Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики

Кафедра мультимедійних систем

Розробка інтелектуальної системи підтримки користувачів на основі Retrieval-Augmented Generation

Кваліфікаційна робота

за спеціальністю «Інженерія програмного забезпечення» 121

освітній ступінь – магістр

Керівник кваліфікаційної роботи

Ст.в. Горборуков В.В.

_____*(підпис)*

«____» _____ 2025 р.

Виконав студент Дяченко М.Є.

«____» _____ 2025 р.

Київ – 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Завідуючий кафедри мультимедійних систем,

доцент, кандидат наук

Жежерун О. П.

_____*(підпис)*

«____» _____ 2025р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студента 2 курсу факультету інформатики

Дяченка Максима Євгенійовича

Тема: Розробка інтелектуальної системи підтримки користувачів на основі Retrieval-Augmented Generation

Зміст кваліфікаційної роботи:

- Вступ
- Розділ 1. Теоретичні аспекти побудови інтелектуальних систем на основі великих мовних моделей

- Розділ 2. Аналіз предметної області та постановка задачі
- Розділ 3. Огляд та обґрунтування вибору технологій
- Розділ 4. Розробка інтелектуальної системи підтримки абітурієнтів НаУКМА
- Розділ 5. Функціональні розширення та супутні компоненти системи
- Результати та висновки
- Обмеження та перспективи подальшого розвитку
- Список літератури

Дата видачі «____» _____ 2025 р.

Керівник_____*(підпис)*

Завдання отримав _____*(підпис)*

Зміст

Вступ

У сучасному цифровому середовищі великі мовні моделі (LLM) дедалі активніше використовуються для автоматизації підтримки користувачів у різних сферах. Завдяки здатності розуміти та генерувати природну мову, такі моделі стали основою для побудови інтелектуальних чат-ботів і віртуальних асистентів. Водночас, однією з актуальних проблем залишається забезпечення точності та обґрунтованості відповідей таких систем, особливо в умовах потреби в доступі до локалізованої або спеціалізованої інформації, що не індексується загальнодоступними пошуковими системами.

Одним з перспективних підходів до вирішення цієї проблеми є Retrieval-Augmented Generation (RAG), який поєднує можливості генеративних моделей із механізмами пошуку релевантної інформації з зовнішніх джерел знань. Такий підхід дозволяє створювати більш достовірні відповіді, що особливо важливо у сфері, де користувачам необхідно надавати точну й актуальну інформацію.

Метою цього дослідження є побудова архітектури та розробка системи підтримки з використанням підходу RAG, що включатиме в себе створення та наповнення векторного індексу, пошук релевантного контексту та генерація відповідей та користувацький інтерфейс. Як джерело даних, планується використати офіційний сайт вступної кампанії НаУКМА: https://vstup.ukma.edu.ua/. Розроблена система повинна бути гнучкою, тобто працювати з іншими веб-джерелами без внесення значних змін до коду.

У рамках дослідження ставиться низка завдань, зокрема передбачається здійснити теоретичний аналіз переваг і обмежень LLM у контексті інформаційної підтримки користувачів, а також дослідити, яким чином застосування RAG дозволяє компенсувати типові недоліки генеративних систем. При реалізації системи необхідно врахувати особливості структури веб-джерел та їх метаданих для побудови

векторного індексу та реалізації пошуку по ньому. Планується здійснити порівняльний аналіз якості пошуку з урахуванням різних стратегій пошуку та моделей генерації векторних вкладень, з подальшим тестуванням генерації відповідей.

Перелік умовних позначень та абревіатур

- LLM – Large Language Model, велика мовна модель.
- RAG – Retrieval-Augmented Generation, генерація з доповненням через пошук.
- API – Application Programming Interface, прикладний програмний інтерфейс.
- ANN – Approximate Nearest Neighbor, наближений пошук найближчого сусіда.
- H@K – Hits@K, метрика наявності правильної відповіді серед K найкращих результатів.
- URL – Unified Resource Locator, уніфікований локатор ресурсів.
- HTML – HyperText Markup Language, мова розмітки гіпертексту.
- DOM – Document Object Model, об’єктна модель документу.
- SEO – Search Engine Optimization, оптимізація пошукових систем.
- CSV – Comma-Separated Values, формат файлу з розділом даних через кому.

Розділ 1. Теоретичні аспекти побудови інтелектуальних систем на основі великих мовних моделей

1.1 Переваги використання LLM

Велика мовна модель (або LLM від англ. large language model) - це модель мови, що складається з нейронної мережі з багатьма параметрами (від десятків мільйонів до мільярдів), навчених на великій кількості немаркованого тексту за допомогою самокерованого або напівкерованого навчання[1]. Вони побудовані на архітектурі типу трансформер, і здатні обробляти, аналізувати та генерувати тексти природною мовою з високим рівнем узгодженості, стилістичної відповідності та контекстної релевантності. Застосування LLM у сфері побудови інтелектуальних систем підтримки користувачів відкриває нові горизонти в автоматизації інформаційного обслуговування, особливо в контексті задач, які вимагають інтерпретації відкритих, неконтрольованих запитів.

1.1.1 Моделювання та розуміння людської мови

Однією з ключових переваг LLM є їх здатність універсально моделювати людську мову без жорсткого програмування правил або попереднього визначення сценаріїв. Це дозволяє створювати системи, які можуть відповідати на запитання, перефразовувати інформацію, наводити приклади та узагальнювати тексти в рамках єдиної моделі, без необхідності спеціалізованої логіки для кожної окремої дії. Таким чином, LLM слугують гнучкою та масштабованою основою для побудови систем підтримки, які можуть адаптуватися до нових запитів без ручного втручання. LLM не оперують знаннями в класичному семантичному сенсі, а працюють із ймовірнісними розподілами послідовностей токенів. Проте завдяки навчанню на великих обсягах тексту вони демонструють поведінку, що наближається до розуміння.

1.1.2 Стійкість до помилок у запитах

Хоча наявність помилок у текстових запитах може призводити до зниження точності відповідей, великі мовні моделі демонструють стійкість до них. Це відрізняє їх від rule-based систем, де порушення очікуваної структури запиту часто призводить до повної відмови від генерації коректної відповіді або до помилки. У випадку з LLM відповідь, як правило, залишається частково релевантною, навіть якщо запит сформульований з помилками. В цьому контексті цікавим є дослідження впливу типографічних помилок на якість відповідей LLM від Національного університету Сінгапуру. Використовуючи змагальний алгоритм атаки типографічними помилками (Adversarial Typo Attack), було виявлено що типографічні помилки в запитах погіршують якість відповідей моделей з відкритим кодом на 5-15%, в залежності від типу задачі. Однак, цей показник знизився до 3-4% з використанням моделі GPT-4[2]. Така стійкість до помилок є дуже корисною у консультаційних сервісах, де користувачі можуть формувати запити у неформальній манері та допускаючи помилки.

1.2 Виклики використання LLM у практичних задачах

Попри значний прогрес у розвитку великих мовних моделей, їх використання в реальних задачах, зокрема у сфері підтримки, супроводжується низкою практичних обмежень. У цій частині роботи розглядаються ключові виклики, які виникають під час інтеграції LLM у прикладні системи.

1.2.1 Обмеженість знань і проблема актуальності

Хоча LLM мають доступ до величезного обсягу інформації, ці знання є статичними - обмеженими моментом завершення навчання моделі, описаною у спільноті розробників терміном «обрив знань (від англ. knowledge cutoff)». Наприклад, для моделі GPT-4o зазначеною датою обриву знань є 1 жовтня 2023 року[3]. Крім обмеження за часом, LLM можуть не охоплювати спеціалізовані теми,

що не потрапили до навчального корпусу даних. Це створює суттєву проблему у динамічних ти вузькоспеціалізованих доменах, таких як вступні кампанії. У таких випадках модель або взагалі не відповідає, або надає застарілу чи неправдиву інформацію.

Щоб частково компенсувати це обмеження, застосовується підхід доповнення генерації пошуком (Retrieval-Augmented Generation, RAG), за якого модель отримує зовнішній контекст із бази даних або пошукового індексу перед генерацією відповіді. Однак і цей підхід має свої виклики: якість результату залежить від якості індексації, релевантності результату пошуку, коректного формулювання запиту для пошуку тощо. Крім того, зовнішній контекст не завжди може повністю компенсувати брак знань, якщо питання складне або містить кілька підтекстів.

1.2.2 Галюцинації

У контексті штучного інтелекту, галюцинацією називають генерацію штучним інтелектом відповіді на запит, яка містить неправдиву або оманливу інформацію, подану як факт[4]. Основними з існуючих методів вирішення проблеми галюцинацій є доповнення пошуком (RAG), самовдосконалення моделей через ланцюг думок (Chain of Thought, скорочено CoT), та інженерія запитів (prompt engineering).[5]

1.2.3 Обмежене контекстне вікно

Більшість LLM мають фіксований ліміт на обсяг інформації, який може бути оброблений в одному запиті (наприклад, 8000 або 32000 токенів). Це може обмежувати глибину аналізу великих документів або довготривалих діалогів. Хоча нові моделі пропонують збільшені контекстні вікна, це створює накладні витрати при використанні таких моделей через більшу кількість ресурсів, потрібних на обрахунки.

1.2.4 Відсутність постійної пам’яті

Моделі не зберігають стан попередньої взаємодії, якщо його явно не передано в контексті запиту. Це ускладнює побудову персоналізованих систем, які повинні пам’ятати історію запитів користувача. Частково ця проблема вирішується через додаткові механізми пам’яті на рівні архітектури додатку.

1.2.5 Вартість та продуктивність

Запити до комерційних моделей таких компаній, як Google, OpenAI, Grok, Anthropic та інших, супроводжуються витратами на генерацію згідно тарифних планів на обрану мовну модель. У випадку частих звернень або великої кількості користувачів це може стати стримуючим фактором для масштабування.

1.3 Роль векторних баз даних у побудові інформаційного контексту

Як зазначалося у попередньому підпункті, одним із найефективніших підходів для подолання обмеженості знань великих мовних моделей є Retrieval-Augmented Generation (RAG) - метод, який передбачає попередній пошук релевантної інформації з зовнішнього джерела, що потім подається LLM як контекст для формування відповіді. Якість роботи такого підходу критично залежить від здатності системи точно і швидко знаходити відповідні фрагменти знань на основі запитів, сформульованих природною мовою.

Класичні підходи до пошуку інформації, зокрема текстовий пошук, є малоефективними в контексті інтеграції з великими мовними моделями. У масштабах десятків або сотень тисяч текстових фрагментів така стратегія стає ресурсоємною та повільною. Натомість, для обробки тексту природньою мовою в системах штучого інтелекту та машинного навчання використовують векторні вкладення. Вкладання слів (англ. word embedding) - це загальна назва низки методик мовного моделювання та навчання ознак в обробці природної мови (ОПМ), в яких слова або фрази зі словника відображують у вектори дійсних чисел [6]. Ідея цього представлення полягає

в тому, щоб представити запити і документи (або їхні фрагменти) у вигляді векторів у багатовимірному просторі, де семантично схожі тексти розташовані близько один до одного. Векторні вкладення генеруються за допомогою спеціальних моделей, що навчені знаходити семантичні подібності між текстами.

Векторні бази даних використовуються для зберігання векторів разом з іншими елементами даних. Векторні бази даних зазвичай реалізовують один чи декілька алгоритмів наближено найближчих сусідів (ННС, англ. Approximate Nearest Neighbor, ANN), що дає можливість пошуку базою даних за допомогою вектора запиту, знаходячи найближчі відповідні записи бази даних.[7] Алгоритм пошуку найближчих сусідів відповідає за знаходження векторів у базі, які знаходяться на мінімальній відстані до вектора запиту. Ці алгоритми ґрунтуються на математичній концепції метричних просторів - де знаходяться точки даних і визначені відстані між ними. Для їх обрахунку використовуються такі загальні функції, як Евклідова відстань або косинус подібності.[8]

$$\frac{\sum_{i=1}^{n} A_i AB_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Формула 1. Косинус подібності

$$\sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Формула 2. Евклідова відстань

Більшість векторних баз даних реалізують спеціалізовані структури даних та алгоритми, які дозволяють зменшити час виконання запиту з незначною втратою точності. Сучасні векторні бази даних підтримують масштабування, горизонтальну

розподіленість, обробку великих обсягів даних у потоковому режимі, а також можливість комбінувати семантичний пошук із фільтрами, що відкриває широкі можливості для побудови інтелектуальних систем. Саме тому такі бази стають ключовим компонентом у архітектурах, побудованих навколо великих мовних моделей, зокрема при реалізації Retrieval-Augmented Generation.

Висновок до Розділу 1

У першому розділі було проаналізовано ключові переваги великих мовних моделей для створення систем підтримки - гнучкість, генерація природної мови, адаптивність - а також основні виклики: обмеженість знань, галюцинації, відсутність пам’яті, контекстні та обчислювальні обмеження. Окремо акцентовано на проблемі актуальності знань, що зумовлює використання зовнішніх джерел, зокрема в межах Retrieval-Augmented Generation. Представлені положення є підґрунтям для реалізації системи, що розглядатиметься у наступних розділах.

Розділ 2 Аналіз предметної області та постановка задачі

Розробка інтелектуальної системи підтримки абітурієнтів вимагає ґрунтовного розуміння предметної області - зокрема специфіки вступної кампанії як інформаційного та організаційного процесу. У цьому розділі розглянуто ключові особливості взаємодії абітурієнтів із закладом вищої освіти, структуру наявної інформації, а також типові запити, з якими звертаються користувачі впродовж вступного періоду. Аналіз цих аспектів дозволяє виявити основні потреби цільової аудиторії та сформувати вимоги до функціональної системи, на основі яких відбуватиметься подальша розробка.

2.1 Вступна кампанія як процес: структура, інформаційні потреби, типові запити

Вступна кампанія до закладів вищої освіти є складним багатокомпонентним процесом, що поєднує нормативно-правові, адміністративні та інформаційні аспекти. Її особливістю є висока інтенсивність, чітка прив’язка до календарних строків та потреба в оперативному інформуванні великої кількості зацікавлених осіб -здобувачів освіти, їхніх батьків, а також представників навчальних закладів.

У структурі вступної кампанії можна виокремити кілька основних етапів: реєстрація та подача документів, вибір спеціальностей і подання заяв, конкурсний відбір, оприлюднення результатів та зарахування. Кожен із цих етапів супроводжується інформаційними запитами від абітурієнтів, які можуть стосуватися як загальних правил (умови вступу, перелік спеціальностей, строки подачі документів), так і індивідуальних обставин (пільги, квоти, документи іноземного зразка тощо).

До типових інформаційних запитів, що виникають у вступників, належать: «Які спеціальності є у вашому університеті?», «Який прохідний бал на бюджет?», «Як можна перевестися з іншого університету?», «Які пільги враховуються при вступі?»,

«Коли буде оголошено результати конкурсу?» тощо. Значна частина таких запитів є повторюваною та стандартизованою, однак відповіді на них часто потребують врахування актуальних умов, специфіки факультету та спеціальності або особистих даних вступника.

Враховуючи обмеженість людських ресурсів та зростання кількості звернень у період активної фази вступу, особливо актуальною є задача автоматизації відповіді на запити абітурієнтів. Це вимагає створення системи, яка не лише зберігатиме структуровану інформацію про вступ, а й дозволить взаємодіяти з користувачем у зручній для нього формі - природною мовою, з урахуванням контексту та мінімальною потребою в уточненні формулювань.

2.2 Аналіз користувацьких сценаріїв для абітурієнтів

Розробка інтелектуальної системи підтримки передбачає чітке розуміння того, як саме абітурієнти взаємодіють з інформаційними ресурсами під час вступної кампанії. Для цього необхідно проаналізувати типові користувацькі сценарії - повторювані шаблони поведінки, які характеризують способи отримання та використання інформації майбутніми студентами. Основні сценарії можна поділити на кілька категорій:

- Пошук загальної інформації про університет - ознайомлення з історією, структурою, цінностями навчального закладу, перегляд переліку спеціальностей та освітніх програм.
- Орієнтація у правилах прийому - з’ясування вимог до абітурієнтів, умов подачі документів, термінів реєстрації, процедур зарахування, використання пільг та квот.
- Порівняння спеціальностей і вибір пріоритетів - аналіз конкурсних балів попередніх років, кількості місць на, вартості навчання за контрактом.

- Уточнення індивідуальних умов - ситуації, пов’язані з особливими обставинами абітурієнта: іноземне громадянство, можливість навчання за обміном, спеціальні умови вступу, тощо.

Окрім категорій питань, варто розуміти що деякі слова чи фрази в запиті можуть нести високу семантичну вагу, оскільки визначають контекст та конкретний елемент вступної кампанії. До таких визначних лексичних елементів належать:

Освітній рівень

- *бакалаврат*, *магістратура*, *аспірантура*- впливають на все: вимоги, строки, перелік документів, конкурсний бал.

Цільова аудиторія

- *іноземці*, *пільговики*, *військовослужбовці*, *переселенці* - фільтрують інформацію по спеціальних умовах вступу.

Форма навчання

- *денна*, *заочна*, *контракт*, *бюджет* - змінює перелік можливих програм і пріоритетів.

Етап вступу

- *подати заяву*, *пройти співбесіду*, *подати оригінали*, *зарахування* - визначає, про який момент процесу йдеться.

У кожній з наведених категорій питань абітурієнт стикається з великим обсягом інформації, яка може бути представлена у вигляді складних нормативних документів, сторінок сайту або таблиць, що потребує годин пошуку для знаходження усіх відповідей. Водночас можливість отримати стислі, чіткі та персоналізовані відповіді може значно спростити вступний процес.

2.3 Вимоги до системи підтримки абітурієнтів, постановка задачі

На основі аналізу предметної області, типових сценаріїв використання та характеру запитів, які формулюють абітурієнти, можна визначити ключові вимоги до функціональності майбутньої системи. Ці вимоги охоплюють як функціональні аспекти взаємодії з користувачем, так і нефункціональні обмеження, що стосуються якості, швидкодії та достовірності відповідей.

Функціональні вимоги:

1. **Обробка запитів природною мовою.** Система повинна приймати запити користувачів у довільній формі, сформульовані українською чи іноземними мовами.
2. **Категоризація запитів.** Модель має розпізнавати лексичні одиниці, що задають контекст (наприклад, бакалаврат, магістратура, бюджет, контракт), та враховувати їх при формуванні відповіді.
3. **Пошук релевантної інформації.** Система має здійснювати пошук у зовнішній базі знань отриманій з сайту vstup.ukma.edu.ua.
4. **Генерація відповіді.** На основі знайденого контексту система повинна формувати стислі, точні та зрозумілі відповіді природною мовою.
5. **Надання джерела.** Кожна відповідь має містити посилання на джерело інформації, яке було використано при генерації відповіді.
6. **Можливість подальших уточнень.** Система повинна підтримувати уточнюючі запити в межах одного діалогу.
7. **Наявність адміністративного інтерфейсу для керування векторним сховищем.** Система повинна надавати можливість виконання CRUD-операцій (створення, читання, оновлення, видалення) над записами у базі знань через веб-інтерфейс.

Нефункціональні вимоги:

1. **Час відповіді.** Середній час формування відповіді не повинен перевищувати 1 хвилину.
2. **Достовірність.** Відповіді мають відповідати актуальній інформації вступної кампанії поточного року.
3. **Гнучкість оновлення бази знань.** Система має дозволяти регулярне оновлення інформації без необхідності перенавчання моделі.
4. **Стійкість до некоректно сформульованих запитів.** Система повинна коректно обробляти запити з граматичними помилками або розмовною лексикою.
5. **Простота використання.** Інтерфейс має бути зрозумілим для широкого кола користувачів без технічної підготовки.

Висновок до Розділу 2

У цьому розділі було проведено аналіз предметної області вступної кампанії, описано основні інформаційні потреби абітурієнтів і типові сценарії їхньої взаємодії з системою підтримки. Було визначено ключові категорії запитів та лексичні маркери, що впливають на контекст пошуку і генерацію відповідей. На основі проведеного аналізу сформульовано функціональні й нефункціональні вимоги до майбутньої системи, які слугуватимуть основою для вибору архітектури, технологічного стеку та підходів до реалізації.

Розділ 3. Огляд та обґрунтування вибору технологій

У межах розробки системи було обрано технологічний стек, який забезпечує оптимальне поєднання простоти розгортання, широкої підтримки інструментів для роботи з великими мовними моделями та мінімізації технічних ризиків.

3.1. Мова програмування: Python

Python обрано як мову розробки через її широке застосування у сфері штучного інтелекту та машинного навчання. Завдяки великій кількості бібліотек і фреймворків для обробки природної мови, побудови нейронних мереж, роботи з даними та взаємодії з векторними базами даних, Python фактично став стандартом для створення систем із використанням машинного навчання та штучного інтелекту. Підтримка бібліотек для інтеграції з обраною LLM та векторною базою даних, значно спрощує процес розробки рішень на базі сучасних LLM.

Крім того, Python вирізняється простотою синтаксису та швидкістю розробки, що дозволяє ефективно реалізувати усі необхідні компоненти системи в рамках єдиного технологічного середовища. Це зменшує кількість точок потенційних збоїв між різними технологіями, спрощує обслуговування коду та прискорює інтеграцію окремих модулів - зокрема веб-сервісу, модуля генерації вкладень та системи пошуку на основі векторних представлень.

3.2. Моделі штучного інтелекту: моделі від OpenAI, bge-m3

Для генерації вкладень текстових фрагментів та формування відповідей на запити користувачів було обрано підхід, заснований на використанні готових API великих мовних моделей. Від розгортання локальних моделей було свідомо відмовлено через відсутність значних обчислювальних ресурсів, необхідних для розгортання LLM.

Серед доступних API-рішень було обрано сервіси компанії OpenAI. Такий вибір зумовлений високою стабільністю роботи, якістю генерації відповідей для україномовних запитів, а також зручною інтеграцією API, зокрема через бібліотеку «openai» для Python. Згідно з рейтингом незалежного аналітичного агентства Artificial Analysis [9], модель «gpt-o4» має найвищий індекс інтелекту (середнє значення метрик оцінювання LLM на різних масивах задач), достатнє для поставленої задачі контекстне вікно в 200000 символів, та помірну ціну.

У рамках рішення для генерації векторних представлень тексту обрано кілька моделей для тестування. Модель «text-embedding-3-small» від OpenAI забезпечує хороше співвідношення між якістю вкладень, швидкістю обробки запитів та їх вартістю. Для порівняння також використано модель «text-embedding-3-large», яка забезпечує кращу якість представлення тексту, проте потребує більших обчислювальних ресурсів. Як альтернативне рішення, протестовано open-source модель «bge-m3» (сімейство моделей BAAI), яке є багатомовною моделлю з підтримкою, що показала конкурентні результати у порівнянні з моделями OpenAI при векторному пошуку [999].

Таким чином, використання OpenAI API для генерації відповідей забезпечує баланс між якістю генерації, продуктивністю обробки запитів та швидкістю розробки. Вибір моделі для векторних вкладень потребує подальшого тестування, що буде описано у наступному розділі.

3.3. Векторна база даних: ChromaDB

Для організації пошуку релевантної інформації на основі вкладень було обрано векторну базу даних ChromaDB. Основним критерієм вибору стало поєднання високої продуктивності та наявність інтеграції з Python.

ChromaDB підтримує як локальне (in-memory), так і віддалене (хмарне) зберігання векторних даних завдяки сервісу Chroma Cloud. У межах даного проєкту

було прийнято рішення використовувати локальний режим роботи в оперативній пам’яті. Такий підхід обґрунтовується кількома факторами.

По-перше, розміщення векторів безпосередньо в оперативній пам'яті істотно знижує затримки при виконанні пошуку за наближеними найближчими сусідами (ANN). Завдяки відсутності потреби у мережевій взаємодії із зовнішніми серверами мінімізується час відповіді на запити, що є критично важливим для забезпечення швидкої реакції системи у режимі реального часу.

По-друге, використання in-memory режиму спрощує розгортання рішення, оскільки відсутня необхідність налаштовувати окремі мережеві підключення та системи автентифікації.

Враховуючи відносно невеликий обсяг початкового набору даних і прогнозовані навантаження на систему, локальне in-memory зберігання є оптимальним рішенням для початкової версії системи підтримки абітурієнтів. У разі необхідності масштабування в майбутньому передбачена можливість міграції даних у хмарну інфраструктуру ChromaDB.

3.4. Збір даних з веб-ресурсів: Selenium

Одним із важливих етапів розробки системи стало забезпечення якісного збору текстових даних із вебсайту НаУКМА. Оскільки частина вмісту завантажується динамічно за допомогою JavaScript, для гарантованого отримання повного текстового контенту було обрано використання Selenium.

Selenium -дозволяє автоматизовано керувати веб-браузером (у даному проєкті - Google Chrome у headless-режимі, тобто без графічного інтерфейсу), що дає змогу отримувати текст після повного виконання сценаріїв на стороні клієнта. На відміну від статичних методів парсингу, які працюють лише з початковим HTML-кодом (наприклад, BeautifulSoup), використання Selenium гарантує доступ до реального

вмісту DOM після завантаження сторінки, що відповідає відображенню у браузері користувача.

Ключовими перевагами використання Selenium у межах даного проєкту є:

- можливість обробки динамічного вмісту сторінок без втрат текстових даних;
- забезпечення відповідності з тим, що бачить кінцевий користувач у браузері;
- інтеграція із середовищем Python.

Попри суттєві переваги у повноті отримання даних, використання Selenium має певні недоліки. Основним із них є відносно низька швидкість роботи у порівнянні зі статичним парсингом HTML, оскільки кожен запит вимагає повноцінного завантаження сторінки у браузері та виконання всіх супутніх скриптів. Проте у межах даного проєкту повільніша робота інструменту не є критичною, оскільки процес збору даних є одноразовим етапом для попередньої генерації бази знань і не впливає на швидкодію фінальної системи на етапі обробки користувацьких запитів.

3.5. Flask для побудови вебсервісу

Для розгортання API-сервісу, який забезпечує взаємодію користувача із системою, було обрано вебфреймворк Flask. Цей інструмент є одним із найпопулярніших мінімалістичних рішень у середовищі Python для створення вебзастосунків і сервісів.

Вибір Flask обумовлено його простотою, легкістю інтеграції та гнучкістю конфігурації. Фреймворк дозволяє швидко створити повноцінний серверний застосунок із підтримкою обробки HTTP-запитів, маршрутизації, а також можливістю обслуговування статичних файлів (наприклад, HTML-сторінок інтерфейсу користувача). Завдяки мінімалістичному підходу Flask не вимагає складної структури проєкту або додаткових шарів абстракції, що особливо важливо для невеликих систем із чітко окресленим обсягом функціональності.

Важливою перевагою використання Flask є можливість розробки в межах одного технологічного стеку, без необхідності залучення окремого серверу чи зовнішнього фреймворку для побудови фронтенду. У межах даного проєкту було реалізовано статичний HTML-інтерфейс і набір API-ендпоінтів для обробки запитів користувачів та адміністративних запитів, що повністю покриває функціональні потреби системи.

Висновок до Розділу 3

У цьому розділі було обґрунтовано вибір технологічного стеку, який забезпечує оптимальне поєднання простоти розгортання, високої продуктивності та підтримки сучасних інструментів для роботи з великими мовними моделями. Обрана мова програмування Python, моделі OpenAI для генерації вкладень і відповідей, векторна база даних ChromaDB для ефективного пошуку, бібліотека Selenium для збору динамічного вебконтенту та фреймворк Flask для реалізації вебсервісу створюють єдину інтегровану архітектуру, що відповідає вимогам проєкту. Сформований стек технологій дозволяє забезпечити якісне виконання поставлених завдань при збереженні гнучкості для подальшого масштабування або розвитку системи.

Розділ 4. Розробка інтелектуальної системи підтримки абітурієнтів НаУКМА

У цьому розділі описано процес розробки інтелектуальної системи підтримки абітурієнтів, реалізованої на основі LLM та векторного пошуку. Детально розглянуто архітектуру застосунку, підходи до збору, обробки даних та генерації відповідей. Описано реалізацію окремих функціональних компонентів та вирішення проблем у роботі з великими мовними моделями.

4.1 Архітектура системи

Архітектура системи побудована за принципами Retrieval-Augmented Generation (RAG) і складається з кількох компонентів, які забезпечують повний цикл обробки запиту - від збору даних до формування відповіді. На діаграмі зображено логічну структуру взаємодії елементів системи:

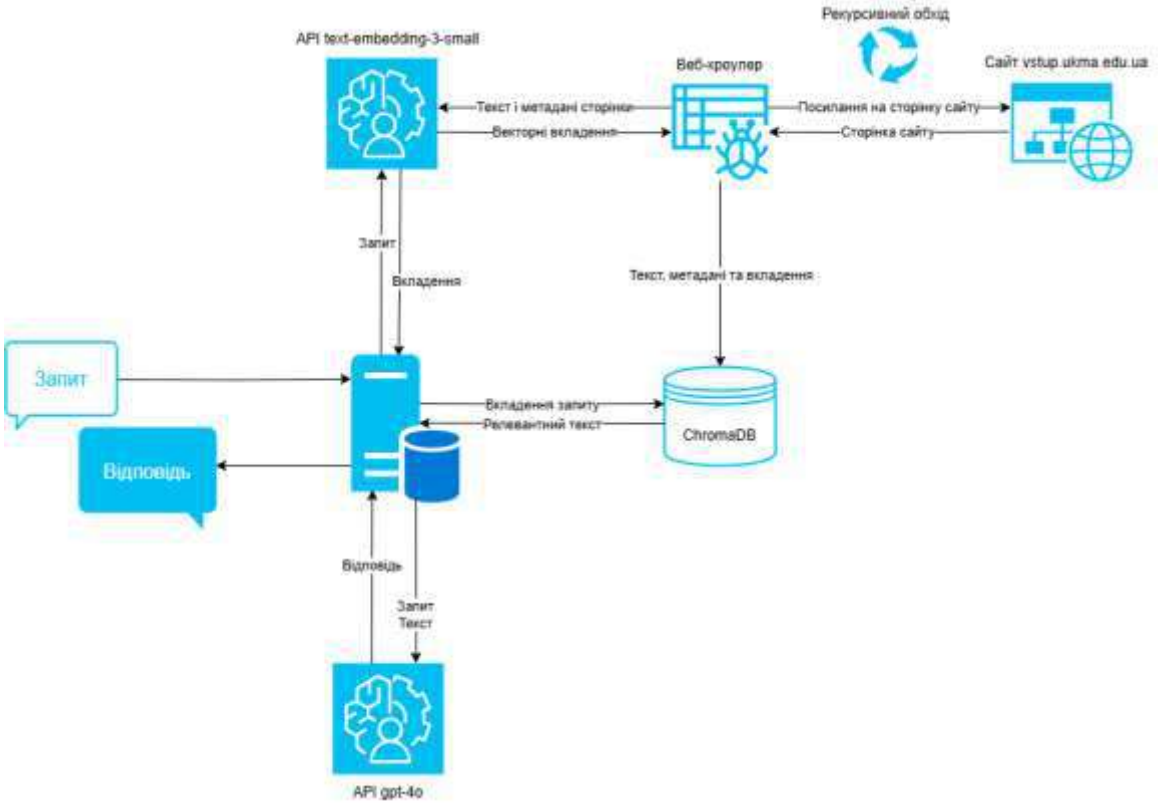

Рисунок 1. Діаграма архітектури застосунку

Ключові компоненти системи:

- Сервер додатку - центральний координуючий компонент, який обробляє запити користувачів, взаємодіє з API OpenAI та базою даних для генерації відповідей та зміни наповнення бази даних.
- Веб-кроулер - модуль, що здійснює рекурсивний обхід сайту vstup.ukma.edu.ua, завантажує вміст сторінок, очищає їх від службової інформації (меню, скрипти тощо) та передає очищений текст для подальшої обробки.
- Сайт vstup.ukma.edu.ua - основне джерело контенту для побудови бази знань. Містить релевантну інформацію, з якою може взаємодіяти абітурієнт.
- ChromaDB - використовується для зберігання векторних представлень тексту (embedding) разом із метаданими. Забезпечує швидкий пошук тексту до запиту користувача за допомогою механізму наближених найближчих сусідів (ANN).
- API моделей OpenAI - включає дві зовнішні моделі: «text-embedding-3-small» для генерації векторних вкладень тексту та «gpt-4o» для формування відповідей на основі релевантного контексту.

На основі цих компонентів можна описати два основні сценарії використання системи – наповнення бази даних та генерація відповіді на запит користувача:

- Наповнення бази даних: веб-кроулер виконує обхід сайту, завантажує вміст сторінок і метадані. Після цього, генеруються вкладення з використанням моделі «text-embedding-3-small». Текст сторінок разом з відповідними метаданими і вкладеннями зберігаються у ChromaDB.
- Генерація відповіді на запит користувача: користувач формулює запит природною мовою. Сервер генерує вектор вкладення цього запиту за допомогою моделі «text-embedding-3-small» і виконує пошук релевантних текстових фрагментів у ChromaDB. Результат пошуку разом із початковим

запитом передаються до моделі «gpt-4o», що формує фінальну відповідь, яка відображається в користувацькому інтерфейсі.

Також, наведено діаграму яка спрощено ілюструє організацію коду застосунку:

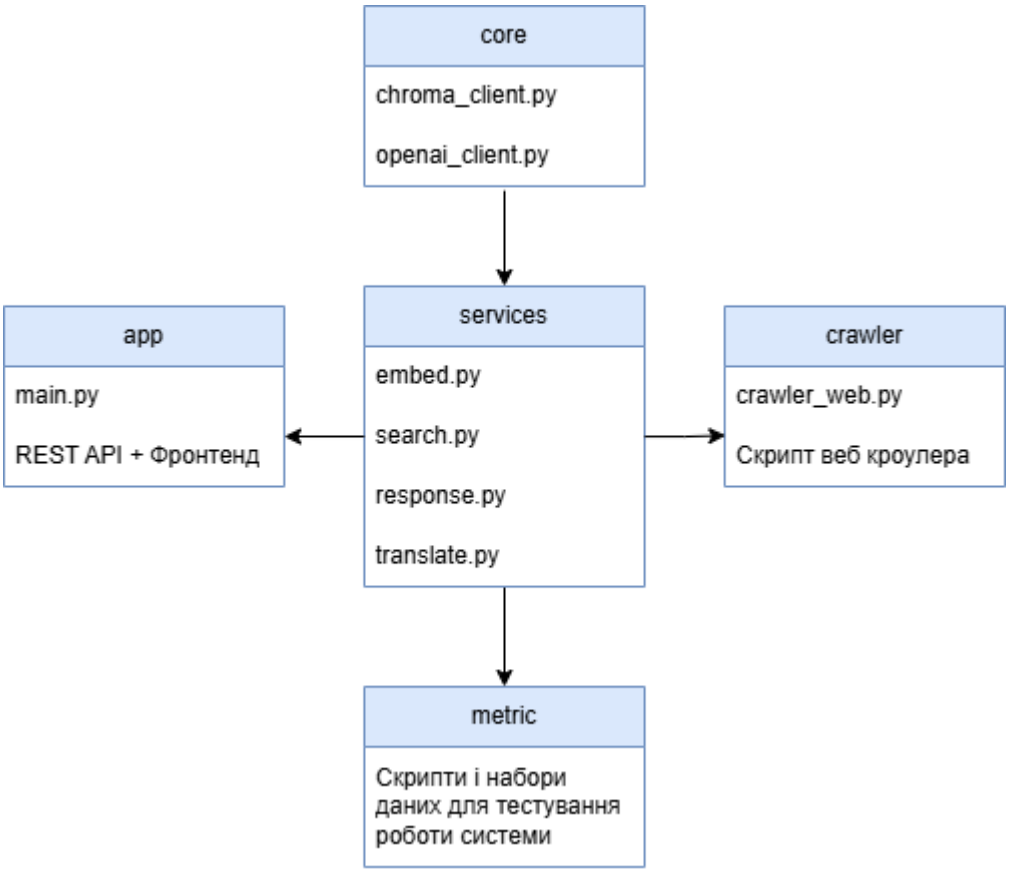

Рисунок 2. Функціонально-модульна арітектура

Папка «core» містить клієнти для взаємодії з зовнішніми сервісами: ChromaDB для векторного зберігання та моделі LLM та векторних вкладень для пошуку та генерації відповідей. У папці «services» розміщені компоненти, які відповідають за основну функціональність системи: генерація вкладень (embed.py), пошук (search.py), побудова відповідей (response.py) та переклад (translate.py). Папка «crawler» містить скрип, що відповідає за рекурсивний обхід сайту з використанням Selenium. Папка «app» та файл «main.py» - є точкою входу до програми, реалізує API та містить сторінки з користувацьким інтерфейсом. Папка «metric» містить допоможні скрипти

та тестові дані, що використовуються для оцінювання якості пошуку та параметрів токенізації тексту.

4.2 Отримання і попередня обробка текстових даних з веб-джерел

Для побудови бази знань системи використовується вміст офіційного сайту вступної кампанії НаУКМА «vstup.ukma.edu.ua». Завдяки наявності внутрішніх посилань, стає можливим автоматизований обхід усього сайту, починаючи з однієї стартової адреси, шляхом послідовного переходу за виявленими посиланнями. Перевагою використання існуючого веб-ресурсу як основного джерела знань для системи є можливість автоматизованого збору вже оприлюдненої та структурованої інформації без потреби в ручному наповненні бази знань або створенні окремого контенту. Нижче наведено псевдокод рекурсивного алгоритму веб-кроулінгу:

```
def crawl(url, visited):
    if url in visited:
        return
    visited.add(url)

    page = open_page(url)
    page_text, metadata = extract_text_and_metadata(page)
    embed_page(url, page_text, metadata)

    links = get_links_in_page(page)
    for link in links:
        if is_internal(link):
            crawl(link, visited)
```

Рисунок 3. Псевдокод веб-кроулеру

Алгоритм починає свою роботу з початкової сторінки, отримуючи її посилання (параметр в псевдокоді **url**) та масив відвіданих сторінок (**visited**). Функція виконує перевірку на наявність посилання в масиві раніше відвіданих, що є умовою завершення виконання. Така перевірка є необхідною, оскільки посилання на одну

сторінку можуть містити різні сторінки, що допомагає уникати дублювання при збереженні інформації. Далі, сторінка відкривається в Headless-браузері за домогою Selenium (функція «**open**_page»), що дозволяє отримати текст та метадані сторінки (функція «**extract_text_and_metadata»**). Отримані текст та метадані зберігаються у базі даних (функція «**embed_page»**). Після цього, здійснюється пошук посилань на сторінці. За умови, що знайдене посилання є внутрішнім, відбувається рекурсивний виклик функції crawl з посиланням.

У процесі збору текстових даних із веб-сторінок важливо відфільтрувати елементи, які не міститимуть змістового навантаження для використання у мовній моделі. Для збору релевантної текстової інформації зі сторінки можна використати «document.body.innerText», що дозволяє зчитувати лише той вміст, який відображається у браузері, та є ефективним фільтом, що прибирає скрипти, стилі та прихований текст. Однак, навіть серед видимого контенту існує багато повторюваних або другорядних елементів - меню навігації, бічні панелі, елементи інтерфейсу що дублюються на кожній сторінці. Щоб уникнути включення цих елементів у фінальний текст, вони цілеспрямовано видаляються зі структури DOM до зчитування innerText. На практиці це досягається через вилучення елементів із такими тегами, як <header>, <footer> тощо. Після попереднього фільтрування HTML, також було видалено знаки пунктуації з метою запобігання шуму у даних.

З метою підвищення ефективності ранжування результатів під час пошуку векторних вкладень, запропоновано використовувати метадані сторінки в комбінації з основним текстовим контентом, що отримується з innerText. У даному контексті, метадані – це структурні елементи сторінки, розміщені в тегу <head> HTML-документу, що використовуються пошуковими системами для індексації та кращого розуміння вмісту сторінки. Згідно з рекомендаціями Google SEO Starter Guide [10], к лючову роль відіграють теги <title> і <meta name="description">, які формують заголовок і опис сторінки у видачі. Також велике значення мають Open Graph-

метатеги (наприклад, <meta property="og:title">, <meta property="og:description">), що використовуються для відображення попереднього перегляду при поширенні посилань у соцмережах. Саме ці метадані містять узагальнений опис вмісту сторінки й ключові слова, що можуть зустрічатися у запитах користувачів, тому є цінним джерелом семантичної інформації, яку доцільно враховувати під час побудови векторної бази знань. Для векторизації метаданих, запропоновано використовувати конкатенацію різних атрибутів у одну стрічку для генерації вектору метаданих. Перераховані вище теги є стандартом SEO-оптимізації, та зустрічаються на багатьох веб-ресурсах, зокрема на сайті вступу НаУКМА, тож можуть бути використані для ранжування результатів векторного пошуку:

```
<meta property="og:type" content="article">
<meta property="og:title" content="Інтенсив від Підготовчого відділення НаУК
МА">
<meta property="og:description" content="Підготовче відділення Національного
університету «Києво-Могилянська академія» оголошує набір на інтенсивні підго
товчі курси з предметів, що входять до...">
<meta property="og:image" content="https://storage.smart.ukma.edu.ua/storag
e/NEWS/121/logo">
```

Рисунок 4. Приклад SEO-тегів на сторінці сайту вступної кампанії

У процесі аналізу вмісту сторінок було виявлено, що багато з них містять у тексті ключові слова, які позначають рівень освіти, зокрема «бакалавр», «магістр», «аспірант», що мають високе семантичне значення, оскільки суттєво впливають на контекст запиту та вибір релевантної відповіді. Тому такі маркери доцільно автоматично виокремлювати під час обробки сторінки та зберігати у вигляді окремого поля для ключових слів в метаданих. Це дозволяє покращити подальшу категоризацію документів і підвищити точність пошуку за рахунок додаткового фільтрування або повторного ранжування результатів.

На наведеному нижче рисунку показано, які частини HTML-сторінки враховуються при формуванні векторного представлення документа. Видаляються структурні теги, що не містять інформативного навантаження, зберігаються ключові

текстові фрагменти з основного вмісту та метадані. Окремо виділяються та включаються до метаданих ключові слова, що допомагають категоризувати документ при запиті.

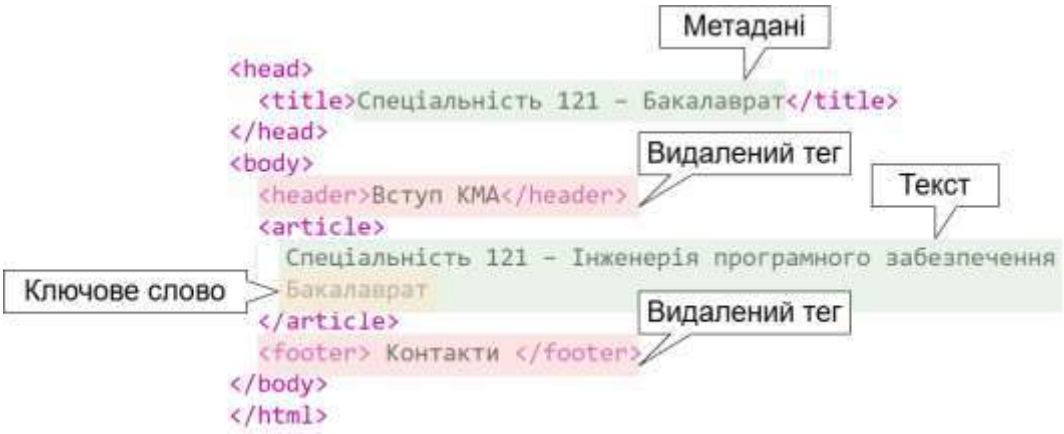

Рисунок 5. Семантична декомпозиція сторінки

Окремої уваги потребує адаптація веб-кроулінгу до динамічної структури сайту. Наприклад, на сторінці https://vstup.ukma.edu.ua/bachelor/entry-steps зміна типу вступу (бюджет/контракт/пільговик) відбувається не через перехід на нову сторінку, а шляхом динамічного оновлення контенту після натискання на відповідні кнопки. Така реалізація створює складність для класичного парсингу, оскільки необхідна інформація не завантажується при відкритті сторінки. У рамках реалізації веб-кроулера за допомогою Selenium була передбачена можливість емуляції кліків по цим елементам, що дозволяє отримати весь релевантний вміст, включно з критично важливими даними, наприклад, строками подачі документів.

Ще одним джерелом потенційно цінної інформації є PDF-документи, які публікуються на сайті. Як правило, дані файли містять нормативні документи та приклади форм для друку. На поточному етапі ці документи не інтегруються в базу знань напряму, однак можливість їх аналізу розглядається як потенційне покращення системи. Як тимчасовий компроміс, система може вказувати на сторінку, що містить посилання на необхідний PDF-файл.

4.3 Генерація вкладень та побудова векторного індексу

Побудова системи семантичного пошуку в межах архітектури RAG передбачає створення векторного індексу, що дозволяє швидко знаходити релевантні фрагменти тексту на основі подібності векторних вкладень. На етапі генерації вкладень текстові фрагменти та метадані, попередньо отримані з веб-сторінок, перетворюються у вектори фіксованої розмірності за допомогою моделі векторизації. Ці вектори разом із допоміжною інформацією зберігаються у векторній базі даних ChromaDB.

Векторна база даних ChromaDB організована у вигляді колекції записів, кожен з яких описує одну сторінку або її фрагмент та пов’язану з інформацію (метадані). Структура кожного запису включає такі елементи:

- embedding – векторне вкладення, згенероване з тексту сторінки.
- document – текст сторінки, що був векторизований.
- id – унікальний ідентифікатор у форматі "{URL}_{порядковий номер фрагменту}", що точно дозволяє ідентифікувати походження тексту.
- metadata – об’єкт з метаданими сторінки
- metadata.url – посилання на сторінку.
- metadata.chunk_number – порядковий номер сегментованого фрагменту тексту сторінки.
- metadata.metadata – набір SEO-метаданих сторінки.
- metadata.metadata_embedding – векторне представлення метаданих, використовується для повторного ранжування результатів пошуку.

Для оцінки якості векторного пошуку без урахування метаданих було проведено контрольне тестування за фіксованою вибіркою запитів. Було сформовано набір із 40 коротких тестових запитів довжиною в 4-10 слів, що охоплюють різні категорії типових інформаційних потреб абітурієнтів. Для кожного запиту вручну визначено одну сторінку з сайту вступної кампанії, яка містить релевантну відповідь – ця сторінка вважається еталонною для відповідного запиту. Для оцінки ефективності

векторного пошуку в даному проєкті використовувалась метрика Hits@K або, скорочено, H@K – це індекс ефективності, який вимірює ймовірність знайти правильний прогноз у перших найвищих K прогнозах моделі, найчастіше використовується K=10. H@K відображає точність моделі побудови векторних вкладень.[11]

$$H@K = \frac{|\{q \in Q : q < K\}|}{|Q|} \in [0, 1]$$

Формула 3. Hits@K

Успішним результатом вважався випадок, коли очікуване джерело (URL) містилося серед перших K результатів, повернутих пошуковою системою. Дана метрика дозволяє об'єктивно оцінити здатність системи виявляти релевантні сторінки на основі векторного представлення запиту та векторної БД.

Вибір саме такого підходу обумовлений практичними обмеженнями: у контексті запитів щодо вступної кампанії кількість потенційно релевантних сторінок може бути значною, а інформація часто дублюється або розподіляється по кількох джерелах. У зв’язку з цим повне ручне маркування всіх релевантних відповідей для кожного запиту було відкинуте через надмірну складність.

Такий метод може бути використано для попередньої оцінки ефективності різних стратегій векторизації та слугувати порівняльним маркером для подальшого покращення пошуку, зокрема з використанням метаданих. Однак, варто враховувати недолік цього методу при тестуванні, а саме – можливість отримання інформації з дублюючого альтернативного джерела, що не було вказано як еталонне. У такому випадку, система все ще зможе надати правильну відповідь на основі іншого джерела, хоча при тестуванні еталонний результат так і не було знайдено.

У процесі розробки було протестовано дві моделі від OpenAI з різною розмірністю векторів – «text-embedding-3-small» з розміністю векторів в 1536 та «text-embedding-3-large» з розмірністю 3072. Згідно з документацією OpenAI на квітень 2025, модель «text-embedding-3-small» має значно нижчу вартість використання $0.02

проти $0.13, та сфокусована на швидкість роботи, що може бути корисним для обробки запитів у реальному часі. Натомість, модель «text-embedding-3-large» вважається більш точною та чутливою до контексту, що може покращити результати пошуку за складними запитами.[13]

Для «text-embedding-3-small» співвідношення H@K були наступними: K = 5, H@K = 0.575; K = 10, H@K = 0.725; K = 20, H@K = 0.8. Далі наведено таблицю з результатами тестування H@K, K = 20 з розбиттям на підкатегорії:

	Hit		
Category	false	true	Quantity
Bachelor	2	8	10
General	3	12	15
Master	2	8	10
PHD	1	4	5
Total	8	32	40

При тестуванні з використанням моделі «text-embedding-3-large» результати виявились помітно гіршими, з H@K = 0.6 для K = 20. Такий результат свідчить про те, що модель «text-embedding-3-small» є кращим варіантом для обраної задачі, яку можна узагальнити як пошук коротких текстів за простими запитами (середня довжина векторизованих текстів – 2000 символів).

Також, було проведено тестування open-source моделі «bge-m3». Ця модель продемонструвала наступні результати H@K: K = 5, H@K = 0.725; K = 10, H@K = 0.8; K = 20, H@K = 0.875.

	Hit		
Category	false	true	Quantity
Bachelor	2	8	10
General	2	13	15
Master	1	9	10
PHD	0	5	5
Total	5	35	40

У результаті цього дослідження, модель «text-embedding-3-small» продемонструвала краще співвідношення точності, швидкодії та вартості при роботі з короткими текстами (до ~2000 символів) у порівнянні з «text-embedding-3-large», що свідчить про те, що збільшення розмірності не гарантує автоматичного покращення якості в задачах такого типу. Модель «bge-m3» як альтернативне рішення перевершила обидва варіанти від OpenAI за всіма значеннями K, зокрема при K=20 показала найвищий результат (H@K = 0.875). Це робить її ефективним безкоштовним інструментом для побудови векторного пошуку. У подальших експериментах «text-embedding-3-small» та «bge-m3» будуть використані для глибшого порівняння при доповненні векторного пошуку ранжуванням за метаданими.

4.4 Обґрунтування параметрів сегментації тексту

У межах проєкту було прийнято рішення використовувати моделі «text-embedding-3-small», «text-embedding-3-large» та «bge-m3» для побудови векторного індексу текстових фрагментів. Дані моделі підтримують створення векторних вкладень до 8192 токенів на запит [14], що дозволяє генерувати вкладення для достатньо об’ємних текстів без додаткової сегментації.

З метою перевірки, чи необхідно застосовувати попередню сегментацію тексту перед векторизацією, було проведено емпіричний аналіз обсягу вмісту окремих сторінок з офіційного сайту приймальної комісії НаУКМА (vstup.ukma.edu.ua). Для цього, до функції автоматизованого обходу сайту, описаної у попередньому розділі, було додано функцію логування кількості символів у тексті без урахування HTML-розмітки і знаків пунктуації, що дозволило оцінити приблизний обсяг вхідних даних для подальшої векторизації. Для зручності аналізу, результати були збережені у вигляді CSV-файлу, в якому кожен запис містить унікальний URL сторінки як ідентифікатор, а також відповідне числове значення довжини тексту, що дозволяє відсортувати сторінки за кількістю символів.

```python
def log_page_size(url, page_text, csv_file="page_sizes.csv"):
    """Log the URL and the length of the page text to a CSV file."""
    file_exists = os.path.isfile(csv_file)
    with open(csv_file, mode="a", newline="", encoding="utf-8") as file:
        writer = csv.writer(file)
        if not file_exists:
            writer.writerow(["URL", "Page Text Length"])
        writer.writerow([url, len(page_text)])
```

Рисунок 6. Функція логування кількості символів сторінки

Було сформовано список сторінок із найбільшою кількістю тексту, на основі якого можна оцінити граничну потребу у поділі:

URL	**Довжина тексту (символів)**
https://vstup.ukma.edu.ua/master-degree/motyvaciyniy-lyst	8298
https://vstup.ukma.edu.ua/NUOU-Ivana-Cherniakhovskoho	8250
https://vstup.ukma.edu.ua/education-program-info?ep-id=113	8030
https://vstup.ukma.edu.ua/master-degree/preferential-categories/special-terms	7865
https://vstup.ukma.edu.ua/education-program-info?ep-id=67	7718

Для точнішої оцінки обсягів вхідного контексту при генерації вкладень було враховано результати дослідження Language Model Tokenizers Introduce Unfairness Between Languages (2023) [12], в якому проаналізовано ефективність роботи токенізаторів з різними мовами, зокрема cl100k_base, що використовується у моделях OpenAI, та XLM-RoBERTa у «bge-m3». Згідно з результатами дослідження, спостерігається значна варіативність кількості токенів залежно від мови при токенізації аналогічних за змістом текстів. Для системної оцінки того, наскільки справедливо токенізатори обробляють еквівалентні речення, запропонвано поняття преміуму токенізації – співвідношення між кількістю токенів аналогічних текстів,

написаних різними мовами. Преміум токенізації для української мови в порівнянні з англійською для токенізатора cl100k_base складає 3.00.

З метою практичного підтвердження можливості обробки найбільших текстових фрагментів без попередньої сегментації, було вирішено здійснити фактичний підрахунок кількості токенів для сторінки з найбільшим обсягом тексту. Такий підхід дозволяє оцінити реальне навантаження на токенізатор та гарантувати, що навіть у граничних випадках система залишатиметься в межах допустимого контекстного вікна при генерації текстових вкладень.

Для первинної оцінки потенційної кількості токенів без фактичного прогону тексту через токенізатор було застосовано наближену методику розрахунку. Для англійської мови у токенізаторі cl100k_base середньою нормою вважається співвідношення близько чотирьох символів на один токен. Використаємо наступну формулу для визначення приблизної кількості токенів українською T_U, де L – довжина тексту (8298), а P – преміум токенізації.

$$T_U = \frac{L}{4} \times P$$

Формула 4. Приблизний підрахунок токенів cl100k_base

Результатом приблизної оцінки є довжина тонізованого тексту в 6224 токени, що є значно нижчим за ліміт моделі «text-embedding-3-small» у 8192 токени, та дозволяє працювати з текстами сайту вступної кампанії НаУКМА без додаткового розбиття текстів на менші фрагменти.

Проте емпіричне тестування з бібліотекою «tiktoken» показало, що фактичне співвідношення символів до токенів є суттєво кращим, ніж теоретично передбачене. Для найбільшої за обсягом сторінки, що містила 8298 символів, реальна кількість токенів після токенізації склала 5103 токени. Це відповідає приблизно 1.63 символа на токен для української мови. Дана невідповідність може бути зумовлена тим, що для

обрахунку паритету використовується перекладений текст, що має більшу кількість символів.

У випадку з моделлю «bge-m3», яка використовує токенізатор XLM-RoBERTa, співвідношення між символами та токенами є набагато сприятливішим. Згідно з наявними оцінками, паритет символів до одного токена для української мови складає близько 1.2, що дозволяє ще ефективніше використовувати контекстне вікно моделі. Для тієї ж сторінки довжиною в 8298 символів, приблизна кількість токенів становила 1874, що суттєво нижче від граничного ліміту в 8192 токени для цієї моделі [19], але й від аналогічного показника для «cl100k_base», де було зафіксовано 5103 токени. Це свідчить про те, що модель «bge-m3» є більш економною з точки зору токенізації та краще підходить для обробки довших текстів українською мовою без попереднього дроблення на менші фрагменти.

Проведені розрахунки та вимірювання засвідчили, що навіть найбільші за обсягом текстові фрагменти, отримані з джерела, не перевищують ліміту контекстного вікна моделей «text-embedding-3-small» та «bge-m3». Це дозволяє оптимізувати процес обробки даних, уникнувши необхідності їх попередньої сегментації та спрощуючи подальшу реалізацію побудови векторного індексу. Проведене тестування також дозволило підтвердити практичну застосовність преміумів токенізації, запропонованих в дослідженні Language Model Tokenizers Introduce Unfairness Between Languages (2023). Підрахунки кількості токенів показали високий рівень відповідності з результатами цього дослідження. Запропонована формула приблизного підрахунку токенів з використанням преміуму є придатною для практичного застосування в умовах, де відсутня можливість спрогнозувати максимальний розмір текстового фрагменту, що можна додати у індекс.

4.5 Реалізація пошуку релевантного контексту

Після побудови векторного індексу наступним етапом є реалізація пошуку релевантної інформації, яка буде використовуватися як контекст для генерації відповіді LLM. У цій системі пошук реалізовано за принципом семантичної близькості: користувацький запит переводиться у векторне представлення за допомогою тієї ж моделі вкладень, якоб були згенеровані вектори, після чого проводиться порівняння з наявними векторами в базі знань. Використання однієї моделі для пошуку є обов'язковою умовою для коректності його роботи.

На першому етапі пошуку здійснюється прямий семантичний пошук за допомогою алгоритму ANN (Евклідова відстань), який повертає фіксовану кількість сторінок K з їх текстом та метаданими. Такий пошук зазвичай повертає сторінки чи фрагменти сторінок, які можуть бути лише частково релевантними до сформульованого запиту.

Щоб покращити якість результатів, реалізовано етап повторного ранжування знайдених результатів за метаданими. Для цього, відбувається порівняння пошукового вектору з векторами метаданих, побудованими з коротких текстів, доповнених ключовими словами та позбавленими шуму. Для порівняння застосовується формула косинусу подібності, що отримується за допомогою скалярного добутку векторів у Евклідовому просторі.

Результатом цього пошуку є повернення повного тексту знайдених сторінок разом з їх URL. Це дозволяє не лише використовувати контекст для генерації відповіді, а й надає користувачеві можливість самостійно переглянути повну інформацію, з якої була згенерована відповідь. Таке рішення забезпечує прозорість роботи додатку, надає кінцевому користувачу можливість дослідити знайдені джерела щоб отримати додаткові деталі та перевірити інформацію надану системою.

Нижче наведено діаграму що ілюструє фіналізовану версію двокрокового алгоритм пошуку з ранжуванням за метаданими:

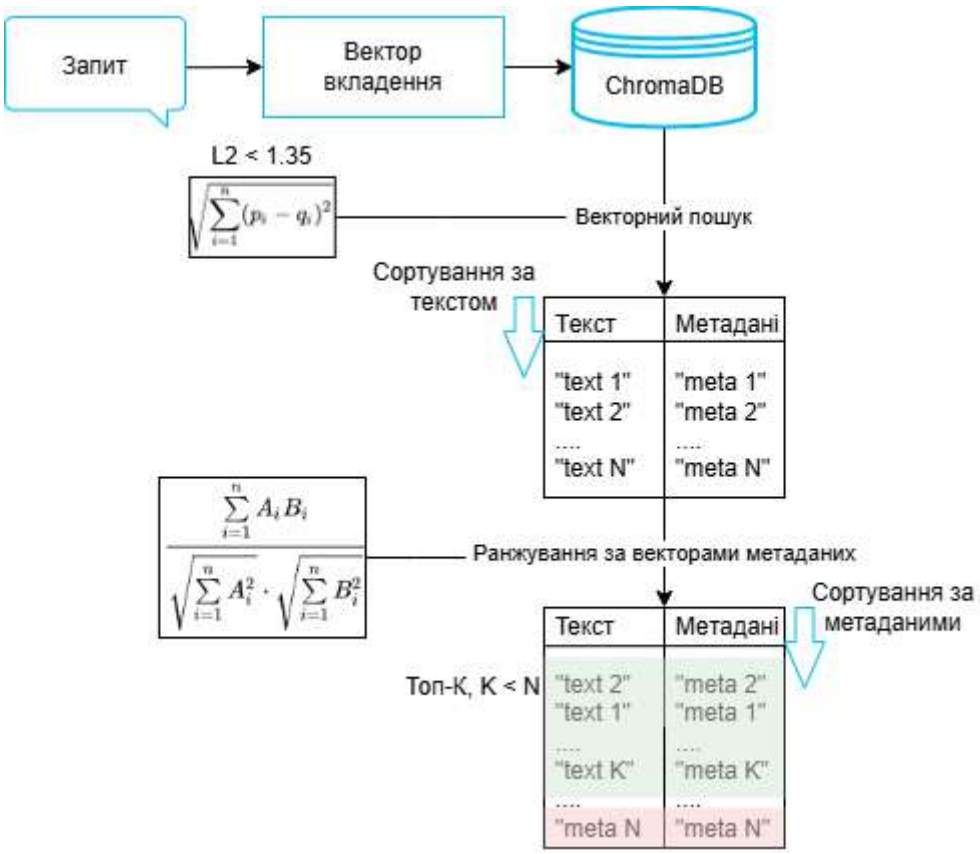

Рисунок 7. Діаграма алгоритму пошуку

На цьому зображенні проілюстровано три етапи пошуку релевантного контексту:

1. Генерація векторного вкладення з запиту користувача.
2. Пошук релевантних текстів в ChromaDB з використанням Евклідової відстані, відсіювання нерелевантних результатів з топ-N.
3. Ранжування знайдених результатів за векторними вкладеннями метаданих сторінок.

Для перевірки впливу метаданих та другого етапу ранжування на якість результату, було використано тестування метрикою H@K, яке було описано в попередньому розділі. Однак у рамках цього експерименту було враховано особливість двоетапної стратегії пошуку, де на першому етапі відбувається вибір більш широкого пулу кандидатів, а на другому – повторне ранжування з урахуванням метаданих. З цією метою було введено два окремі параметри:

- K – кількість найкращих результатів, що використовуються для генерації відповіді.
- N – розмір початкового пулу фрагментів, отриманих на першому етапі векторного пошуку, серед яких здійснюється повторне ранжування за сторінками з урахуванням метаданих.

Тест продуктивності метрикою H@K з K=10, K=20 та N=30 показав кращі результати за простий векторний пошук без повторного ранжування з моделлю «text-embedding-3-small» та відсутність покращень з моделлю «bge-3m», нижче наведено таблиці з порівнянням результатів, коефіцієнт H@K позначено як «metadata» для алгоритму з ранжуванням за метаданими та «plain» для простого векторного пошуку.

text-embedding-3-small	K	metadata	plain
	10	0.75	0.725
	20	0.825	0.8
bge-m3	K	metadata	plain
	10	0.8	0.8
	20	0.875	0.875

Крім цього, для моделі «text-embedding-3-small» зменшилась середня позиція знайденого релевантного тексту в списку результатів, з 2.3 до 1.7 для K=5, з 3.55 до 2.37 для K=10, та з 4.53 до 3.34 для K=20.

Для подальшої оцінки ефективності було проведено подальше тестування цих двох моделей. Тестовий набір даних було розширено до 140 запитів, що містили більш реалістичні та наближені до природної мови формулювання. Зокрема, до запитів було навмисно включено семантично надлишкові конструкції, такі як "будь ласка", "я шукаю", "поясни, що таке" та подібні звернення. Такий підхід дозволив перевірити стабільність роботи системи в умовах інформаційного шуму, характерного для реальних запитів. Результати цієї перевірки довели ефективність застосування метаданих для моделі «text-embeddind-3-small», де показник H@K зріс на 7% та

показав результат подібний до попередньої перевірки. Натомість, цей показник знизився на 6-7% для моделі «bge-m3». При цьому, показники моделі «bge-m3» все ще залишаються найкращими.

text-embedding-3-small	K	metadata	plain
	10	0.7286	0.65
	20	0.8214	0.75
bge-m3	K	metadata	plain
	10	0.7714	0.8429
	20	0.8714	0.9357

Збільшення початкового пулу результатів (параметра N) до 40, всупереч очікуванням, призвело не до покращення, а до погіршення метрики Hits@K. Такий ефект може бути спричинений тим, що при здільшенні пулу відповідей збільшується частка нерелевантних результатів, метадані яких можуть бути ближчими за формальними ознаками. У результаті релевантний документ опускається нижче в ранжуванні і випадає з топ-К.

У результаті перевірки часу пошуку з ранжуванням по метаданих не було виявлено суттєвого впливу на продуктивність в порівнянні з простим пошуком. Ключовим чинником, що впливає на загальний час відповіді системи, залишається продуктивність моделі генерації вкладень, яка зазвичай вимагає найбільшої кількості обчислювальних ресурсів через використання нейромережі. Ранжування відбувається достатньо швидко для обмеженого пулу результатів. У випадку використання API OpenAI, додатковим фактором є мережева затримка при передачі запиту та отримання відповіді.

У процесі розробки було додано механізм відсікання нерелевантних результатів за пороговим значення схожості, що дозволяє зменшити кількість випадкових або нерелевантних фрагментів у пулі пошуку. Це рішення дає змогу системі уникати використання слабко пов’язаного з запитом тексту при генерації відповіді, а також –

просигналізувати про неможливість надати відповідь. Експериментальним шляхом було встановлено, що значення порогу, яке б не дозволило випадково відкинути релевантну інформацію, складає 1.35 для використання моделі «text-embedding-3-small» та 1.65 для «bge-m3». При ньому спостерігалось незначне покращення при тестуванні H@K на моделі від OpenAI, K=5 (з 0.65 до 0.675), та було збережено попередній показник при K=10.

Додатковим спостереженням стало те, що застосування порогового значення також дозволяє ефективно відсікати запити, що не стосуються обраного для системи домену знань. Це свідчить про потенційну корисність механізму фільтрування за пороговим значенням не лише для підвищення якості пошуку, а й для виявлення запитів, на які неможливо надати відповідь на основі наявної бази знань. Така поведінка може стати основою для повідомлення користувача про те, що відповідь не може бути сформована, зменшуючи ризик галюцинацій у випадку питання на яке не може бути знайдена відповідь. Крім цього, це оптимізує використання ресурсів генеративної моделі, оскільки кількість витрачених на генерацію ресурсів є пропорційною до об’єму запиту.

У підсумку було реалізовано алгоритм пошуку, який демонструє часткове покращення за простий векторний пошук в залежності від моделі та передбачає механізм відкидання нерелевантних результатів. Отримані покращення результату метрики H@K для моделі «text-embedding-3-small» свідчать про практичну користь такого алгоритму для підвищення точності пошуку саме для цієї моделі.

В той же час, наявність додаткового ранжування негативно вплинула на роботу моделі «bge-m3». Важливим висновком цього дослідження є те, що усі критично важливі аспекти якості системи векторного пошуку є тісно пов’язаними з обраною моделлю – цей чинник є визначним при побудові додаткових евристик для покращення векторного пошуку. Також, такі результати свідчать про різницю у підходах до векторизації тексту цими моделями.

Фінальна версія системи реалізована на основі потужнішої моделі «bge-m3», яка забезпечує найкращу якість векторного пошуку. Водночас, у системі залишено можливість використання менш ресурсоємної моделі «text-embedding-3-small» у поєднанні з двокроковим ранжуванням за метаданими. Такий підхід дозволяє адаптувати систему до різних умов розгортання.

Зокрема, у випадках з високим навантаженням або обмеженими обчислювальними ресурсами на сервері, вигіднішим може виявитись підхід із викликами зовнішнього API для генерації відповідей, без необхідності локального розрахунку вкладень. Натомість, при локальному зберіганні векторного індексу та ембедінгів, можливе пришвидшення відповіді та контроль над даними.

4.6 Формування відповіді за допомогою великої мовної моделі

Одним із ключових етапів побудови системи підтримки абітурієнтів є контроль якості вихідної відповіді, яка генерується великою мовною моделлю. Незважаючи на високу здатність LLM до узагальнення інформації та генерації природної мови, без чітко визначених інструкцій модель може генерувати надмірно загальні, стилістично невдалі або навіть хибні твердження. Для уникнення таких ситуацій застосовується підхід, відомий як інженерія запитів. Інженерія запитів (англ. prompt engineering) – це процес структурування тексту, який може бути інтерпретовним та зрозумілим для моделі генеративного ШІ. У цьому контексті, запит (англ. prompt) – це текст природною мовою, що описує завдання, яке повинен виконати ШІ.

Вхідний промпт у системі будується з кількох обов’язкових компонентів:

1. Інструкція до моделі щодо бажаного формату та стилю відповіді.
2. Запит користувача, сформульований природною мовою.
3. Знайдені фрагменти тексту з бази знань, які були знайдені пошуковим алгоритмом.
4. Допоміжна інформація про користувача (освітній рівень, форма навчання).

Під час розробки було використано Responses API – просунутий інтерфейс OpenAI для генерації відповідей різними моделями. Для налаштування відповідей LLM, дане API має структуру, що розділяє загальні інструкції для генерації (instruction) і запит (input).

Інструкція задає роль моделі та визначає правила, яких вона має дотримуватись при формуванні відповіді. У даному випадку інструкція містить опис ролі системи як "помічника абітурієнта", її функціональне призначення, вимоги до достовірності відповіді, правила, що регулюють ситуації, коли відповідь не може бути сформована, та вказівка щодо мови відповіді. Нижче наведено повний текст інструкції, що надається великій мовній моделі:

«Ти – помічник абітурієнта або студента НаУКМА. Використовуючи надані фрагменти з офіційного сайту, надай об’ємну відповідь на запит користувача. Якщо інформація не міститься в наданих фрагментах, однак стосується вступної кампанії, надай її на основі власних знань. З наданих фрагментів обери найбільш релевантний та додай це посилання <url> в кінці відповіді у форматі: “Джерело: <url>”. Якщо відповідь не можна надати впевнено, вкажи, що інформація відсутня або потребує уточнення. Якщо питання не стосується НаУКМА, вступної кампанії чи освітніх планів, повідом, що надаєш відповіді лише на ці питання. Ввід буде містити інструкцію щодо мови, якою має бути відповідь. Приклад такої інструкції: ‘Відповідь має бути написана такою мовою: uk’»

Система реалізує компонент генерації відповідей у рамках підходу Retrieval-Augmented Generation (RAG), що поєднує семантичний пошук з подальшою генерацією на основі релевантного контексту. В основі генерації лежить побудова промпту доповненого додатковою інформацією, який передається великій мовній моделі (LLM) для отримання сформульованої відповіді. Згідно з попередніми результатами тестування H@K, було обрано доповнювати запит текстом 10 сторінок – такий підхід позволяє досягти балансу між повнотою контексту та витратами на

обробку кожного запиту. Обробка та доповнення запиту складаються з наступних етапів:

1. Користувацький запит за наявності доповнений попереднім повідомленням та відповіддю. Попереднє повідомлення зокрема використовується для пошуку. Це допомагає враховувати контекст розмови у випадку уточнюючих питань. Наприклад, питання «Розкажи про спеціальність ІПЗ» може бути доповнене уточнюючим питанням з вказівним займенником по типу «Який прохідний бал на цю спеціальність?». На відміну від назви спеціальності «ІПЗ», займенник «цю» не містить семантичного значення поза контекстом попереднього речення. Отже, обробка таких сценаріїв потребує доповнення запиту.
2. До запиту додаються результати пошуку з посиланнями на сторінки. Пряме додавання посилань необхідне для доповнення відповіді джерелами інформації, та зменшує ризик галюцинації, коли неіснуюче посилання може бути згенероване мовною моделлю.
3. Запит доповнюється додатковими інструкціями та вказівкою щодо мови відповіді.

Важливою частиною налаштування генерації відповідей є встановлення параметру «temperature». Цей параметр може варіюватися від 0 до 2, і вищі значення зроблять результат більш об’ємним та випадковим, тоді як нижчі значення зроблять його більш детерміністичним [15]. У контексті системи підтримки користувачів доцільно протестувати різні значення цього параметру, щоб знайти баланс між точністю та варіативністю відповідей.

В коді було реалізовано API, яке дозволяє використовувати модуль генерації відповіді за допомогою POST-запиту. Нижче наведено приклад тіла запиту:

```
{
    message: "Який прохідний бал на ІПЗ?", context: {
      lastUserMessage: sessionStorage.getItem('lastUserMessage'),
      lastBotMessage: sessionStorage.getItem('lastBotMessage'),
      educationLevel: sessionStorage.getItem('educationLevel'),
      fundingType: sessionStorage.getItem('fundingType'),
    }
}
```

Рисунок 8: Структура запиту на генерацію відповіді

Параметри запиту: message – повідомлення користувача, context – контекст діалогу, параметри якого зберігаються в sessionStorage, lastUserMessage – останнє повідомлення користувача, lastBotMessage – остання відповідь, educationLevel – рівень освіти, fundingType – спосіб оплати вступу (бюджет, контракт, або за пільговими умовами).

У рамках обмеженого ручного тестування, проведеного з метою оцінки якості генерації відповідей, було виявлено, що використання моделі «bge-m3» демонструє кращу стабільність у порівнянні з «text-embedding-3-small». Зокрема, вона забезпечувала коректне знаходження відповідей із меншою потребою у перифразі початкового запиту. Це підтверджує попередні результати дослідження, де було виявлено що «bge-m3» є кращим варіантом при пошуці.

4.7 Користувацький інтерфейс

Для забезпечення зручної взаємодії абітурієнтів із системою була реалізована мінімалістична веб-сторінка, що дозволяє ставити запитання природною мовою та миттєво отримувати відповіді у форматі діалогу. Інтерфейс орієнтований на широку аудиторію – включно з користувачами без технічного бекграунду, тому пріоритетом при його розробці були простота, читабельність та контекстна підказка щодо джерела інформації. Нижче наведено знімок екрану з інтерфейсом застосунку:

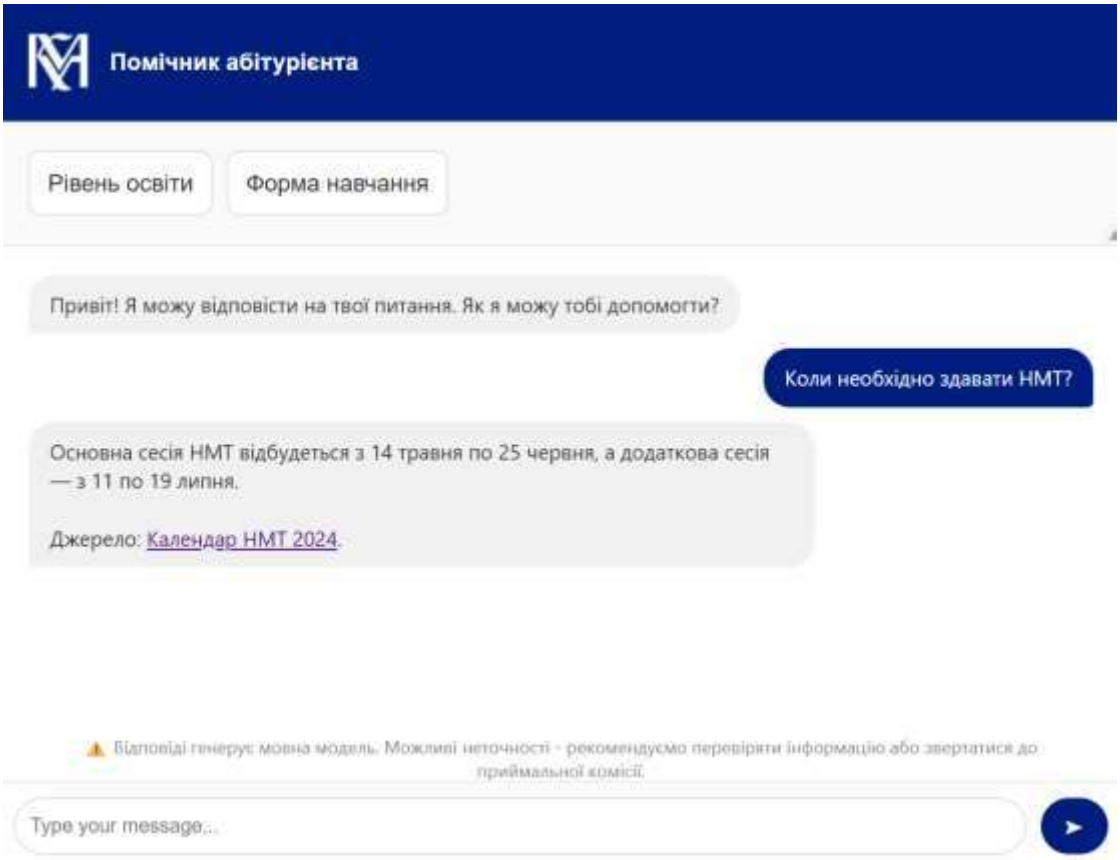

Рисунок 9. Інтерфейс чату

Ключові елементи інтерфейсу включають:

- Випадаючі списки для налаштування інформації про користувача – дозволяють вказати рівень освіти (бакалавр, магістр, аспірант) та форму навчання (бюджет, контракт, пільговик). Ця інформація передається як додатковий контекст при пошуку текстів, що покращує релевантність відповідей. Визначення цих параметрів є необов’язковим, а сам блок можна згорнути для збільшення діалогового блоку, що може бути зручно при перегляді
- Діалоговий блок – реалізовано просту логіку чату, де з лівого боку відображаються відповіді системи, а з правого – згенеровані відповіді системи.
- Джерела – кожна відповідь містить посилання на джерело (сторінку сайту), з якого було отримано інформацію, що забезпечує прозорість та можливість верифікації відповіді користувачем.

- Дисклеймер – в нижній частині діалогового блоку відображається попередження про те, що відповіді формуються мовною моделлю, отже можуть містити неточності та інформацію за потреби варто перевіряти у приймальній комісії.

Інтерфейс взаємодії з користувачем реалізовано у вигляді окремої веб-сторінки, яка може бути інтегрована на сайт вступної кампанії НаУКМА за допомогою тегу <iframe>.

Висновок до Розділу 4

У цьому розділі поетапно описано розробку системи підтримки абітурієнтів НаУКМА на основі архітектури RAG. Було розглянуто загальну архітектуру системи та визначено два основні процеси: заповенення векторного індексу та генерацію відповіді. Було детально описано процес отримання та очищення текстових даних із веб-ресурсів, що включає стандартні підходи до вилучення інформації з DOM і використання Selenium для роботи з динамічним вмістом.

Особливу увагу приділено процесу побудови векторного індексу, зокрема – генерації вкладень текстів та метаданих для подальшого використання в пошуку. Було обґрунтовано параметри сегментації тексту, виходячи з обмежень токенізатора та характеру сторінок. На основі експериментальних тестувань було встановлено ефективність моделей генерації вкладень при пошуку та виявлено, що модель «text-embedding-3-small» перевершує «text-embedding-3-large» при роботі з короткими україномовними запитами, а модель «bge-m3» перевершує обидві моделі.

Реалізовано алгоритм пошуку релевантного контексту у два етапи – простий векторний пошук та повторне ранжування за метаданими. З допомогою метрики H@K доведено ефективність використання повторного ранжування по метаданих для «text-embedding-3-small», в той час як «bge-m3» показали вищу якість без застосування цієї

додаткової евристики. На фінальному етапі було описано процес генерації відповідей з використанням спеціалізованого промпту та користувацький інтерфейс системи.

Розділ 5. Функціональні розширення та супутні компоненти системи

Окрім базового функціоналу, пов’язаного з пошуком релевантного контексту, генерацією відповідей та взаємодією з користувачем, у межах розробки системи було реалізовано низку додаткових компонентів, спрямованих на покращення зручності, стабільності та гнучкості її використання. Ці компоненти не є невід’ємною частиною застосунку, проте вони виконують важливі допоміжні функції. У цьому розділі описано реалізацію зазначених компонентів, а також обґрунтовано доцільність їх інтеграції до системи.

5.1 Система адміністрування векторного індексу сайту

Для забезпечення гнучкості у роботі системі, можливості її адаптації до змін у змісті сайту, та потенційного розширення функціоналу для роботи з іншими сайтами, було реалізовано адміністративний модуль для усправління вмістом векторної бази знань. Така панель управління забезпечує функціонал налаштування векторного індексу.

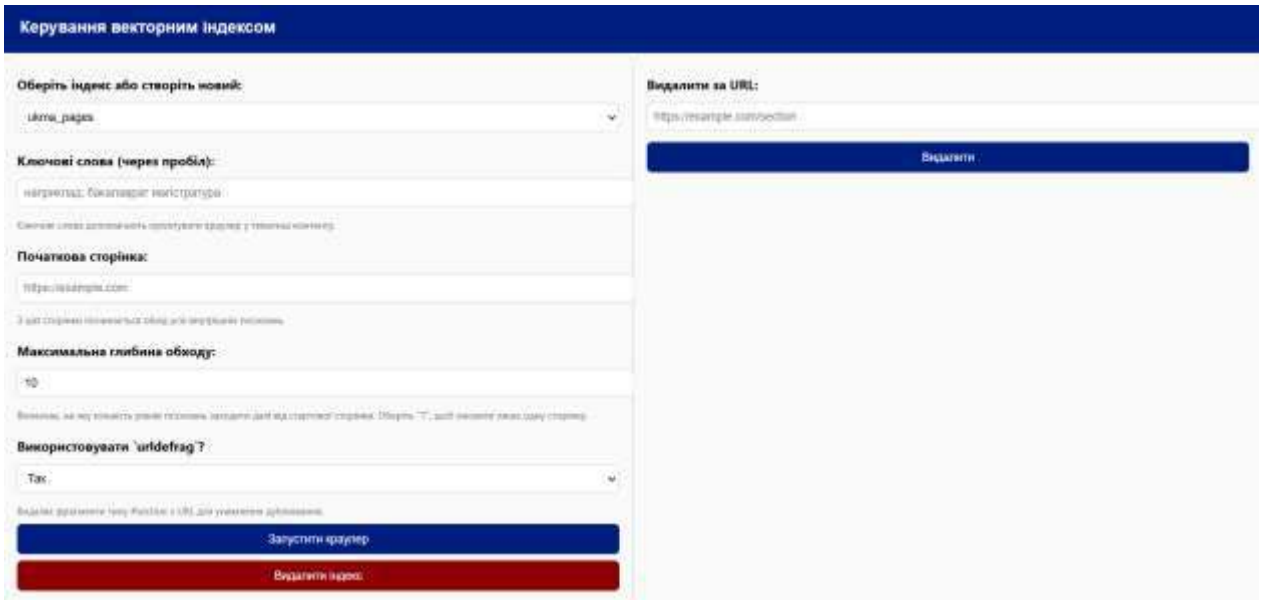

Рисунок 10. Панель керування векторними індексами

Було реалізовано наступні функції:

- Вибір існуючого векторного індексу для редагування або створення нового індексу.
- Ввід ключових слів, що можуть бути використані для ранжування за метаданими.
- Максимальна глибина обходу, з можливістю конфігурації для обходу однієї сторінки.
- Використання urldefrag – параметру що очищує сторінки від дублікатів з використанням посилань через «id» елементу (https://example.com/article#section). Може бути вимкненим у специфічних випадках, коли в такий спосіб реалізована навігація по різних сторінках.
- Повне видалення векторного індексу.

Даний набір операцій охоплює базові функції, однак може бути розширений для роботи з більшою кількістю параметрів. Перспективною є ідея налаштування фільтрів для повторюваних чи нерелевантних тегів,

5.2. Підтримка запитів іноземними мовами

Оскільки користувачами системи можуть бути іноземні абітурієнти, важливо забезпечити можливість обробки запитів, сформульованих іноземними мовами. Для цього було реалізовано двоетапний механізм автоматичної підтримки багатомовних запитів. Першим етапом є визначення мови для визначення необхідності перекладу. Якщо мова запиту відрізняється від української, запит перекладається на визначену мову. Попередній переклад запиту є важливим при векторному пошуку, оскільки усі збережені векторні представлення текстів збережені українською мовою, що погіршує якість пошуку у випадку використання іншої мови у запиті. В результаті практичного тестування з використанням метрики H@K, якість пошуку текстів англійською та

французькою мовами погіршилась на 5%, що свідчить про погіршення якості, яке можна спробувати покращити за допомогою перекладу запиту.

На першому етапі використовується бібліотека «langid» – Python-бібліотека для визначення мови тексту. Вона підтримує понад 90 мов і забезпечує високу точність розпізнавання завдяки використанню класифікатора, натренованого на великому корпусі текстів [16]. Бібліотека «langid» дозволяє проводити повторне навчання або обмеження набору мов, що особливо корисно для підвищення точності в умовах вузької предметної області чи близьких мовних груп.

Якщо мова запиту відрізняється від української, запит автоматично перекладається українською мовою за допомогою бібліотеки «translate», яка забезпечує простий інтерфейс до популярних сервісів машинного перекладу (за замовчуванням – MyMemory) [17]. Перекладений запит далі обробляється системою у звичайному режимі: виконується пошук релевантного контексту та генерація відповіді. Якщо мова запиту була визначена як іноземна, то до інструкції для мовної моделі при формуванні запиту додається назва цієї мови у форматі коду, що вказує на мову, якою потрібно надати відповідь. Делегування перекладу відповіді на LLM дозволяє спростити систему, поєднавши переклад та генерацію відповіді в один крок, що покращує швидкість роботи.

Обраний підхід для підтримки запитів іноземними мовами дозволяє підтримувати широкий набір мов без необхідності зберігання перекладених версій сайту, що зменшує витрати на заповнення векторної бази даних та пришвидшує роботу системи. До того ж, цей підхід не вимагає вибору мови користувачем, покращуючи досвід взаємодії із системою підтримки.

5.3. Аналіз кіберзагроз для RAG-систем

Інтелектуальні системи, що базуються на підході RAG, мають низку специфічних ризиків у сфері інформаційної безпеки. Враховуючи архітектуру

рішення – використання векторного пошуку, зовнішніх API та генерацію відповіді на основі природної мови, варто виокремити ключові потенційні загрози.

Зловживання API

Одним із найбільш реалістичних сценаріїв кібератаки є зловживання публічним інтерфейсом API. Відсутність обмежень на кількість запитів може призвести до DoS-атаки (Denial of Service), вичерпання квоти платного API OpenAI або до погіршення швидкості роботи внаслідок надмірного навантаження. Для протидії цьому у рамках проєкту було впроваджено механізм обмеження кількості запитів за IP-адресою.

Prompt Injection

Ще однією загрозою, притаманною системам, що взаємодіють із LLM, є prompt injection – маніпулювання змістом запиту користувача з метою змінити поведінку моделі, підмінити інструкції або змусити її ігнорувати захисні обмеження. Однак у контексті системи підтримки вступників ця атака має обмежену релевантність. Вся інформація, що використовується для генерації відповідей, базується виключно на публічних даних, отриманих з сайту вступної кампанії НаУКМА. Таким чином, навіть у разі вдалої ін'єкції, ризику розголошення конфіденційної інформації не існує. З цих міркувань, не передбачено впровадження спеціальних механізмів протидії цій загрозі.

Висновок до Розділу 5

У цьому розділі було розглянуто низку функціональних розширень, що покращують роботу системи. Основним покращенням є створення адміністративного інтерфейсу, який забезпечує створення нових індексів, запуск веб-кроулеру для оновлення всього індексу чи індивідуальних сторінок. Завдяки автоматичній підтримці запитів іноземними мовами покращено користувацький досвід для іноземців.

Результати та висновки

У межах роботи було реалізовано систему інтелектуальної підтримки абітурієнтів на основі генерації з доповненням пошуком, що поєднує переваги великих мовних моделей з механізмами пошуку релевантного контексту. Система охоплює повний цикл обробки запитів: від збору й обробки даних з веб-джерел до генерації відповідей на природній мові з посиланням на джерело інформації. Також, було реалізовано користувацький інтерфейс чату.

Було запропоновано та реалізовано двоетапний алгоритм пошуку: первинний векторний пошук за текстом сторінки після чого застосовується повторне ранжування на основі метаданих, що також були перетворені у векторні представлення. Проведене тестування з використанням метрики Hits@K засвідчило, що така стратегія може підвищити якість пошуку та релевантність знайдених результатів для моделі «text-embedding-3-small» для побудови векторних вкладень. Було проведено порівняльний аналіз моделей «text-embedding-3-small» та «bge-m3», в результаті якого виявлено що остання модель є кращою за «text-embedding-3-small» у контексті поставленого завдання.

Окрему увагу приділено інфраструктурній частині системи. Було створено прототип системи побудови та адміністрування векторних індексів веб-сторінок, який дозволяє автоматизовано індексувати сайт шляхом рекурсивного обходу. З метою зручного керування індексами реалізовано веб-інтерфейс, що дозволяє налаштовувати параметри обходу сайту, додавати тексти вручну, а також переглядати конфігурацію поточних індексів.

Таким чином, розроблене рішення продемонструвало не лише технічну життєздатність архітектури RAG у прикладному освітньому середовищі, але й її практичну ефективність у більш загальному контексті автоматизованої інформаційної підтримки. Отримані результати створюють підґрунтя для подальшого покращення системи та можливості її впровадження як офіційного каналу підтримки.

Обмеження та перспективи подальшого розвитку

Незважаючи на досягнуті результати, система має низку обмежень, які визначають можливі напрямки її подальшого розвитку.

Обмеження джерел інформації

Поточна реалізація зосереджена лише на сторінках сайту вступної кампанії НаУКМА, що обмежує можливість оцінки універсальності та масштабованості системи. Подальше тестування на інших веб-сайтаї дозволить перевірити адаптивність алгоритмів до різних структур сайтів і тематики контенту. Розширення системи до універсального інтелектуального асистента з централізованим векторним індексом, уніфікованою базою знань і підтримкою діалогу природною мовою є перспективним напрямом. Однак така розробка потребуватиме вирішення ряду задач, зокрема пов’язаних з ефективною обробкою великого обсягу даних і нестандартизованих веб-структур.

Відсутність підтримки текстових документів

У поточній версії система не враховує інформацію, представлену у форматі PDF та інших форматах текстових документів, попри те, що значна частина нормативних документів, інструкцій, договорів та офіційних заяв поширюється саме у такому вигляді. Це створює інформаційну прогалину для користувачів, які очікують комплексних відповідей. Наразі, ця інформаційна потреба частково вирішена за рахунок надання сторінки з джерелом інформації, що може містити потрібний договір.

Тестування з застосуванням більшої кількості метрик

Через складність формування великого тестового набору даних, у рамках дослідження не було реалізовано повноцінного вимірювання точності та повноти (precision, recall). Застосування цих метрик може виявити потенційні недоліки системи та стати інструментом для покращення системи. Це відкриває можливість для

подальшої оптимізації системи на основі більшої кількості об'єктивних показників якості.

Список використаних джерел

1. Вікімедіа, У. П. (2023, June 15). Велика мовна модель. https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%BB%D0%B8%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C
2. Gan, E., Zhao, Y., Cheng, L., Yancan, M., Goyal, A., Kawaguchi, K., Kan, M., & Shieh, M. (2024). Reasoning robustness of LLMs to adversarial typographical errors. Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, 10449–10459. https://doi.org/10.18653/v1/2024.emnlp-main.584
3. Compare models - OpenAI API. (n.d.). https://platform.openai.com/docs/models/compare?model=gpt-4o
4. hallucination. (2025). In Merriam-Webster Dictionary. https://www.merriam-webster.com/dictionary/hallucination
5. Tonmoy, S. M. T. I., Zaman, S. M. M., Jain, V., Rani, A., Rawte, V., Chadha, A., & Das, A. (n.d.). A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models. arXiv.org. https://arxiv.org/abs/2401.01313
6. Вікімедіа, У. П. (2020, October 20). Вкладання слів. https://uk.wikipedia.org/wiki/%D0%92%D0%BA%D0%BB%D0%B0%D0%B4%D0%B0%D0%BD%D0%BD%D1%8F_%D1%81%D0%BB%D1%96%D0%B2
7. Вікімедіа, У. П. (2023, October 25). Векторна база даних. https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%BD%D0%B0_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85
8. Team, E. P. (2025, May 15). Understanding the approximate nearest neighbor (ANN) algorithm. Elastic Blog. https://www.elastic.co/blog/understanding-ann
9. LLM Leaderboard - Compare GPT-4o, Llama 3, Mistral, Gemini & other models | Artificial Analysis. (n.d.). https://artificialanalysis.ai/leaderboards/models
10. SEO Starter Guide: The Basics | Google Search Central | Documentation | Google for Developers. (n.d.). Google for Developers. https://developers.google.com/search/docs/fundamentals/seo-starter-guide
11. Chen, Z., Wang, Y., Zhao, B., Cheng, J., Zhao, X., & Duan, Z. (2020). Knowledge Graph Completion: A Review. *IEEE Access*, *8*, 192435-192456.
12. Petrov, A., Emanuele, L. M., Torr, P. H. S., & Bibi, A. (2023, May 17). Language model tokenizers introduce unfairness between languages. arXiv.org. https://arxiv.org/abs/2305.15425
13. The guide to text-embedding-3-small | OpenAI. (n.d.). https://zilliz.com/ai-models/text-embedding-3-small

14. Vector embeddings - OpenAI API. (n.d.). https://platform.openai.com/docs/guides/embeddings#embedding-models
15. API Reference - OpenAI API. (n.d.). https://platform.openai.com/docs/api-reference/responses/create#responses-create-temperature
16. Saffsd. (n.d.). GitHub - saffsd/langid.py: Stand-alone language identification system. GitHub. https://github.com/saffsd/langid.py
17. translate. (2021, July 6). PyPI. https://pypi.org/project/translate/
18. Borgne, Y. L. (2024, February 26). OpenAI vs Open-Source Multilingual Embedding Models. Medium. https://medium.com/data-science/openai-vs-open-source-multilingual-embedding-models-e5ccb7c90f05
19. *BAAI/bge-m3 · Hugging Face*. (n.d.). https://huggingface.co/BAAI/bge-m3