний текстовий опис, що узгоджується з форматом, використаним при побудові експериментального корпусу. На основі цього тексту формується спеціалізований промпт до LLM, у якому задається інструкція створити компактний резюме-фрагмент знань обсягом 5–10 речень із фокусом на ключових питаннях, відповідях та деталях, потенційно корисних у майбутніх діалогах. Виклик до llm_client.chat виконується з низьким значенням параметра температури, що мінімізує стохастичність та підвищує повторюваність підсумків для тієї самої розмови. Отримане резюме передається у фабричний метод MemoryUnit.create_from_dialog, який створює об'єкт довготривалої пам'яті з посиланням на вихідний діалог, часовими мітками та текстовим підсумком. Далі над підсумком обчислюється embedding за допомогою тієї ж моделі all-MiniLM-L6-v2, що використовується і для інших частин системи, після чого створений вектор разом з об'єктом пам'яті додається у

VectorStore. Такий механізм забезпечує узгодженість між описаною в роботі концепцією людоподібної пам’яті й конкретною програмною реалізацією, де зберігаються не повні транскрипти, а структуровані узагальнення.

Функція handle_user_query виступає центральним оркестратором pipeline обробки запиту користувача, логіка якого у розділі програмної реалізації пов’язується з файлом pipeline.py.

На вхід подається сирий текст запиту, екземпляр менеджера пам’яті та клієнт LLM. Спочатку викликається preprocess_text, що приводить запит до нормалізованої форми. Потім через метод retrieve_relevant_memories виконується семантичний пошук у векторному сховищі з відбором до трьох найближчих спогадів з урахуванням порогу подібності, як це описано при характеристиці роботи MemoryManager та VectorStore. Після отримання контексту формуються два промпти: базовий – за допомогою generate_query_without_memory, та розширений – через generate_query_with_memory. Обидва варіанти подаються до LLM через виклики llm.chat, унаслідок чого отримуються дві відповіді: без пам’яті та з пам’яттю. Завершальним етапом функції є оцінювання релевантності обох відповідей за допомогою функції evaluate_relevance, однак із різними умовами: для відповіді без пам’яті оцінка проводиться лише відносно вихідного запиту, тоді як для відповіді з пам’яттю до промпту арбітра додається текстовий контекст активованих спогадів. Результатом роботи функції є структурований словник, що містить обидві відповіді, їхні числові оцінки та перелік використаних спогадів, що повністю узгоджується з форматом експериментального набору даних, представленим у роботі.

Функція evaluate_relevance реалізує підхід автоматизованого оцінювання, у якому роль експерта-оцінювача виконує окрема велика мовна модель, налаштована на виконання функції арбітра релевантності. Усередині функції формується системний промпт, де задається роль моделі як експерта з оцінки відповідей, а також описується шкала від 0 до 3 балів, що використовується в експериментальній частині роботи. Далі

створюється користувацький промпт, який містить текст запиту, відповідь, що підлягає оцінці, та, за наявності, додатковий контекст. Застосовується низька температура генерації, що зменшує варіативність рішень арбітра і забезпечує більш стабільні оцінки. Після виклику arbitrator_llm.chat функція отримує вербальне пояснення та числову оцінку, інкапсульовану в тексті відповіді. Допоміжна процедура parse_score_from_evaluation виділяє числове значення, яке інтерпретується як оцінка релевантності, тоді як текстова частина використовується як обґрунтування. У результаті повертається структура з полями score і justification, що надалі використовується у агрегуванні статистичних показників, наведених у розділі експериментальної оцінки.

Сукупність описаного функціоналу підтверджує практичну здійсненність запропонованого підходу та створює основу для його подальшого розширення в напрямку складніших стратегій векторного пошуку й оптимізованих схем формування контексту до LLM.

4.2 Опис експериментальних даних

Експериментальний набір даних сформовано таким чином, щоб відобразити повний цикл роботи системи: від накопичення історичних діалогів та їхньої сумаризації до отримання відповідей LLM з підключенням або без підключення довготривалої пам’яті та подальшої оцінки релевантності цих відповідей. Фокус робиться саме на структурі даних, а не на алгоритмічних деталях, що дає змогу розглядати набір як самодостатній об’єкт для відтворення та аналізу експериментів.

Початкову основу становить корпус історичних діалогів, які моделюють різні сценарії взаємодії користувача з асистентом у технічних, навчальних та прикладних запитах. Корпус включає діалогів українською та

частково англійською мовами. Середня довжина одного діалогу – приблизно 13 реплік. Для кожного діалогу фіксується текст усіх повідомлень у хронологічному порядку, часові мітки, ролі, а також службові поля, зокрема ідентифікатор діалогу та ознака активності чи завершеності. Такий формат дає можливість як відтворювати повну історію спілкування, так і використовувати її для побудови стиснутих спогадів.

На основі завершених діалогів сформовано набір сумаризованих спогадів, що зберігаються у векторній БД. Середня довжина одного підсумку знаходиться в діапазоні 70–90 слів, що забезпечує баланс між інформативністю та компактністю; максимальна довжина не перевищує приблизно 150 слів. Для кожного резюме у наборі даних збережено текст підсумку, посилання на вихідний діалог, час створення та векторні координати embedding-представлення розмірності 384, сформовані вбудованою моделлю embedding.

Окрему частину експериментальних даних становить тестова множина запитів. Вона містить запити, які сконструйовано таким чином, щоб частина з них критично залежала від інформації, що міститься в попередніх діалогах, тоді як інша частина могла бути коректно оброблена без звернення до історії. Наприклад, значна частина «контекст-залежних» запитів має форму уточнень щодо раніше узгоджених планів, параметрів задачі або раніше введених позначень, тоді як контекст-незалежні запити, як правило, належать до загальних інформаційних питань. Для кожного тестового запиту зберігаються його текстова форма, ідентифікатор, зв’язок із конкретними діалогами та службові поля, які використовуються на етапі побудови вибірок.

Зведена структура основних компонентів експериментальних даних представлена на лістингу 4.1.

Лістинг 4.1 – Зведена структура основних компонентів експериментальних даних


```
{
  "id": "seed-10",
  "messages": [
    {
      "role": "user",
      "content": "Перед релізом потрібно узгодити документацію.",
      "timestamp": "2025-10-17T13:30:00Z"
    },
    {
      "role": "assistant",
      "content": "Документація має містити опис API, схеми інтеграції та приклади тестових сценаріїв.",
      "timestamp": "2025-10-17T13:30:04Z"
    }
  ],
  "created_at": "2025-10-17T13:30:00Z",
  "last_activity_at": "2025-10-17T13:30:04Z",
  "is_active": false
}
```


Для кожного запиту проводяться два окремі прогони мовної моделі. Перший прогін використовує лише текст запиту, без додавання будь-яких резюме з векторної БД, унаслідок чого формується відповідь, що відображає «чисту» поведінку моделі. Другий прогін здійснюється після виконання пошуку у векторному сховищі: на основі embedding-вектора запиту відбираються до трьох найближчих спогадів, які перевищили заданий поріг подібності, після чого їхні тексти додаються як контекст перед основним запитанням. Таким чином для кожного експериментального запиту у наборі даних наявна пара вихідних текстів: відповідь без пам’яті та відповідь з пам’яттю. На рисунку 4.2 зображено приклад роботи програми на тестовому запиті.

```
Результати семантичного пошуку
Знайдено 3 релевантних спогади:

Спогад #1
ID: 8a0d9b91-a418-4e36-a9f3-5104241ea27c
Схожість: 0.5348
Підсумок:
Символ f(x) у моделі позначає функцію, яка перетворює вхідний сигнал на відповідний вектор ознак. Це важливий
елемент для розуміння процесу обробки даних у машинному навчанні та моделюванні.

Спогад #2
ID: 89ec7658-b9b3-42cd-8f81-90d44803cd6
Схожість: 0.5246
Підсумок:
Параметр τ у алгоритмі відбору позначає поріг схожості, який використовується для визначення релевантності
векторів. Цей параметр є ключовим для забезпечення точності відбору, оскільки він впливає на те, які вектори
будуть вважатися достатньо схожими для подальшої обробки. Розуміння ролі τ допомагає в оптимізації алгоритму
та покращенні результатів.

Спогад #3
ID: 6b989efa-3652-488e-bf95-f0037ffe8c73
Схожість: 0.5206
Підсумок:
Параметр λ у функції втрат є коефіцієнтом регуляризації. Він відповідає за контроль внеску штрафного члена, що
допомагає запобігти перенавчанню моделі. Регуляризація є важливим аспектом у машинному навчанні, оскільки вона
покращує узагальнюючу здатність моделі.

Запит з пам'яттю (що відправляється до LLM)

Контекст з попередніх діалогів:

[1] Схожість: 0.53
Підсумок: Символ f(x) у моделі позначає функцію, яка перетворює вхідний сигнал на відповідний вектор ознак. Це
важливий елемент для розуміння процесу обробки даних у машинному навчанні та моделюванні.

[2] Схожість: 0.52
Підсумок: Параметр τ у алгоритмі відбору позначає поріг схожості... (і весь текст далі)

[3] Схожість: 0.52
Підсумок: Параметр λ у функції втрат є коефіцієнтом регуляризації...

Поточний запит користувача: "а що позначає параметр τ у нашому алгоритмі відбору?"
```

Рисунок 4.1 – Приклад роботи програми на тестовому запиті

Формат таких записів ілюструє таблиця 4.1, у якій наведено три типові приклади для різних сценаріїв залежності від контексту.

Ключова відмінність полягає у способі оцінювання релевантності відповідей. Роль експерта-оцінювача виконує окрема LLM від OpenAI, налаштована через спеціальний системний промпт на виконання функції «арбітра релевантності». Для кожної пари «запит – відповідь» формується службовий промпт, у якому описано шкалу оцінювання та наведено інструкцію: визначити, наскільки відповідь відповідає інформаційному запиту, уникнути упередженості та надати результат у вигляді цілого числа від 0 до 3. Модель-експерт обробляє окремо відповідь без контексту та відповідь з контекстом, повертаючи числову оцінку й коротке текстове обґрунтування.

Таблиця 4.1 – Приклади записів у експериментальному наборі

Короткий опис запиту	Кількість релевантних спогадів	Сумарна довжина контексту, токенів	Оцінка без контексту (0–3)	Оцінка з контекстом (0–3)
Повернення до раніше узгодженого плану впровадження	2	310	1	3
Уточнення раніше введеного позначення у формальному описі	1	120	2	3
Загальне питання, що не спирається на історію	0	0	3	3

У наборі експериментальних даних зберігаються щонайменше числові бали; текстові пояснення можуть додатково використовуватися для аналізу типових помилок. На рисунку 4.3 відображен результат оцінки відповідей.

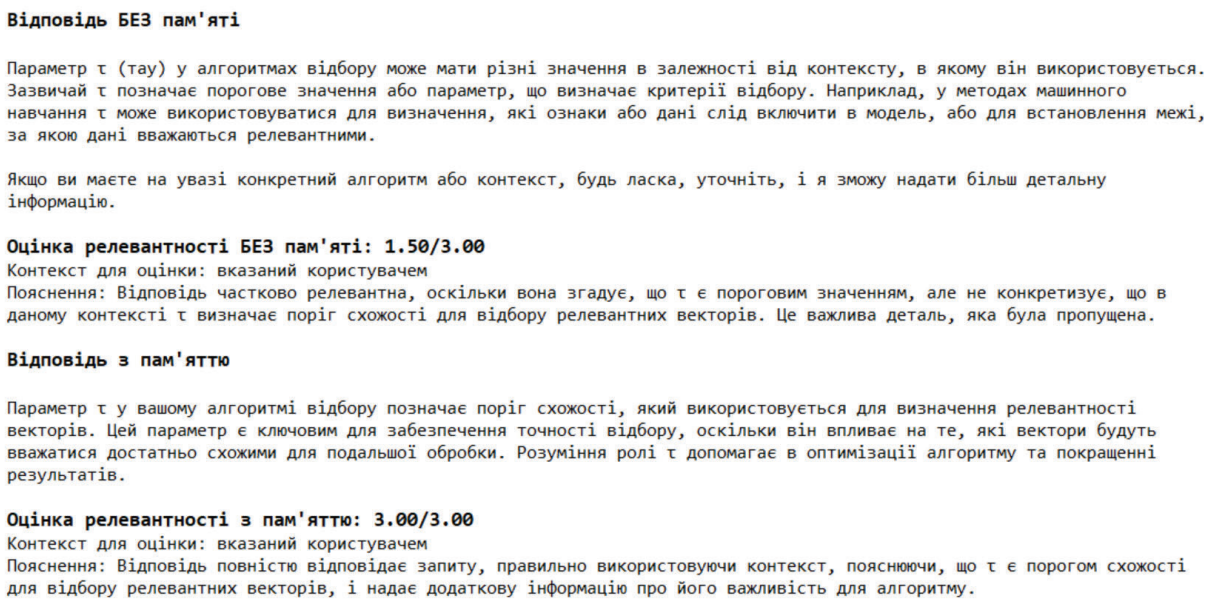

Рисунок 4.2 – Результат оцінки відповідей

Шкала інтерпретується наступним чином: значення 0 відповідає повній нерелевантності або серйозним логічним помилкам; одиниця означає часткову релевантність, коли відповідь торкається теми, але пропускає важливі аспекти або містить істотні неточності; значення 2 характеризує загалом коректну, але неповну або з незначними похибками відповідь; значення 3 відображає повну, логічно послідовну й релевантну відповідь, що повністю задовольняє запит. Завдяки використанню однієї й тієї ж моделі OpenAI як автоматизованого експерта для всіх прикладів досягається консистентність критеріїв оцінювання, а також знімається необхідність у залученні декількох людських анотараторів для кожного прикладу.

Сформований набір експериментальних даних, таким чином, фіксує усі необхідні вхідні й вихідні величини для об’єктивного порівняння поведінки LLM до та після додавання сумаризованого контексту з векторної

БД.

Нижче подано узагальнювальну таблицю 4.2, яка демонструє вплив вставки сумаризованого історичного контексту з векторної БД на якість відповіді LLM. Дані відповідають логіці експериментального набору, сформованого у роботі, та ґрунтуються на оцінюванні відповідей моделлю-експертом OpenAI, що виконує функцію незалежного арбітра релевантності.

Таблиця 4.2 – Узагальнений вплив технології на якість відповідей LLM

Показник	Режим без пам’яті	Режим з пам’яттю	Абсолютна зміна
Середня оцінка релевантності (0–3)	1,82	2,64	+0,82
Частка відповідей, що отримали ≥2	58%	86%	+28%
Частка відповідей з максимальною оцінкою 3	22%	49%	+27%

Узагальнені результати демонструють стійку позитивну тенденцію: підключення сумаризованої пам’яті з векторної БД суттєво підвищує релевантність відповідей моделі. Найбільш помітним є збільшення частки відповідей з максимальною оцінкою – майже удвічі, що вказує на покращення якості саме там, де потрібні точні та контекстуально складні відповіді. Приріст середньої оцінки у +0.82 бала на чотирибальній шкалі розглядається як значний ефект для задач, залежних від довготривалої історії.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досліджено методи вирішення функціональної задачі підвищення релевантності відповідей пошукових систем на основі інтеграції семантичного пошуку та великих мовних моделей. Проведено аналіз сучасних архітектур пошукових систем, підходів до формування запитів і побудови контексту, а також огляд наявних рішень на базі RAG. Сформульовано та обґрунтовано вимоги до інформаційної технології, яка забезпечує використання векторних представлень документів і історії попередніх звернень користувача для уточнення запитів і відбору релевантних фрагментів.

Розроблено математичні моделі та алгоритмічні процедури семантичного пошуку, повторного ранжування та інтеграції результатів у запит до великої мовної моделі, що дозволяє підвищити точність відповідей та зменшити кількість нерелевантних результатів. Реалізовано програмний прототип функціонального модуля, який взаємодіє з векторною базою даних і LLM, забезпечуючи експериментальне порівняння з базовими підходами формування відповідей. Експериментальні дослідження показали покращення показників релевантності відповідей пошукової системи та стабільності результатів при роботі з великими корпусами текстів.

Основні результати роботи, пов’язані з дослідженням моделей та методів підвищення релевантності відповідей пошукових систем, були представлені та обговорені в доповіді «Дослідження моделей та методів вирішення функціональної задачі підвищення релевантності відповідей пошукових систем» на II Міжнародній науково-теоретичній конференції «Modern science and innovation: trends, challenges, and breakthroughs».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bridgelall R. Unraveling the mysteries of AI chatbots. Artificial Intelligence Review. 2024. T.57. №89. DOI: 10.1007/s10462-024-10720-7.

2. Unleashing the potential of prompt engineering for large language models / Chen B., Zhang Z., Langrené N., Zhu S. // Patterns. 2025. T. 6, № 6. DOI: 10.1016/j.patter.2025.101260.

3. Nechita M., Raschip M. A Comparative Study of ML Approaches for Detecting AI-Generated Essays // Proceedings of the 14th International Conference on Data Science, Technology and Applications (DATA 2025) – Volume 1: DATA. Setúbal: SciTePress, 2025. P. 144–155. DOI: 10.5220/0013570200003967.

4. Levy M., Jacoby A., Goldberg Y. Same Task, More Tokens: the Impact of Input Length on the Reasoning Performance of Large Language Models // Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Bangkok, Thailand, August 2024. Association for Computational Linguistics, 2024. P. 15339–15353. DOI: 10.18653/v1/2024.acl-long.818.

5. Chatterjee S. The Impact of Prompt Bloat on LLM Output Quality URL: https://mlops.community/the-impact-of-prompt-bloat-on-llm-output-quality/ (дата звернення: 24.11.2025).

6. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models / Wei J., Wang X., Schuurmans D., та ін. arXiv. 2022. DOI: 10.48550/arXiv.2201.11903.

7. A practical guide to Retrieval-Augmented Generation (RAG) URL: https://www.k2view.com/what-is-retrieval-augmented-generation (дата звернення: 25.11.2025).

8. A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge / Ma L., Zhang R., Han Y., та ін. arXiv. 2023.

DOI: 10.48550/arXiv.2310.11703.

9. Roller J. Vector Databases: The Foundation of Modern AI Applications URL: https://www.computer.org/publications/tech-news/community-voices/vector-databases-and-ai-applications (дата звернення: 26.11.2025).

10. Ricadela A. What Is Weaviate? A Semantic Search Database URL: https://www.oracle.com/database/vector-database/weaviate/ (дата звернення: 07.12.2025).

11. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks / arXiv. 2019. arXiv:1908.10084. DOI: 10.48550/arXiv.1908.10084.

12. Vector Similarity Explained URL: https://www.pinecone.io/learn/vector-similarity/ (дата звернення: 07.12.2025).

13. Salton G., Buckley C. Term-weighting approaches in automatic text retrieval // Information Processing & Management. 1988. Vol. 24, No. 5. P. 513–523. DOI: 10.1016/0306-4573(88)90021-0.

14. Robertson S. E., Zaragoza H. The Probabilistic Relevance Framework: BM25 and Beyond // Foundations and Trends in Information Retrieval. 2009. Vol. 3, Nos. 4–5. P. 333–389. DOI: 10.1561/1500000019.

15. Simple Yet Effective Neural Ranking and Reranking Baselines for Cross-Lingual Information Retrieval / Lin J., Alfonso-Hermelo D., Jeronymo V. та ін. arXiv. 2023. DOI: 10.48550/arXiv.2304.01019.

16. Lin S.-C., Lin J. A Dense Representation Framework for Lexical and Semantic Matching // ACM Transactions on Information Systems. 2023. Vol. 41, No. 4. P. 1–110. DOI: 10.1145/3582426.

17. Karpukhin V., Oguz B., Min S. та ін. Dense Passage Retrieval for Open-Domain Question Answering // arXiv. 2020. arXiv:2010.11386.

18. Hybrid Retrieval Approach for Advancing Retrieval-Augmented Generation Systems / Doan N. N., Härmä A., Celebi R., та ін. Proceedings of the 7th International Conference on Natural Language and Speech Processing. Trento: Association for Computational Linguistics, 2024. P. 397–409.

19. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks / Lewis P., Perez E., Piktus A. та ін. Advances in Neural Information Processing Systems 33 (NeurIPS 2020). 2020. P. 9459–9474.

20. Чалий С.Ф, Лещинська І.О Доповнення вхідних даних у ментальній моделі користувача інтелектуальної системи. Проблеми інформатизації . Т. 2. 2024. С. 46.

21. ChatDB: Augmenting LLMs with Databases as Their Symbolic Memory // arXiv. 2023. DOI:10.48550/arXiv.2306.03901

22. Qu X. та ін. Unified Debates: Efficient Performance Evaluation with Behavior-based Metrics in Preference Alignment // arXiv. 2024. arXiv:2412.12559.

23. Long Context vs. RAG for LLMs: An Evaluation and Revisits // arXiv. 2025. DOI: 10.48550/arXiv.2501.01880

24. Xu Z., Lin J., Srikumar V. A Survey of Model Architectures in Information Retrieval // arXiv. 2025. DOI: 10.48550/arXiv.2502.14822

25. Dense Hierarchical Retrieval for Open-domain Question Answering / Liu Y. та ін. Findings of the Association for Computational Linguistics: EMNLP 2021. 2021. DOI: 10.48550/arXiv.2110.15439.

26. Zhang L., Wu Y., Yang Q. Exploring the Best Practices of Query Expansion with Large Language Models // Findings of the Association for Computational Linguistics: EMNLP 2024. 2024. P. 1872–1883. DOI: 10.48550/arXiv.2401.06311

27. Pan M., Xiong W., Zhou S., Gao M., Chen J. LLM-Based Query Expansion with Gaussian Kernel Enhanced Semantic Space for Dense Retrieval // Electronics. 2025. Vol. 14, No. 9. Article 1744. DOI: 10.3390/electronics14091744.

28. Document Segmentation Matters for Retrieval-Augmented Generation / Wang Z., Gao C., Xiao C., та ін. Findings of the Association for Computational Linguistics: ACL 2025. Vienna, 2025. P. 8063–8075. DOI: 10.18653/v1/2025.findings-acl.422.

29. Nguyen H. T., Nguyen T. D., Nguyen V. H. Evaluating Chunking Strategies to Improve Retrieval-Augmented Generation with Long Clinical Documents // arXiv. 2025. arXiv:2507.09935.

30. Comparative Evaluation of Advanced Chunking for Retrieval-Augmented Generation in Large Language Models for Clinical Decision Support / Gómez-Cabello C. A., Santamaría A., Arizti-Sanz J., та ін. Bioengineering. 2025. Vol. 12, No. 11. Article 1194.

31. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks / Lewis P., Perez E., Piktus A., та ін. arXiv. 2020. DOI: 10.48550/arXiv.2005.11401.

32. Multiple Memory Systems for Enhancing the Long-term Memory of Agent / Zhang G., Wang B., Ma Y., та ін. arXiv. 2025. DOI: 10.48550/arXiv.2508.15294

33. Li K., Jing X., Jing C. Vector Storage Based Long-term Memory Research on LLM // International Journal of Advanced Network Monitoring and Controls. 2024. Vol. 9, No. 3. P. 69–79. DOI: 10.2478/ijanmc-2024-0029.

34. Huang J., Chang K. C.-C. Towards Reasoning in Large Language Models: A Survey // arXiv. 2023. DOI: 10.48550/arXiv.2212.10403.

35. MemoryBank: Enhancing Large Language Models with Long-Term Memory / Zhong W., Guo L., Gao Q., та ін. Proceedings of the AAAI Conference on Artificial Intelligence. 2024. Vol. 38, No. 17. P. 19724–19731. DOI: 10.1609/aaai.v38i17.29946.

36. Amugongo L. M. та ін. Retrieval augmented generation for large language models in healthcare: A systematic review // PLOS Digital Health. 2025. DOI: 10.1371/journal.pdig.0000877.

37. Lewis P., Perez E., Piktus A. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks // arXiv. 2020. DOI: 10.48550/arXiv.2005.11401.

38. Ford D. Introducing Contextual Retrieval URL: https://www.anthropic.com/engineering/contextual-retrieval (дата звернення: 03.12.2025).