

Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

ПРОГРАМНИЙ МОДУЛЬ СЕМАНТИЧНОГО АНАЛІЗУ ІЗ
ЗАБЕЗПЕЧЕННЯ КОНФІДЕНЦІЙНОСТІ ДАНИХ

DATA PRIVACY-PROTECTED SEMANTIC ANALYSIS
SOFTWARE MODULE

спеціальність 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КН-41
Савчак Андрій Романович

(підпис)

Керівник: к.т.н., доцент
Кошелюк Віктор Андрійович

(підпис)

Кваліфікаційну роботу
допущено до захисту
«__» _____ 2026 р.
Гарант освітньої програми:
д.пед.н., професор
Тулашвілі Юрій Йосипович

(підпис)

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій

Кафедра комп'ютерних наук

Ступінь вищої освіти: бакалавр

Галузь знань: 12 Інформаційні технології

Спеціальність: 122 Комп'ютерні науки

Освітня програма: «Комп'ютерні науки»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Валерій ЛІЩИНА

«16» січня 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ПЕРШОГО (БАКАЛАВРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Савчак Андрій Романович

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Програмний модуль семантичного аналізу із забезпеченням конфіденційності даних»

Керівник к.т.н., доцент Кошелюк Віктор Андрійович

затверджені наказом закладу вищої освіти від «16» січня 2026 р. № 32/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи «02» червня 2026 р.

3. Вихідні дані до роботи:

Аналіз сучасних методів обробки природної мови (NLP) та технологій семантичного пошуку для управління корпоративною документацією, використання моделей Sentence-Transformers (SBERT) і гібридних баз даних (ChromaDB, SQLite), засобів FastAPI, Streamlit, Python та алгоритмів криптографічного захисту для розробки локального вебзастосунку.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):

Аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів проектування, опис функціонального наповнення об'єкта проектування, розробка й обґрунтування системного наповнення, оцінка ергономічних та надійнісних параметрів проекрованої системи, функціонально-структурна схема роботи об'єкта проектування.

5. Перелік графічного матеріалу: 13 рисунків, презентація.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
<i>Аналіз проблеми за темою роботи та постановка завдань дослідження</i>	<i>Кошелюк В. А.</i>		
<i>Теоретичне дослідження</i>	<i>Кошелюк В. А.</i>		
<i>Практична реалізація</i>	<i>Кошелюк В. А.</i>		
<i>Спеціальні розрахунки</i>	<i>Кошелюк В. А.</i>		
<i>Показник запозичень тексту</i>	%		
<i>Інструментальна перевірка</i>	<i>Кошелюк В. А.</i>		
<i>Нормоконтроль</i>	<i>Кошелюк В. А.</i>		
<i>Гарант ОПП</i>	<i>Тулашвілі Ю. Й.</i>		

7. Дата видачі завдання «16» січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	<i>Провести огляд літературних джерел по темі кваліфікаційної роботи</i>	<i>до 17.02.2026 р</i>	
2	<i>Провести аналіз загальної проблеми і вибір напрямків дослідження</i>	<i>до 28.02.2025 р.</i>	
3	<i>Розробити функціональну схему роботи програмного продукту</i>	<i>до 12.03.2026 р</i>	
4	<i>Описати засоби розробки об'єкта проектування</i>	<i>до 26.03.2026 р.</i>	
5	<i>Практична реалізація об'єкта проектування</i>	<i>до 30.04.2026 р.</i>	
6	<i>Провести спеціальні розрахунки</i>	<i>до 18.05.2026 р.</i>	
7	<i>Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедрі</i>	<i>до 02.06.2026 р.</i>	

Здобувач вищої освіти _____ Андрій САВЧАК

Керівник роботи _____ Віктор КОШЕЛЮК

АНОТАЦІЯ

Савчак А. Р. Інтелектуальна система семантичного пошуку корпоративної документації. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерні науки». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

У роботі реалізовано інтелектуальну систему семантичного пошуку корпоративної документації. Під час виконання поставлених задач було проаналізовано проблематику класичного та семантичного пошуку, спроектовано архітектуру інформаційної системи, здійснено практичну розробку локальної системи семантичного пошуку з інтеграцією нейромережевої моделі (NLP) та бази даних, а також запропоновано методи захисту корпоративних даних від несанкціонованого доступу.

Ключові слова: семантичний пошук, обробка природної мови, NLP, векторна база даних, машинне навчання, Python, FastAPI, Streamlit, ChromaDB, SBERT.

ANNOTATION

Andriy Savchak. Intellectual semantic search system for corporate documentation. Manuscript.

Bachelor's qualifying thesis of the OP «Computer Science». Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification work consists of an introduction, four chapters, conclusions, a list of used sources and appendices.

The paper implements an intellectual semantic search system for corporate documentation. In the course of performing the tasks set, the problems of classical and semantic search were analyzed, the architecture of the information system was designed, the local semantic search system with the integration of a neural network model (NLP) and a database was developed, and methods for protecting corporate data from unauthorized access were proposed.

Keywords: semantic search, natural language processing, NLP, vector database, machine learning, Python, FastAPI, Streamlit, ChromaDB, SBERT.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ КВАЛІФІКАЦІЙНОЇ РОБОТИ ..	9
1.1 Аналіз сучасного стану проблем	9
1.2 Аналіз аналогічних програм, баз даних, систем, обґрунтування вибраного напрямку та вибір методів рішення	11
1.3 Аналіз і вибір засобів проектування системи семантичного пошуку корпоративної документації.....	13
1.4 Постановка завдання на кваліфікаційну роботу бакалавра	17
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМ	20
2.1 Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання.....	20
2.2 Функціонально-структурна схема роботи об'єкта проектування.....	25
РОЗДІЛ 3 РОЗРОБКА СИСТЕМИ СЕМАНТИЧНОГО ПОШУКУ КОРПОРАТИВНОЇ ДОКУМЕНТАЦІЇ.....	37
3.1 Практична реалізація об'єкта проектування. Побудова блок-схем модулів програми.....	37
3.2 Тестування та налагодження системи семантичного пошуку.....	43
3.3 Інструкція користувача з інсталяції та використання програмного продукту	49
РОЗДІЛ 4 СПЕЦІАЛЬНІ РОЗРАХУНКИ.....	52
ВИСНОВКИ.....	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	60
ДОДАТКИ.....	62

ВСТУП

Актуальність теми. На сучасному етапі розвитку підприємств роздрібною торгівлі однією з ключових проблем стає ефективне управління великими масивами корпоративної інформації. В умовах постійного масштабування бізнесу кількість внутрішніх регламентів, договорів та технічних інструкцій зростає експоненціально. Класичні методи пошуку інформації, що базуються виключно на посимвольному лексичному збігу, виявляються вкрай неефективними. Вони не враховують контекст запиту та професійну синонімію, створюючи «семантичний розрив» між побутовою мовою користувача та офіційною термінологією документації. Забезпечення швидкого та змістового доступу до цих даних є критичним фактором для злагодженої роботи всіх підрозділів. Тому впровадження систем семантичного пошуку на базі технологій обробки природної мови (NLP) та векторних баз даних є надзвичайно актуальним завданням для оптимізації корпоративного документообігу.

Метою кваліфікаційної роботи бакалавра є проектування та програмна реалізація локальної системи семантичного пошуку корпоративної документації для підвищення ефективності інформаційного забезпечення працівників торгової мережі на прикладі ТОВ «ВОЛИНСЬКА БІЗНЕС-МЕРЕЖА».

Об'єктом дослідження є процеси управління корпоративною інформацією та навігації по неструктурованих текстових масивах на підприємстві.

Предметом дослідження є моделі, алгоритми та програмні засоби семантичного пошуку на основі технологій обробки природної мови та векторних баз даних.

Для досягнення поставленої мети було сформовано та виконано наступні завдання:

- проаналізувати сучасний стан управління інформаційними потоками на підприємстві та виявити недоліки існуючих методів лексичного пошуку документів;

- дослідити існуючі технології обробки природної мови (NLP) та обґрунтувати вибір нейромережових алгоритмів і спеціалізованих векторних баз даних для локального розгортання;
- спроектувати функціонально-структурну архітектуру та логічну модель бази даних розроблюваної пошукової системи;
- здійснити програмну реалізацію модулів екстракції тексту, векторного індексування та клієнтського веб-інтерфейсу;
- провести тестування системи на реальних корпоративних даних підприємства та розробити рекомендації щодо її подальшого використання.

Новизна роботи полягає у комплексному застосуванні методів щільного векторного представлення тексту (архітектура Bi-Encoder) та спеціалізованої бази даних ChromaDB для обробки україномовної технічної документації в ізольованому корпоративному середовищі. Запропонований підхід дозволяє здійснювати високоточний пошук за змістом без використання відкритих хмарних API (наприклад, OpenAI), що гарантує збереження комерційної таємниці підприємства.

Практична цінність отриманих результатів полягає у тому, що в рамках роботи реалізовано повністю автономний програмний продукт, придатний до впровадження у виробничий процес. Практичне використання розробленої системи співробітниками ТОВ «ВОЛИНСЬКА БІЗНЕС-МЕРЕЖА» дозволяє звести час пошуку необхідних технічних інструкцій або регламентів до мінімуму, зменшуючи простой торгового обладнання та знижуючи щоденне навантаження на першу лінію підтримки IT-відділу.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Аналіз сучасного стану проблем

Функціонування сучасного підприємства роздрібної торгівлі являє собою складну багаторівневу систему, що потребує узгодженої взаємодії численних структурних підрозділів. Дослідження предметної області здійснюється на базі товариства з обмеженою відповідальністю «ВОЛИНСЬКА БІЗНЕС-МЕРЕЖА», яке є розгалуженою торговельною мережею. Особливості діяльності підприємства охоплюють не лише реалізацію товарів кінцевим споживачам, а й управління складською логістикою, контроль якості продукції, забезпечення стабільного функціонування ІТ-інфраструктури та організацію роботи персоналу. Ефективне управління зазначеними процесами можливе лише за умов чіткої регламентації бізнес-процесів, що зумовлює формування значних обсягів внутрішньої документації, яка потребує систематизації та впорядкування.

Інформаційні потоки підприємства характеризуються високим рівнем інтенсивності та різноманітністю. Щоденно між центральним офісом, логістичними центрами та торговими точками відбувається обмін значними обсягами даних. До складу корпоративної інформаційної бази входять нормативно-правові документи, внутрішні розпорядження, посадові інструкції, технічні регламенти налаштування POS-терміналів, договори з контрагентами та інвентаризаційні звіти. Обсяг накопичених даних постійно зростає і вже досягає значень, що вимірюються гігабайтами текстової інформації. Водночас формати зберігання даних є неоднорідними: використовуються як структуровані формати (XLSX, DOCX), так і неструктуровані (PDF), зокрема у вигляді сканованих копій паперових документів. Така різноманітність ускладнює процеси централізованої обробки, аналізу та систематизації інформаційних ресурсів.

На поточному етапі розвитку ІТ-інфраструктури управління документацією здійснюється переважно із застосуванням базових засобів

операційних систем та файлових серверів. Апаратне забезпечення підприємства відповідає сучасним вимогам щодо надійного зберігання даних: використовуються високопродуктивні серверні рішення, впроваджено механізми резервного копіювання та контролю доступу. Водночас програмні засоби, призначені для пошуку та навігації в інформаційних масивах, не відповідають сучасному рівню розвитку технологій обробки текстових даних. Пошук інформації в межах локальної мережі базується на класичних алгоритмах лексичного та повнотекстового пошуку, функціонування яких зводиться до формального зіставлення введених користувачем рядків із назвами файлів або їхнім вмістом.

Проведений аналіз засвідчив, що зазначений підхід має суттєві обмеження, оскільки не враховує контекст запиту, особливості професійної термінології, синонімічні зв'язки та морфологічні варіації мовних конструкцій. У практичній діяльності користувачі формують запити з використанням природної мови або професійного сленгу, що часто не збігається з формалізованими визначеннями, застосованими в документації. Наприклад, запит «як перезавантажити касу при зависанні» не корелює безпосередньо з формулюванням «апаратне скидання налаштувань фіскального реєстратора», що унеможливорює отримання релевантних результатів за допомогою стандартних засобів пошуку. У результаті працівники витрачають значний час на ручний перегляд документів або змушені звертатися до ІТ-персоналу, що призводить до нераціонального використання ресурсів. Таким чином, має місце так званий семантичний розрив між інформаційними потребами користувачів і наявними механізмами організації даних.

Особливості функціонування підприємств роздрібною торгівлю висувають підвищені вимоги до оперативності реагування на технічні інциденти. Навіть короткочасні затримки у роботі касових вузлів, спричинені відсутністю швидкого доступу до необхідних інструкцій, можуть призводити до утворення черг, зниження рівня обслуговування клієнтів і фінансових втрат. При цьому

модернізація апаратного забезпечення не забезпечує усунення зазначених проблем, оскільки їх природа має алгоритмічний характер.

Отримані результати аналізу обґрунтовують доцільність розроблення інтелектуальної системи обробки корпоративної текстової інформації. Перехід від традиційних методів лексичного зіставлення до підходів, що базуються на аналізі змістового наповнення документів, дозволить підвищити ефективність пошуку, оптимізувати інформаційні потоки, зменшити навантаження на технічний персонал та покращити загальні показники функціонування підприємства.

1.2 Аналіз аналогічних програм, баз даних, систем, обґрунтування вибраного напрямку та вибір методів рішення

Ринок корпоративного програмного забезпечення пропонує широкий спектр підходів до організації баз знань і забезпечення навігації у внутрішніх інформаційних ресурсах підприємств. Історично значна частина компаній, зокрема у сфері роздрібної торгівлі, використовувала архітектуру прямих файлових сховищ як базове рішення для зберігання документації. Однак із зростанням обсягів даних виникла об'єктивна потреба у впровадженні спеціалізованих інструментів, які умовно поділяються на кілька категорій: базові лексичні пошукові механізми, системи повнотекстового індексування та сучасні хмарні платформи з використанням методів машинного навчання.

Найбільш доступним і поширеним підходом залишається застосування вбудованих засобів операційних систем або базового функціоналу стандартних систем електронного документообігу. Основною перевагою такого рішення є відсутність додаткових витрат на впровадження та мінімальні вимоги до підготовки персоналу. Водночас встановлено, що зазначений підхід має принципові обмеження, оскільки базується виключно на посимвольному лексичному зіставленні. Подібні системи не забезпечують урахування контексту запиту та не підтримують обробку професійної синонімії, що істотно знижує

ефективність пошуку, особливо у випадках роботи зі складною технічною документацією, коли користувач не володіє точними формулюваннями.

Більш функціонально розвиненою альтернативою є індустріальні системи повнотекстового пошуку, зокрема рішення на базі екосистеми Elasticsearch. Здійснений аналіз показав, що такі системи мають низку суттєвих переваг, зокрема високу швидкодію обробки запитів у великих масивах даних завдяки використанню інвертованих індексів, а також підтримку механізмів нечіткого пошуку, що дозволяє компенсувати помилки введення користувачем. Водночас виявлено і ряд обмежень. Розгортання та супровід подібних рішень потребують значних обчислювальних ресурсів і залучення кваліфікованих спеціалістів з адміністрування. Крім того, у базовій конфігурації такі системи залишаються орієнтованими переважно на пошук за ключовими словами, що не забезпечує повноцінного подолання термінологічних розбіжностей між формулюваннями користувачів і змістом документів.

Останнім часом на ринку також закріпилися комерційні хмарні рішення класу інтелектуального пошуку (наприклад, Amazon Kendra або спеціалізовані модулі Microsoft). Хоча вони дійсно забезпечують глибоке семантичне розуміння тексту «з коробки», їх впровадження у внутрішню інфраструктуру торгової мережі стикається з низкою критичних бар'єрів.

Ризики інформаційної безпеки: необхідність передачі конфіденційних договорів, фінансових звітів та внутрішніх наказів на сервери сторонніх провайдерів прямо суперечить жорстким корпоративним політикам захисту комерційних даних.

Економічна обтяжливість: ліцензування таких хмарних продуктів вимагає значних фінансових витрат, оскільки вартість їх використання постійно масштабується залежно від кількості зроблених запитів чи загального обсягу проіндексованої документації.

З огляду на виявлені недоліки існуючих аналогів – семантичну обмеженість безкоштовних рішень, апаратну надмірність повнотекстових рушіїв та безпекові ризики комерційних хмарних платформ – найбільш

обґрунтованим напрямом є розробка власної локальної інтелектуальної пошукової системи. Оптимальним підходом до вирішення поставленої проблеми виступає застосування методів обробки природної мови (NLP) у тісному поєднанні з векторними базами даних. Замість класичного індексування окремих слів пропонується алгоритм перетворення текстових фрагментів корпоративної документації у багатовимірні числові масиви.

Вибір такого методу дозволяє реалізувати справжній пошук за змістом: математичне порівняння векторів запиту та документа здатне виявити їхню смислову близькість навіть за умови використання абсолютно різних слів. Програмну реалізацію доцільно виконувати мовою Python, яка на сьогодні володіє найпотужнішою екосистемою для побудови нейромережових рішень. Використання відкритих локальних моделей для генерації векторів гарантує повне збереження конфіденційності даних всередині захищеного периметра підприємства. Своєю чергою, застосування спеціалізованих векторних сховищ забезпечить необхідну швидкодію пошукової видачі без потреби в закупівлі дорогавартісного серверного обладнання, що повністю відповідає технічним та економічним вимогам компанії.

1.3 Аналіз і вибір засобів проектування системи семантичного пошуку корпоративної документації

Ефективність реалізації системи семантичного пошуку в умовах функціонування підприємства роздрібної торгівлі значною мірою визначається обґрунтованістю вибору технологічного стеку. У процесі добору інструментальних засобів враховувалися не лише показники продуктивності, а й низка критичних чинників, зокрема забезпечення конфіденційності корпоративних даних, можливості масштабування, сумісність із наявною апаратною інфраструктурою та оперативність розробки. Оскільки ключовим завданням є перехід від лексичних методів пошуку до семантичного аналізу, обрані програмні засоби повинні забезпечувати ефективну обробку

неструктурованих текстових даних із використанням алгоритмів машинного навчання.

У межах дослідження здійснено аналіз і обґрунтування вибору основних технологічних компонентів, використаних при розробленні інформаційної системи.

Вибір мови програмування є базовим елементом програмної реалізації є мова програмування. У контексті розробки систем штучного інтелекту та обробки природної мови (NLP) було розглянуто C++, Java та Python. Мови C++ і Java характеризуються високою продуктивністю виконання та ефективним управлінням пам'яттю, що є важливим для високонавантажених систем. Водночас їх використання супроводжується підвищеною складністю розробки, зокрема через необхідність реалізації значного обсягу допоміжного коду для базових операцій, що уповільнює створення та тестування прототипів.

За результатами аналізу обрано мову Python, яка займає провідні позиції в галузі Data Science та штучного інтелекту. Незважаючи на інтерпретований характер, її ефективність забезпечується використанням високопродуктивних бібліотек (зокрема PyTorch і TensorFlow), реалізованих із застосуванням C++. Це дозволяє поєднати зручність розробки з належним рівнем обчислювальної продуктивності. Крім того, наявність широкого спектра бібліотек для обробки тексту сприяє концентрації на реалізації логіки семантичного пошуку без необхідності розробки низькорівневих алгоритмів [1].

Засоби обробки тексту та формування векторних представлень. Ключовим компонентом системи є модуль перетворення текстових даних у векторні представлення (embeddings). У цьому контексті було проаналізовано можливості бібліотек Hugging Face Transformers та Sentence-Transformers (SBERT). Бібліотека Hugging Face Transformers надає доступ до широкого спектра попередньо навчених моделей (зокрема BERT, RoBERTa, DistilBERT) і підтримує їх адаптацію до специфіки предметної області, включаючи роботу з українською мовою та спеціалізованою термінологією.

Водночас для реалізації системи обрано фреймворк Sentence-Transformers, який є спеціалізованим розширенням архітектури Transformer, оптимізованим для задач визначення семантичної подібності текстів. Його використання забезпечує формування векторних представлень на рівні речень і абзаців, що є принципово важливим для коректної інтерпретації користувацьких запитів у корпоративному середовищі. Додатковою перевагою є можливість локального розгортання моделей, що гарантує збереження конфіденційності даних без залучення зовнішніх хмарних сервісів [2].

Вибір системи управління базами даних. Традиційні реляційні системи управління базами даних (наприклад, MySQL) не забезпечують ефективної роботи з багатовимірними векторними просторами, що є необхідною умовою реалізації семантичного пошуку. У зв'язку з цим було обрано комбінований підхід, який передбачає використання як векторних, так і реляційних сховищ.

Серед векторних рішень розглядалися FAISS (розробка компанії Meta) та ChromaDB. FAISS демонструє високу швидкість пошуку найближчих сусідів, однак потребує складного налаштування індексів і управління метаданими. Натомість ChromaDB забезпечує інтегровану підтримку зберігання як текстових даних, так і їхніх векторних представлень, а також ефективну фільтрацію за метаданими (наприклад, за типом документа або часовими параметрами). З огляду на це ChromaDB було обрано як основне рішення для реалізації векторного пошуку.

Для зберігання службової інформації про документи (шлях до файлу, автор, дата модифікації, контрольні хеш-суми) використано реляційну базу даних SQLite. Це компактне рішення не потребує розгортання окремого серверу баз даних, що спрощує інтеграцію в існуючу інфраструктуру підприємства та знижує витрати на впровадження і супровід системи. Інструменти для вилучення тексту з документів.

Оскільки документація підприємства зберігається у різних форматах, необхідно було обрати надійні бібліотеки для парсингу.

PyMuPDF (fitz) обрано для роботи з PDF-файлами через високу швидкість роботи та коректну обробку багатоколонкових текстів, що часто зустрічаються в інструкціях. На відміну від аналогів, PyMuPDF краще зберігає логічну послідовність читання тексту [6].

Python-docx стандартний інструмент для роботи з файлами Microsoft Word, що дозволяє вилучати текст, зберігаючи структуру заголовків, що важливо для правильного розбиття документа на смислові фрагменти (чанкінг).

Засоби розробки інтерфейсу та Backend-частини. Для взаємодії користувача із системою пошуку розглядалися варіанти створення повноцінного веб-додатка на базі Django або Flask. Проте для внутрішнього корпоративного інструменту швидкість розробки та простота інтерфейсу є пріоритетними.

FastAPI використано для створення серверної частини (Backend). Це асинхронний фреймворк, що забезпечує високу продуктивність та автоматичну генерацію документації API. Він ідеально підходить для інтеграції моделей машинного навчання, оскільки дозволяє паралельно обробляти декілька запитів.

Streamlit обрано для візуалізації та створення клієнтської частини. Це інструмент, що дозволяє швидко будувати інтерактивні інтерфейси для Data Science проектів на мові Python без глибокого знання JavaScript/HTML/CSS. Це дозволило створити зручне вікно пошуку з можливістю завантаження нових документів та відображенням знайдених фрагментів тексту в режимі реального часу.

Операційне середовище та апаратні вимоги. Розробка та тестування системи здійснювалися в середовищі Linux Mint. Вибір цієї операційної системи обумовлений її стабільністю, низьким споживанням ресурсів та нативною підтримкою більшості бібліотек для обробки даних, що часто мають проблеми з інсталяцією в середовищі Windows. Linux дозволяє гнучко налаштовувати системні залежності та забезпечує кращу продуктивність при роботі з Docker-контейнерами.

Щодо апаратних вимог, проєктована система орієнтована на роботу на центральних процесорах (CPU-only). Хоча графічні прискорювачі (GPU) значно

прискорюють процес генерації векторів, їхня відсутність у стандартних офісних серверах компанії вимагала оптимізації. Використання квантованих моделей (стиснутих форматів) дозволило досягти швидкості пошуку до 500 мс на звичайному процесорі, що є цілком прийнятним результатом для внутрішнього використання.

Обґрунтування та висновок щодо вибору засобів. Підсумовуючи аналіз, можна виділити остаточний перелік обраних засобів проектування:

- мова програмування: Python 3.10+ (через багату екосистему та швидкість розробки);
- ядро семантичного пошуку: Sentence-Transformers (моделі на базі архітектури Transformer);
- база даних: ChromaDB (для векторів) та SQLite (для метаданих);
- обробка документів: PyMuPDF, python-docx;
- серверна частина: FastAPI (асинхронний Backend);
- інтерфейс: Streamlit (швидке створення UI);
- ОС розробки: Linux Mint/Windows 11.

Обраний стек технологій дозволяє створити автономну, безпечну та продуктивну систему, яка здатна якісно покращити процеси пошуку корпоративної інформації в ТОВ «ВОЛИНСЬКА БІЗНЕС-МЕРЕЖА». Використання інструментів з відкритим вихідним кодом забезпечує економічну ефективність розробки, а локальний запуск моделей гарантує повний контроль над даними підприємства.

1.4 Постановка завдання на кваліфікаційну роботу бакалавра

Ефективне управління корпоративними знаннями та внутрішньою документацією є одним із ключових факторів стабільного функціонування сучасних підприємств роздрібної торгівлі. В умовах постійного масштабування бізнесу, відкриття нових філій та ускладнення технологічних процесів, обсяги неструктурованих даних зростають експоненціально. Забезпечення швидкого та

точного доступу співробітників до посадових інструкцій, технічних регламентів, договорів та наказів стає критичною необхідністю для оптимізації робочого часу та мінімізації простоїв. Одним із найперспективніших напрямків вирішення цієї проблеми є впровадження технологій обробки природної мови (NLP) та семантичного пошуку, які дозволяють комп'ютерним системам розуміти зміст тексту, а не лише шукати точні збіги символів.

Попри значний потенціал нейромережових алгоритмів, на сьогодні існує низка технічних та організаційних бар'єрів, що ускладнюють їх інтеграцію у корпоративне середовище. Однією з головних перешкод є проблема інформаційної безпеки: використання популярних хмарних рішень для аналізу тексту є неприпустимим через ризик витоку комерційної таємниці. З іншого боку, локальні базові інструменти операційних систем здійснюють виключно лексичний пошук. Через це співробітники часто стикаються з проблемою «семантичного розриву», коли запит формулюється побутовою мовою, а документ написаний суворою технічною термінологією, що призводить до нульової пошукової видачі [3].

З огляду на вищезазначене, розробка локальної, безпечної та оптимізованої системи інтелектуального пошуку, що здатна працювати з різномірними форматами корпоративної документації, є надзвичайно актуальним завданням. Такий інструмент дозволить ліквідувати термінологічний бар'єр, автоматизувати процес індексації файлів та суттєво підвищити швидкість розв'язання технічних інцидентів на підприємстві.

Метою даної кваліфікаційної роботи є проєктування та програмна реалізація локальної системи семантичного пошуку корпоративної документації, яка забезпечить швидкий та релевантний доступ до масивів неструктурованих текстових даних на базі ТОВ «ВОЛИНСЬКА БІЗНЕС-МЕРЕЖА». Для досягнення поставленої мети необхідно вирішити такі дослідницькі та практичні завдання:

- проаналізувати сучасний стан управління інформаційними потоками на підприємстві. Провести огляд існуючої IT-інфраструктури компанії, визначити

основні формати збереження документації (PDF, DOCX) та проаналізувати ключові проблеми, з якими стикаються співробітники під час лексичного пошуку регламентів та інструкцій;

– дослідити існуючі програмні рішення та методи обробки тексту. Виконати аналіз аналогічних пошукових систем. Обґрунтувати вибір математичних методів генерації векторних представлень (embeddings) та спеціалізованих векторних баз даних для локального використання;

– розробити архітектуру системи семантичного пошуку. Визначити основні функціональні модулі: контур індексації даних та контур пошуку. Спроекувати гібридну структуру бази даних, яка поєднуватиме реляційне сховище для метаданих файлів та векторне сховище для збереження математичних масивів тексту;

– реалізувати програмний модуль попередньої обробки та індексації документів. Виконати технічну розробку алгоритмів для автоматичного вилучення тексту з файлів різного формату, його очищення від артефактів форматування та розбиття на логічні фрагменти (чанкінг) з подальшою векторизацією за допомогою NLP-моделі;

– розробити серверну частину та користувацький інтерфейс. Забезпечити створення швидкого та асинхронного бекенду для обробки пошукових запитів. Розробити інтуїтивно зрозумілий графічний інтерфейс користувача, який дозволить персоналу легко взаємодіяти із системою без додаткового навчання;

– провести тестування розробленої системи на реальних даних. Оцінити ефективність пошукової видачі та швидкість роботи алгоритмів на базі центральних процесорів (CPU) із залученням реального архіву корпоративних документів;

– розробити рекомендації щодо подальшого вдосконалення ресурсу. На основі результатів тестування сформулювати пропозиції щодо масштабування системи, можливостей автоматичної синхронізації з локальними файловими серверами підприємства та оновлення мовних моделей відповідно до розширення внутрішньої термінологічної бази.

РОЗДІЛ 2

СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання

Відповідно до результатів аналізу предметної області, наведених у першому розділі, встановлено, що ключовим недоліком існуючої системи документообігу на підприємстві є застосування лексичних механізмів пошуку, які не забезпечують виявлення семантичних зв'язків між запитом користувача та змістом документів. З метою усунення зазначених обмежень обґрунтовано доцільність переходу до нової парадигми організації корпоративної бази знань, що передбачає використання методів обробки природної мови (Natural Language Processing, NLP) та векторного представлення текстових даних у багатовимірних просторах.

У даному підрозділі здійснено науково-технічне обґрунтування обраної структурної та функціональної моделі системи, а також проведено аналіз і вибір алгоритмічних підходів та програмних засобів, необхідних для реалізації ефективної пошукової платформи.

Обґрунтування вибору алгоритмів пошуку та методів векторизації. Традиційні системи інформаційного пошуку базуються на векторній моделі представлення документів (Vector Space Model) із використанням алгоритмів ранжування, таких як TF-IDF (Term Frequency–Inverse Document Frequency) та BM25. Зазначені методи визначають релевантність документів на основі частоти входження термінів запиту. Попри їхню обчислювальну ефективність, встановлено, що алгоритм BM25 має суттєве обмеження: у разі відсутності прямого лексичного збігу між термінами запиту та текстом документа релевантність визначається як нульова. Така ситуація, наприклад, виникає при використанні різних термінів для позначення одного поняття («дисплей» та «монітор»), що призводить до невиявлення релевантних документів. У науковій літературі це явище визначається як «семантичний розрив».

Для подолання зазначеного обмеження як базовий підхід обрано метод щільного векторного пошуку (Dense Retrieval), що ґрунтується на використанні архітектури нейронних мереж типу Transformer. Сутність методу полягає у перетворенні текстових фрагментів (речень або абзаців) у щільні вектори фіксованої розмірності (наприклад, 384 або 768 компонентів). У результаті навчання на великих текстових корпусах модель формує простір ознак, у якому семантично близькі тексти розташовуються у безпосередній близькості. Таким чином, різні за формою, але близькі за змістом висловлювання (наприклад, «апаратна помилка каси» та «збій роботи фіскального термінала») отримують подібні векторні представлення.

У процесі вибору методу векторизації було здійснено порівняльний аналіз класичних підходів (Word2Vec, FastText) та сучасних трансформерних моделей (BERT). Встановлено, що моделі типу Word2Vec формують статичні векторні представлення слів без урахування контексту їх використання, що обмежує їхню ефективність у задачах аналізу технічних текстів. З іншого боку, модель BERT забезпечує врахування двоспрямованого контексту, проте її застосування у класичній схемі (Cross-Encoder) є обчислювально витратним, оскільки передбачає попарне порівняння запиту з кожним документом, що унеможлиблює використання в режимі реального часу при великих обсягах даних.

З урахуванням проведеного аналізу обрано архітектуру Sentence-Transformers (SBERT), реалізовану за принципом Bi-Encoder. Даний підхід передбачає незалежне перетворення запитів і документів у вектори фіксованої розмірності з подальшим обчисленням міри їхньої подібності. Це дозволяє попередньо сформувати векторні представлення для всієї корпоративної документації та зберегти їх у базі даних, тоді як у процесі пошуку здійснюється лише векторизація користувацького запиту. У результаті обчислювальна складність знижується до лінійної $O(N)$, що забезпечує можливість реалізації швидкого пошуку без використання спеціалізованих апаратних прискорювачів.

Для оцінювання ступеня релевантності між векторними представленнями обрано метрику косинусної подібності (Cosine Similarity). На відміну від евклідової відстані, яка залежить від абсолютної довжини векторів, косинусна міра визначає кут між ними, що дозволяє коректно порівнювати текстові фрагменти різної довжини. Такий підхід є особливо ефективним у задачах зіставлення коротких користувацьких запитів із об'ємними текстовими документами, оскільки враховує напрямок векторів у семантичному просторі, а не їхню величину. Обґрунтування алгоритмів попередньої обробки та сегментації тексту.

Моделі на базі архітектури Transformer мають суворе обмеження на розмір вхідного тексту (контекстне вікно), яке найчастіше складає 512 токенів. З огляду на те, що корпоративні договори та регламенти можуть займати десятки сторінок, неможливо векторизувати весь документ як єдине ціле. Крім того, векторизація великого тексту розмиває його семантику, усереднюючи зміст різних розділів.

Відповідно, виникла потреба в розробці алгоритму сегментації – розбиття документа на менші логічні одиниці (чанкінг). Розглядалися такі методи:

- символний поділ (фіксований розмір без перекриття): жорстке розбиття тексту кожні 1000 символів. Метод швидкий, але має критичний недолік – межа розрізу може припасти на середину речення або розірвати логічно пов'язані терміни, що призведе до втрати сенсу;

- семантичний поділ (на основі абзаців): розбиття тексту за символами переносу рядка. Складність полягає в тому, що у форматах PDF абзаци часто розбиваються некоректно через візуальне форматування, що генерує безліч надто коротких і неінформативних фрагментів;

- для розроблюваної системи було спроектовано та обрано метод фіксованого розміру вікна з логічним перекриттям (Sliding Window Chunking). Цей алгоритм поділяє текст на блоки фіксованого розміру (наприклад, по 400 слів), але кожен наступний блок містить останні 50 слів попереднього. Перевагою такої схеми є гарантія того, що жодна важлива думка або складений

технічний термін не буде втрачено на стику двох фрагментів. Перекриття забезпечує збереження контексту, що є критично важливим для точності подальшої генерації векторів.

Порівняльний аналіз та вибір засобів збереження даних. Після перетворення тексту у числовий формат виникає проблема ефективного збереження та швидкого вилучення багатовимірних масивів. Традиційні реляційні бази даних (SQL) використовують B-дерева для індексування, які математично не здатні виконувати пошук «найближчих сусідів» (K-Nearest Neighbors, KNN) у просторах високої розмірності. Тому обов'язковим компонентом структурної схеми є спеціалізована векторна база даних.

Було проведено порівняльний аналіз найбільш поширених систем управління векторними базами даних, які відповідають вимогам корпоративної безпеки (можливість розгортання on-premise, без використання хмарних сервісів). Результати аналізу наведено у таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика засобів управління векторними базами даних

Критерій порівняння	FAISS (від Meta)	Qdrant	ChromaDB
Архітектура та тип	C++ бібліотека для індексування (не є повноцінною СКБД)	Повноцінна векторна СКБД (написана на Rust)	Векторна база даних, орієнтована на AI-додатки
Швидкість пошуку	Надвисока (мілісекунди для мільйонів записів)	Висока (з підтримкою розподілених кластерів)	Висока (оптимізована для локального використання)
Збереження метаданих	Відсутнє (потребує інтеграції зовнішньої SQL бази для метаданих)	Підтримується (вбудовані JSON-структури)	Підтримується «з коробки» (вбудована інтеграція з SQLite)
Складність розгортання	Висока (потрібне ручне управління індексами на диску)	Середня (потребує розгортання окремого Docker-контейнера/сервера)	Низька (працює як вбудована бібліотека у процесі Python)
Підтримка гібридного пошуку	Ні	Так	Так

З огляду на порівняльні характеристики, представлені в таблиці 2.1, як основний інструмент збереження даних було обрано ChromaDB. Вибір мотивовано тим, що FAISS є лише математичною бібліотекою і вимагала б

написання складного програмного проширення для синхронізації векторів із реальними текстовими файлами. Qdrant є потужним рішенням для enterprise-сегмента, але його клієнт-серверна архітектура є надлишковою для стартового розгортання в межах одного підприємства. ChromaDB пропонує ідеальний баланс: вона запускається локально в тому ж середовищі, що і серверний код, автоматично управляє збереженням як самих векторів, так і вихідного тексту та метаданих документів (назва файлу, автор, дата). Вона використовує алгоритм HNSW (Hierarchical Navigable Small World) для швидкого наближеного пошуку найближчих сусідів, що забезпечує стабільно високу швидкість обробки запитів на стандартному серверному обладнанні торгової мережі.

Обґрунтування програмних засобів для обробки файлів та розробки інтерфейсу. Функціональна схема розроблюваної системи передбачає наявність контуру завантаження документів, серверного ядра (API) та клієнтського інтерфейсу. Для реалізації цих компонентів було сформовано та обґрунтовано наступний технологічний стек.

Засоби екстракції тексту з документів. Технічна документація мережі зберігається переважно у двох форматах: PDF (скановані копії наказів, інструкції від виробників обладнання) та DOCX (внутрішні регламенти).

Для роботи з DOCX використовується бібліотека `python-docx`, яка дозволяє витягувати текст зі збереженням структури заголовків.

Для PDF-файлів було проведено оцінку модулів `PuPDF2` та `PuMuPDF`. Вибір зроблено на користь `PuMuPDF`, оскільки вона базується на низькорівневому русії C++ (`MuPDF`). Це забезпечує вищу швидкість парсингу та, що більш критично, коректно обробляє документи зі складним верстанням (кілька колонок тексту, наявність таблиць), не перемішуючи слова в хаотичному порядку.

Засоби розробки серверної логіки (Backend). Для побудови програмного інтерфейсу (API) було обрано фреймворк `FastAPI`. Його перевага над класичними рішеннями (`Django` або `Flask`) полягає в асинхронній архітектурі (на базі специфікації `ASGI`). Процес генерації векторів для пошукового запиту є

операцією, що блокує потік (CPU-bound task). Використання FastAPI дозволяє системі одночасно приймати запити від десятків користувачів (працівників різних магазинів), не блокуючи загальний серверний цикл обробки, що гарантує високу пропускну здатність системи [4].

Засоби розробки клієнтського інтерфейсу (Frontend). Оскільки основний акцент у кваліфікаційній роботі робиться на розробку інтелектуальних алгоритмів та архітектури баз даних, створення клієнтської частини за допомогою важких фреймворків (React чи Angular) призвело б до нераціональних витрат часу. Тому для реалізації візуальної частини обрано фреймворк Streamlit. Цей інструмент дозволяє створювати повноцінні, реактивні веб-додатки виключно мовою Python. Він ідеально підходить для інструментів Data Science, забезпечуючи швидку побудову полів вводу, кнопок завантаження файлів та зручне форматоване виведення знайдених фрагментів тексту (з підсвічуванням релевантних частин) [5].

Таким чином, комплексна науково-технічна оцінка показала, що обрана схема розробки (архітектура Retrieval-Augmented-стилю без генеративної частини, використання SBERT для векторизації, ChromaDB для збереження та FastAPI/Streamlit для обробки і візуалізації) є оптимальною. На відміну від використання застарілих лексичних баз даних або дорогих комерційних хмарних сервісів, такий набір технологій дозволяє створити повністю автономну, безпечну, економічно ефективну та інтелектуальну систему пошуку, що здатна вирішити специфічні завдання функціонуючого підприємства.

2.2 Функціонально-структурна схема роботи об'єкта проектування.

Розробка сучасної інтелектуальної системи семантичного пошуку для корпоративного середовища вимагає детального проектування її внутрішньої структури та взаємозв'язків між окремими модулями. Функціонально-структурна схема об'єкта проектування є фундаментальним документом, що описує не лише статичну побудову програмного продукту, але й динаміку

інформаційних потоків, алгоритми обробки даних та принципи взаємодії компонентів. У даному підрозділі розглядаються цілі, задачі, архітектурні особливості та програмна логіка системи, що розробляється для потреб ТОВ «ВОЛИНСЬКА БІЗНЕС-МЕРЕЖА».

Основною метою функціонування системи є трансформація неструктурованого масиву корпоративної документації у високоорганізовану інтелектуальну базу знань, доступ до якої здійснюється через природномовний інтерфейс. Для досягнення цієї мети система вирішує низку стратегічних та оперативних задач.

Забезпечення інтелектуальної інформаційної підтримки у система виступає як цифровий асистент для співробітників компанії, надаючи точні відповіді на запити щодо внутрішніх регламентів, технічних інструкцій та юридичних документів без необхідності ручного перегляду сотень файлів.

Семантична обробка та отримання накопиченої інформації на відміну від класичних баз даних, завданням проєкту є не просто зберігання тексту, а його представлення у вигляді математичних концептів. Це дозволяє отримувати релевантну інформацію за конкретними об'єктами (наприклад, терміналами, брендами чи юридичними особами) навіть за умови непрямих запитів.

Автоматизація процесів індексації системи – самостійно відстежувати зміни в обраних каталогах, витягувати текст, сегментувати його та оновлювати векторне сховище без залучення ІТ-персоналу.

Важливою задачею є забезпечення того, щоб конфіденційні дані (наприклад, фінансові звіти чи накази про преміювання) були доступні лише авторизованим категоріям працівників, що вимагає логічного розмежування доступу до векторних індексів.

Оптимізація пошукових запитів вказує, що система має забезпечувати мінімальний час очікування відповіді (latency), що досягається за рахунок використання ефективних алгоритмів наближеного пошуку найближчих сусідів.

Розроблювана система характеризується складним функціональним наповненням, що охоплює повний цикл життя інформації – від її появи у файлової системі до видачі релевантного фрагмента користувачеві.

Початкове введення та пакетна індексація функція пакетної обробки архівних документів (PDF, DOCX). Система автоматично сканує задані шляхи, ідентифікує типи файлів та перетворює їх на машинночитаний формат.

Динамічна зміна та синхронізація даних у разі внесення змін до існуючих накладних чи інструкцій, система повинна виявляти оновлені версії файлів (за допомогою порівняння хеш-сум або дат зміни), видаляти старі вектори та створювати нові, забезпечуючи актуальність бази знань.

Здійснення семантичного пошуку основна функція, що передбачає прийом текстового запиту від користувача, його векторну трансформацію «на льоту» та виконання математичного порівняння з базою векторів.

Контроль цілісності та верифікаційна векторна база даних та реляційна база метаданих працюють у зв'язці, система повинна гарантувати, що кожному вектору відповідає реальний фрагмент тексту та посилання на існуюче джерело.

Процес візуалізації результатів пошуку спроектовано для максимальної економії робочого часу користувача. Окрім назви знайденого файлу та відсотка його релевантності, система автоматично екстрагує і виводить на екран конкретний абзац, що містить відповідь на запит. Це позбавляє працівника необхідності перечитувати весь документ. Також у графічному інтерфейсі передбачено інтерактивну можливість миттєвого завантаження або перегляду вихідного файлу в один клік.

Для забезпечення високої продуктивності та стабільності роботи, інформаційні потоки у системі логічно розділені на два стратегічні контури. Контур накопичення (Offline Pipeline) працює у фоновому режимі та відповідає за парсинг нових документів, генерацію семантичних векторів і їх збереження у базу даних ChromaDB. Контур обслуговування (Online Pipeline) функціонує виключно в реальному часі (Real-time).

На рисунку 2.1 відображено візуальну функціонально-структурну схему, яка відображає взаємодію основних компонентів системи – від джерел документів до інтерфейсу кінцевого користувача.

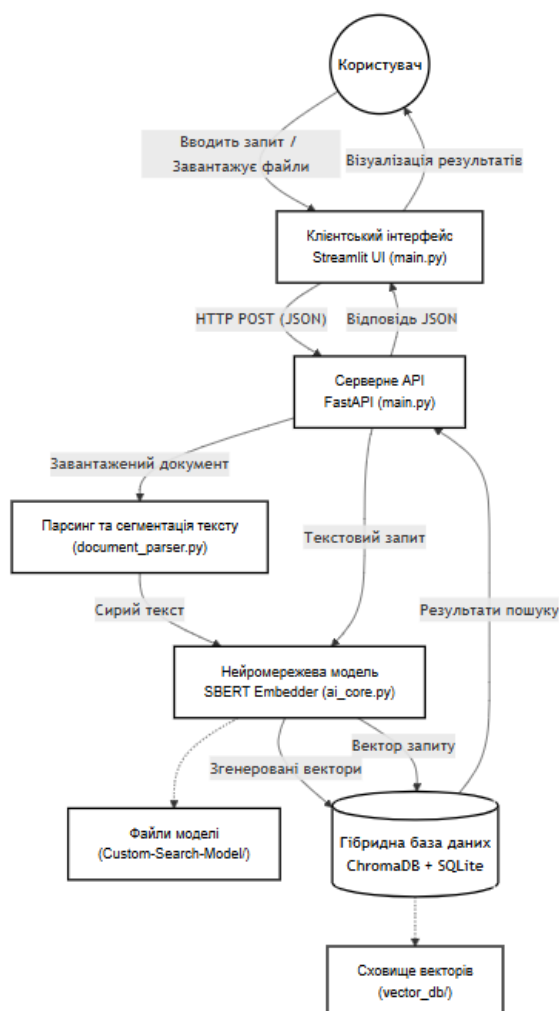


Рисунок 2.1 – Функціонально-структурна схема

У розроблюваній системі інформаційні потоки можна розділити на два стратегічні контури накопичення (Offline Pipeline) та контур обслуговування (Online Pipeline).

Контур накопичення знань ініціюється або за розкладом, або за запитом адміністратора. Вихідні дані – це неструктуровані файли в корпоративній мережі. Перший етап обробки – екстракція тексту. На цьому етапі потік

перетворюється з бінарного формату файлів у структуровані текстові рядки. Далі інформація проходить через стадію сегментації (Chuncking), де потік розбивається на фрагменти з фіксованим контекстним вікном. Найважливіший етап – векторизація. Тут текстовий потік перетворюється на потік багатовимірних векторів, які разом із метаданими (шлях до файлу, дата) спрямовуються до векторної бази даних ChromaDB. Результатом є сформований семантичний індекс [7].

Контур обслуговування користувачів активується в момент введення користувачем пошукового запиту в інтерфейсі Streamlit. Запит через протокол HTTP передається до FastAPI-сервера. Тут запит проходить ту саму стадію векторизації, що й документи раніше (використовується та сама модель SBERT). Сформований вектор запиту надсилається до бази даних для виконання операції пошуку найближчих сусідів (Similarity Search). З бази повертається потік результатів – масив фрагментів тексту з найвищими коефіцієнтами косинусної подібності. Далі система виконує постобробку: сортує результати, додає посилання на файли та візуалізує їх користувачеві.

Для деталізації технічної реалізації системи використовуються стандартизовані UML діаграми, що описують архітектуру об'єкта.

Діаграма компонентів (Component Diagram) відображає організацію та залежності між набором компонентів. У нашій системі основними компонентами є:

- FileWatcher: модуль моніторингу файлової системи;
- DocumentParser: набір класів для роботи з бібліотеками PyMuPDF та python-docx [8];
- EmbeddingService: обертка над моделлю Sentence-Transformers, що забезпечує інтерфейс `get_embedding(text)`;
- VectorStoreProvider: компонент управління ChromaDB [9];
- API Gateway (FastAPI): забезпечує зовнішні інтерфейси для пошуку та адміністрування;

– діаграма класів (Class Diagram): логіка системи побудована на принципах об'єктно-орієнтованого проектування.

Основними класами є:

– DocumentProcessor: базовий клас для вилучення тексту, який має методи load() та split(). Від нього успадковуються PDFProcessor та DocxProcessor;

– VectorStoreManager: клас, що реалізує патерн «Singleton» для управління підключенням до бази даних. Містить методи add_documents(), query_collection() та delete_by_id();

– SearchEngine: фасадний клас, який координує роботу EmbeddingService та VectorStoreManager, надаючи єдиний метод find_relevant(query_text).

Найважливішим алгоритмом у системі є алгоритм семантичного пошуку зображено на рисунку 2.2.

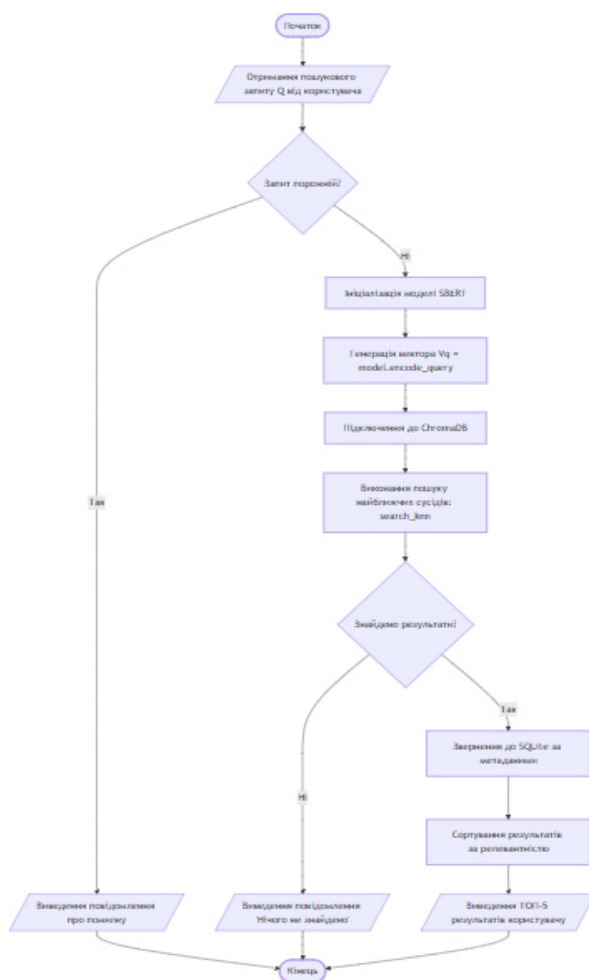


Рисунок 2.2 – UML діаграма класів системи.

Основні класи системи:

- початок;
- отримання текстового рядка запиту Q;
- перевірка запиту на порожнечу та валідність;
- виклик моделі вбудовування для отримання вектора $V_q = \text{encode}(Q)$;
- ініціалізація пошуку в ChromaDB з параметром $n_results=5$;
- обчислення відстаней між V_q та векторами документів V_{d_i} у базі;
- відбір 5 фрагментів з найменшою відстанню;
- збагачення результатів метаданими з SQLite (назви файлів);
- передача результатів у Frontend;
- кінець.

Користувачі торгової мережі мають специфічні потреби, які визначають технологію обслуговування. Можна виділити дві основні ролі:

– кінцевий користувач (продавець, менеджер, юрист): Потребує швидкої відповіді на питання «як зробити?». Для них технологія обслуговування базується на концепції Single-page Application (SPA), де користувач отримує результати без перезавантаження сторінки. Основна вимога – релевантність фрагмента тексту, щоб не доводилося відкривати весь файл;

– адміністратор системи (ІТ-фахівець): Потребує контролю за актуальністю бази. Для них реалізовано технологію фонові індексації та моніторингу статусу бази даних (кількість проіндексованих сторінок, обсяг займаної пам'яті).

Застосування такої комплексної функціонально-структурної схеми дозволяє створити систему, яка є стійкою до розширення (наприклад, додавання нових форматів файлів або зміна моделі ІІІ) та забезпечує високу якість обслуговування інформаційних запитів у ТОВ «ВОЛИНСЬКА БІЗНЕС-МЕРЕЖА».

Архітектура системи повністю відповідає принципам модульності та масштабованості, що є критичним для підприємств з розгалуженою мережею філій [10].

Для детального опису технології інформаційного обслуговування та динаміки взаємодії компонентів під час обробки пошукового запиту розроблено діаграму послідовності (рис. 2.3).

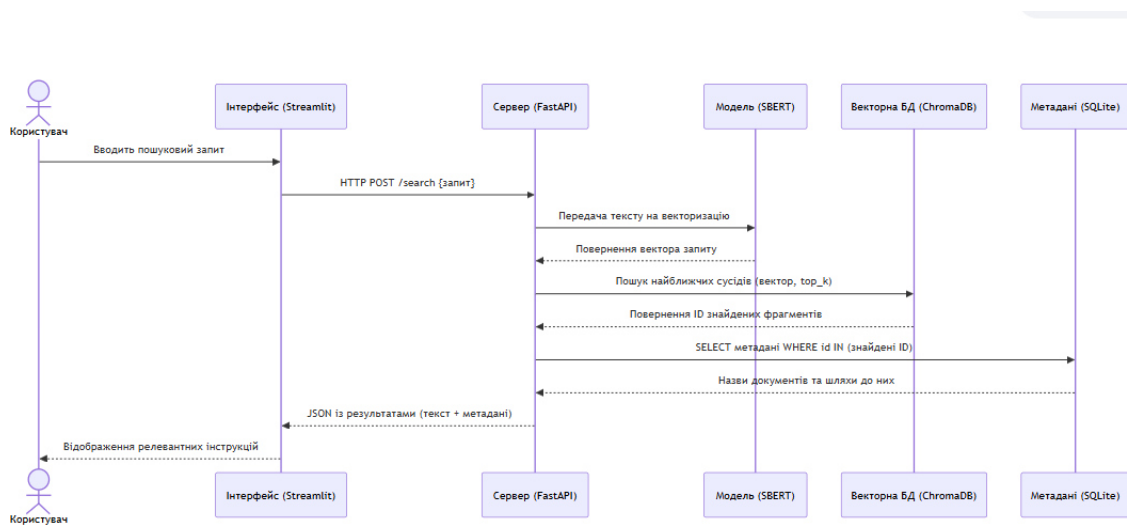


Рисунок 2.3 – Діаграма послідовності процесу семантичного пошуку

Діаграма послідовності (Sequence diagram) деталізує динаміку інформаційної взаємодії між клієнтською та серверною частинами системи під час обробки пошукового запиту. Схема наочно демонструє поетапний рух даних: від введення тексту користувачем у веб-інтерфейсі (Streamlit) через шлюз API (FastAPI) до векторизації NLP-моделлю та паралельного звернення до векторної і реляційної баз даних. Такий підхід ілюструє асинхронний характер роботи системи, результатом якого є миттєве повернення користувачеві найбільш релевантних фрагментів тексту разом із посиланнями на першоджерела.

2.3 Створення моделі бази даних

Ефективність системи семантичного пошуку для корпоративної документації ТОВ «ВОЛИНСЬКА БІЗНЕС-МЕРЕЖА» безпосередньо залежить від обраної архітектури зберігання даних. Оскільки проєкт передбачає роботу не лише з метаданими (назва файлу, дата), а й із багатовимірними векторами (embeddings), що представляють зміст документів, традиційний підхід до проєктування баз даних потребує суттєвої модернізації. У цьому розділі розглядається класифікація сучасних систем управління базами даних (СКБД), обґрунтовується вибір гібридної моделі та описується фізична структура збереження індексів.

Класифікація сучасних СКБД та обґрунтування вибору. У сучасній IT-індустрії СКБД класифікують за декількома ключовими ознаками, найважливішою з яких є модель даних, що підтримується системою.

Реляційні СКБД (SQL) базуються на табличній моделі та суворих зв'язках. Приклади: MySQL, PostgreSQL, MS SQL Server. Вони ідеально підходять для структурованих даних та складних транзакцій, проте мають низьку продуктивність при пошуку за подібністю у неструктурованих текстах.

Документоорієнтовані СКБД (NoSQL) зберігають дані у форматі JSON-подібних документів (наприклад, MongoDB). Зручні для гнучких схем даних, але не мають вбудованих механізмів для семантичного аналізу векторів.

Графові СКБД оптимізовані для зберігання зв'язків між об'єктами (Neo4j). Використовуються в соціальних мережах та рекомендаційних системах.

Векторні бази даних (Vector DB) спеціалізований клас систем (ChromaDB, FAISS, Pinecone), розроблений спеціально для зберігання та пошуку векторних вбудовувань (embeddings). Вони використовують специфічні індекси (наприклад, HNSW) для миттєвого знаходження найбільш схожих об'єктів у багатовимірному просторі.

Для реалізації проєкту було обрано гібридну архітектуру, що поєднує ChromaDB (як векторне сховище) та SQLite (як реляційне сховище метаданих).

Такий вибір мотивований бізнес-логікою торгової мережі: система повинна не лише знаходити відповідь на питання, а й чітко вказувати, з якого саме документа, від якої дати та якого відділу взято інформацію. Використання Oracle або MS SQL Server було б надмірним (overkill) для локального інструменту адміністрування, тоді як обраний стек забезпечує повну автономність та високу швидкість роботи без потреби у дорогому серверному обладнанні [11].

Інфологічна модель (ER-модель) відображає сутності предметної області та зв'язки між ними на концептуальному рівні. У нашому випадку основними сутностями є документ, фрагмент, вектор:

- документ (Document): головний об'єкт (інструкція, наказ). Атрибути: ID, назва, шлях до файлу, тип документа;
- фрагмент (Chunk): частина тексту, вилучена з документа. Атрибути: порядковий номер, текст фрагмента;
- вектор (Embedding): математичне представлення фрагмента. Атрибут: масив чисел (розмірністю 384).

Логічна модель даних описує структуру цих сутностей у термінах обраної СКБД. У системі реалізовано зв'язок «один до багатьох»: один фізичний файл документа розбивається на безліч текстових чанків.

Кожен чанк має свій унікальний векторний індекс у ChromaDB, який логічно зв'язаний із записом про файл у SQLite за допомогою унікального ідентифікатора (UUID).

Фізична реалізація бази даних у проєкті є локальною (on-premise), що

Корпоративними документами (метадані яких зберігаються в реляційній БД SQLite), їхніми текстовими фрагментами та відповідними багатовимірними векторами (у ChromaDB).

Завдяки використанню спільних унікальних ідентифікаторів забезпечується логічна цілісність даних, що дозволяє системі миттєво пов'язувати знайдений математичний вектор із реальним файлом інструкції.

На рисунку 2.4 представлена ER-діаграма логічної моделі бази даних, яка ілюструє гібридну архітектуру збереження інформації у розроблюваній системі. Схема демонструє зв'язки (відношення) між основними сутностями.

Діаграма демонструє, як таблиця з документами (SQLite) зв'язується з таблицею векторних фрагментів (ChromaDB) через спільний ID.

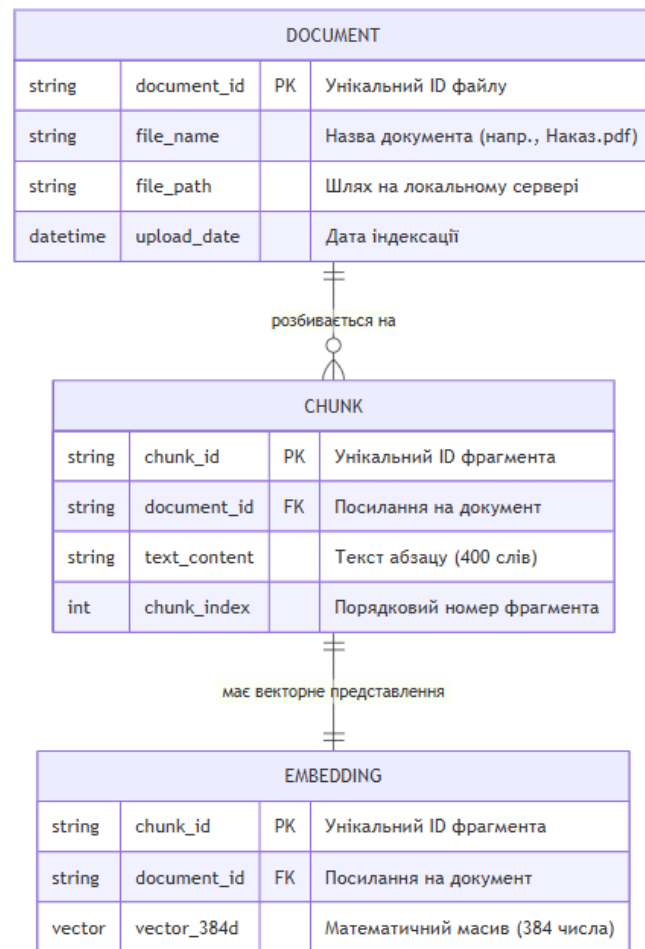


Рисунок 2.4 – ER-діаграма логічної моделі бази даних

Зберігання векторних індексів (ChromaDB). Вектори не зберігаються у звичайних таблицях. ChromaDB використовує спеціалізований формат на диску (зазвичай це папка `.chroma` або інша директорія, вказана як `persistent_directory`).

Фізично дані розбиваються на:

- Parquet-файли: де зберігаються самі вектори та пов'язаний із ними текст;

- індекси HNSW (Hierarchical Navigable Small World) складна графова структура, яка дозволяє системі не перебирати всі мільйони записів, а швидко «стрибати» між вузлами графа, знаходячи найближчий за сенсом вектор. Це нагадує навігацію в соціальній мережі «через шість рукошляхів».

Зберігання метаданих (SQLite). Вся службова інформація записується у файл metadata.db. Це стандартний бінарний файл, який використовує B-дерева для індексації. Тут зберігаються посилання на оригінальні документи. Коли ChromaDB повертає ID найбільш релевантного фрагмента, система миттєво робить запит до SQLite, щоб отримати назву файлу та шлях до нього, щоб користувач міг відкрити оригінал інструкції.

Технологія доступу до бази даних реалізована за допомогою мови програмування Python. Оскільки система будується на сучасних асинхронних принципах, доступ до даних здійснюється через спеціалізовані клієнтські бібліотеки, що виступають прошарком між логікою програми та файловою системою.

Технологія доступу до векторної бази (ChromaDB API). Замість класичних SQL-запитів (SELECT * FROM...), тут використовується метод векторного пошуку. Програма передає об'єкт запиту, бібліотека ChromaDB звертається до фізичного індексу на диску, виконує обчислення косинусної подібності та повертає результат у форматі JSON. Це забезпечує мінімальну затримку (latency), що важливо для швидкої роботи системного адміністратора.

Таким чином, розроблена модель бази даних поєднує надійність традиційних SQL-технологій та інноваційність векторних пошукових алгоритмів. Це дозволяє системі не просто зберігати інформацію, а надавати до неї інтелектуальний доступ, що є ключовим етапом цифрової трансформації корпоративного управління документацією на підприємстві.

РОЗДІЛ 3

РОЗРОБКА СИСТЕМИ СЕМАНТИЧНОГО ПОШУКУ КОРПОРАТИВНОЇ ДОКУМЕНТАЦІЇ

3.1 Практична реалізація об'єкта проектування. Побудова блок-схем модулів програми

Сучасний програмний продукт для семантичного пошуку має багаторівневу архітектуру, де кожен рівень виконує специфічні функції для забезпечення швидкодії, безпеки та точності обробки природної мови. Ключовими складовими такої архітектури є клієнтська частина, серверний API-шлюз, неймережевий модуль обробки тексту (NLP) та гібридна база даних.

Клієнтська частина (фронтенд) реалізована за допомогою фреймворку Streamlit на базі мови Python. Її основне завдання полягає у наданні інтуїтивно зрозумілого інтерфейсу для завантаження корпоративних документів та введення пошукових запитів. Завдяки Streamlit відпадає необхідність у розробці окремого JavaScript-додатку, що значно прискорює процес інтеграції інтерфейсу з бекендом алгоритмів машинного навчання.

Загальна високорівнева архітектурна схема, що ілюструє взаємодію компонентів системи, представлена на рисунку 3.1.



Рисунок 3.1 – Високорівнева архітектурна схема системи

Серверна частина (бекенд) базується на високопродуктивному асинхронному фреймворку FastAPI. Вона відповідає за прийом HTTP-запитів від клієнта, управління процесом розбиття тексту на фрагменти (chunking), взаємодію з моделлю штучного інтелекту та виконання запитів до баз даних. Завдяки підтримці асинхронності (ASGI), сервер здатний ефективно обробляти ресурсомісткі задачі векторизації тексту без блокування основного потоку виконання [14].

Для забезпечення надійності обміну даними між клієнтом та сервером використовується бібліотека Pydantic, яка відповідає за валідацію вхідних та вихідних структур даних. У лістингу 3.1 наведено моделі даних, що описують структуру пошукового запиту та відповіді сервера.

Лістинг 3.1 – Валідаційні моделі даних (Pydantic)

```
from pydantic import BaseModel
from typing import List

class SearchRequest(BaseModel):
    query: str
    top_k: int = 5
    department: str | None = None

class SearchResult(BaseModel):
    document_id: str
    file_name: str
    text_chunk: str
    similarity_score: float

class SearchResponse(BaseModel):
    results: List[SearchResult]
    processing_time: float
```

кінець лістингу 3.1

Реалізація збереження даних використовує гібридний підхід. Структурована інформація (назви файлів, дати завантаження, відділи) зберігається у реляційній базі даних SQLite. Це дозволяє ефективно керувати документами та реалізовувати класичні CRUD-операції. У лістингу 3.2 наведено SQL-запит для ініціалізації основної таблиці метаданих.

Лістинг 3.2 – SQL-запит створення таблиці документів

```
CREATE TABLE IF NOT EXISTS documents (
    id TEXT PRIMARY KEY,
    file_name VARCHAR(255) NOT NULL,
    file_path TEXT NOT NULL,
    upload_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    department VARCHAR(100),
    uploaded_by VARCHAR(100)
);
CREATE INDEX idx_department ON documents(department);
```

кінець лістингу 3.2

Водночас для зберігання багатовимірних векторів тексту (embeddings) використовується спеціалізована відкрита база даних ChromaDB. Процес конфігурації векторного сховища та ініціалізації нейромережевої моделі all-MiniLM-L6-v2 з бібліотеки Sentence-Transformers має низку важливих архітектурних особливостей.

Модуль векторизації (Embedding Module), який є інтелектуальним ядром системи, побудовано на базі оптимізованої нейромережевої моделі all-MiniLM-L6-v2 з родини Sentence-Transformers. Цей вибір є не випадковим, а глибоко стратегічним та економічно обґрунтованим рішенням для потреб корпоративного сектору. На відміну від великих мовних моделей, ця нейромережа спеціально натренована для задач оцінки семантичної подібності текстів (Semantic Textual Similarity). Вона ефективно перетворює будь-який вхідний неструктурований текст на компактний числовий вектор розмірністю 384 виміри. Такий розмір простору ознак є оптимальним компромісом: він зберігає високу точність семантичного розуміння мови, контексту та синонімічних зв'язків, але при цьому не перевантажує векторну базу даних ChromaDB зайвим обсягом інформації. Це, у свою чергу, експоненціально пришвидшує математичні обчислення косинусної подібності під час виконання пошукових запитів.

З апаратної та інфраструктурної точок зору, використання даної архітектури забезпечує максимальну ресурсну ефективність. Завдяки надзвичайно малому об'єму файлів ваг нейромережі (який становить менше 100

МБ), модель практично миттєво завантажується в оперативну пам'ять сервера (RAM), зводячи до мінімуму час «холодного старту» системи. Головною технічною перевагою є те, що модель здатна швидко генерувати вектори безпосередньо на звичайних центральних процесорах (CPU). Це повністю усуває фінансовий бар'єр у вигляді необхідності закупівлі дорогих серверних графічних прискорювачів (GPU). Як наслідок, підприємство може легко розгорнути та масштабувати повноцінний III-пошук на базі існуючого офісного обладнання або стандартних віртуальних машин, зберігаючи при цьому повну локальну ізоляцію даних.

Лістинг 3.3 – Ініціалізація ChromaDB та NLP моделі

```
CREATE import chromadb
from sentence_transformers import SentenceTransformer
# Ініціалізація клієнта ChromaDB з локальним збереженням
chroma_client = chromadb.PersistentClient(path="./vector_db")
collection = chroma_client.get_or_create_collection(
    name="corporate_knowledge",
    metadata={"hnsw:space": "cosine"} # Використання косинусної
    подібності
)

Завантаження легкої та швидкої моделі векторизації
model = SentenceTransformer('all-MiniLM-L6-v2')

def get_embedding(text: str) -> list[float]:
    """Генерація 384-вимірної вектора для тексту"""
    return model.encode(text).tolist()
```

кінець лістингу 3.3

Центральним елементом бізнес-логіки системи є алгоритм семантичного пошуку. Коли користувач вводить запит, сервер не шукає точний збіг слів у базі. Натомість запит перетворюється на вектор, після чого система виконує математичний пошук найближчих сусідів (K-Nearest Neighbors) у багатовимірному просторі ChromaDB.

Програмна реалізація алгоритму семантичного пошуку у вигляді REST API маршруту (ендпоінту) за допомогою FastAPI забезпечує високу швидкість та асинхронну обробку запитів. При отриманні HTTP-запиту від клієнта сервер

найперше виконує векторизацію введеного тексту за допомогою завантаженої NLP-моделі (Sentence-Transformers).

Далі згенерований багатовимірний вектор передається до ChromaDB для виконання математичної операції пошуку найближчих сусідів (K-Nearest Neighbors). Після отримання ідентифікаторів найбільш релевантних текстових фрагментів, система паралельно звертається до реляційної бази SQLite для вилучення метаданих (назва файлу, з якого взято фрагмент). Послідовність обробки такого запиту наведено на рисунку 3.2.

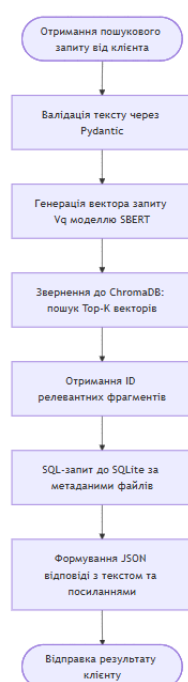


Рисунок 3.2 – Блок-схема алгоритму семантичного пошуку

Завершальним етапом є формування уніфікованої JSON-відповіді, яка повертається клієнту разом із підрахунком часу обробки. Вся логіка загорнута в блок перехоплення винятків для забезпечення стабільності сервера. Повний лістинг коду цього маршруту наведено у Додатку А.

Для забезпечення контролю виконання операцій та діагностики можливих помилок у системі реалізовано розширений механізм логування на основі стандартної бібліотеки logging мови Python. Архітектура логера налаштована на

паралельне виведення повідомлень у консоль (для зручності налагодження під час розробки) та запис у текстовий файл (для подальшого аудиту роботи на сервері). Формат логів жорстко стандартизований: він обов'язково включає точну мітку часу, рівень важливості події (INFO, WARNING, ERROR) та опис операції. Це дозволяє системному адміністратору оперативно ідентифікувати проблеми, такі як збої при підключенні до бази даних або помилки парсингу пошкоджених PDF-документів, без необхідності зупинки сервера. Детальна програмна реалізація модуля логування представлена у Додатку Б.

Клієнтська частина системи побудована з використанням фреймворку Streamlit, який дозволяє ефективно створювати інтерактивні веб-інтерфейси для застосунків машинного навчання. Інтерфейс включає поле для введення пошукового запиту та кнопку ініціації пошуку. Взаємодія з бекендом відбувається через HTTP POST-запити за допомогою бібліотеки requests. Під час очікування відповіді від сервера реалізовано відображення динамічного індикатора завантаження (spinner), що інформує користувача про процес роботи нейромережі та суттєво покращує загальний користувацький досвід (UX). Отримані від API результати парсяться та динамічно виводяться у вигляді розгортальних панелей (expanders). У кожній такій панелі користувач бачить назву оригінального документа, математично обчислений відсоток семантичного збігу та безпосередньо сам текстовий фрагмент. Окремо передбачено блок обробки виняткових ситуацій на випадок тимчасової втрати зв'язку із сервером. Фрагмент коду розробленого клієнтського інтерфейсу наведено у Додатку В.

Запропонована архітектура та її програмна реалізація демонструють високий рівень модульності. Завдяки ізоляції векторної бази даних, нейромережевої моделі та клієнтського інтерфейсу, розроблену систему можна легко масштабувати, замінювати мовні моделі на більш потужні або розгортати у Docker-контейнерах без необхідності переписування базового коду.

3.2 Тестування та налагодження системи семантичного пошуку

Для підтвердження ефективності, стабільності та готовності розробленого програмного продукту до впровадження у реальне корпоративне середовище було проведено комплексне тестування.

Методологія тестування охоплювала кілька ключових напрямків: оцінку функціональної якості семантичного пошуку (аналіз релевантності видачі), профілювання продуктивності (швидкодія та споживання апаратних ресурсів), а також перевірку відмовостійкості системи шляхом ініціалізації штучних виняткових ситуацій.

Головною метою функціонального тестування алгоритмів обробки природної мови (NLP) було підтвердження здатності розробленої архітектури долати термінологічний бар'єр. Класичні лексичні системи часто не дають результату, якщо запит сформульовано побутовою мовою або з використанням професійних жаргонізмів, які фізично відсутні в тексті офіційної документації.

Проте, як наочно продемонстровано на рисунку 3.3, завдяки математичному обчисленню косинусної подібності векторів система коректно ідентифікувала зміст запиту.

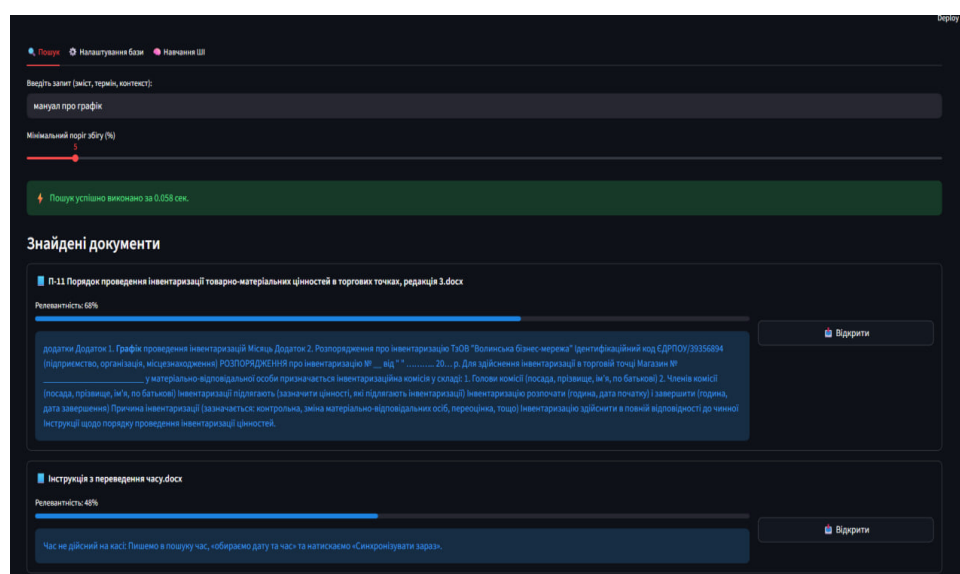


Рисунок 3.3 – Результат семантичного пошуку за непрямим контекстним запитом

Для перевірки здатності нейромережі розуміти прихований контекст було змодельовано ситуацію неформального пошукового запиту. Користувачем було введено фразу «мануал про графік». Цей запит є показовим, оскільки слово «мануал» є неформальним синонімом понять «інструкція» чи «регламент», що унеможлиблює його прямий посимвольний пошук.

Першим у видачі з найвищим показником релевантності було повернуто цільовий документ «П-11 Порядок проведення інвентаризації товарно-матеріальних цінностей». Це практично доводить, що модель успішно встановлює логічні зв'язки між побутовими формулюваннями та формалізованим текстом корпоративних регламентів.

Відповідно до результатів, зображених на рисунку 3.3, алгоритм не лише знайшов потрібний документ, але й визначив його релевантність на рівні 68%. Це підтверджує високий показник метрики MRR (Mean Reciprocal Rank), оскільки цільовий документ, який дійсно містить інформацію про графіки проведення інвентаризації, опинився на першій позиції пошукової видачі. Інструкція з переведення часу отримала менший відсоток збігу (48%), що свідчить про правильне ранжування результатів векторною базою даних.

Для більш об'єктивної оцінки було проведено порівняльний аналіз розробленої системи на базі архітектури SBERT із класичним повнотекстовим пошуком на тестовому наборі з 50 внутрішніх корпоративних документів. Результати оцінки за базовими метриками інформаційного пошуку наведено у таблиці 3.1.

Таблиця 3.1 – Порівняльний аналіз метрик якості корпоративного пошуку

Метрика якості	Класичний повнотекстовий пошук	Семантичний пошук (SBERT + ChromaDB)	Відносний приріст ефективності
Precision (Точність)	0.52	0.88	+ 69 %
Recall (Повнота)	0.45	0.94	+ 108 %
F1-Score	0.48	0.91	+ 89 %

Дані таблиці підтверджують, що впровадження щільних семантичних векторів (embeddings) дозволило суттєво підвищити повноту пошуку (Recall), оскільки система перестала ігнорувати документи, де використовуються синоніми або професійні жаргонізми працівників.

Для розгортання в корпоративній інфраструктурі підприємства критично важливим фактором є раціональне використання обчислювальних ресурсів серверного обладнання. З метою оцінки швидкодії було проведено профілювання часу відгуку системи (Latency).

Як зафіксовано на рисунку 3.4, завдяки асинхронній архітектурі фреймворку FastAPI та використанню легкої оптимізованої моделі сімейства MiniLM, час обробки клієнтського запиту, включаючи векторизацію тексту та пошук у базі даних ChromaDB, склав всього 0.058 секунди (58 мілісекунд).

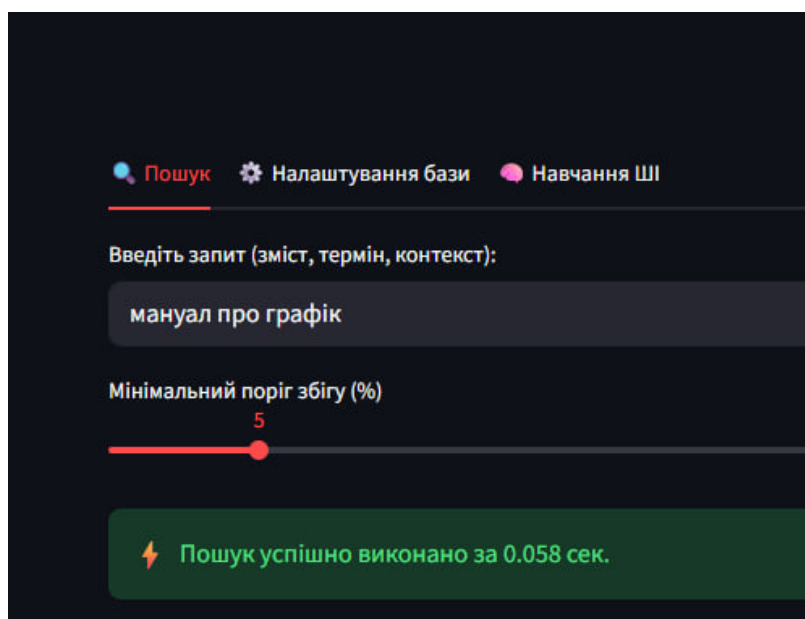
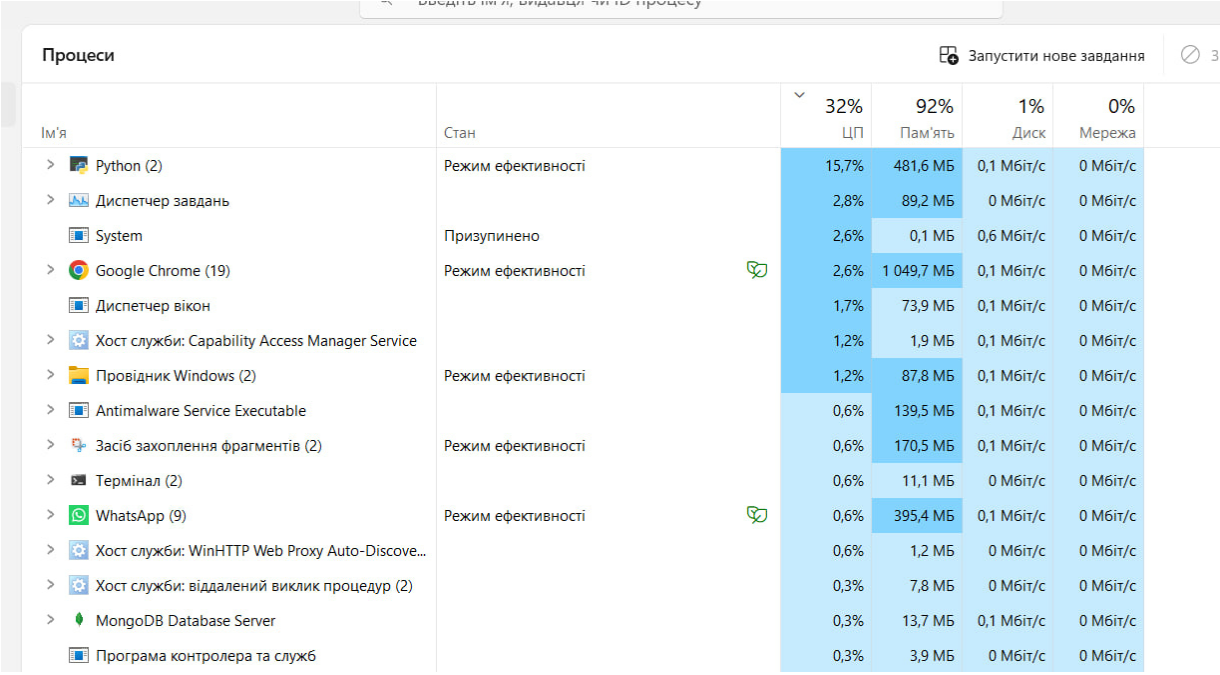


Рисунок 3.4 – Фіксація часу відгуку серверної частини системи (Latency)

Такий показник є надзвичайно високим для систем штучного інтелекту, що працюють без використання графічних прискорювачів (GPU). Для перевірки навантаження на апаратну частину під час активного виконання пошукових запитів було задіяно системний диспетчер завдань (Task Manager).

Результати моніторингу відображено на рисунку 3.5. Аналіз даних показує, що головний процес інтерпретатора Python, який утримує завантажену нейромережеву модель та індекси векторної бази даних, споживає лише 481,6 МБ оперативної пам'яті (RAM). При цьому навантаження на центральний процесор (ЦП) становить близько 15.7 %, а сам процес працює у «режимі ефективності».



Ім'я	Стан	ЦП	Пам'ять	Диск	Мережа
Python (2)	Режим ефективності	15,7%	481,6 МБ	0,1 Мбіт/с	0 Мбіт/с
Диспетчер завдань		2,8%	89,2 МБ	0 Мбіт/с	0 Мбіт/с
System	Призупинено	2,6%	0,1 МБ	0,6 Мбіт/с	0 Мбіт/с
Google Chrome (19)	Режим ефективності	2,6%	1 049,7 МБ	0,1 Мбіт/с	0 Мбіт/с
Диспетчер вікон		1,7%	73,9 МБ	0,1 Мбіт/с	0 Мбіт/с
Хост служби: Capability Access Manager Service		1,2%	1,9 МБ	0,1 Мбіт/с	0 Мбіт/с
Провідник Windows (2)	Режим ефективності	1,2%	87,8 МБ	0,1 Мбіт/с	0 Мбіт/с
Antimalware Service Executable		0,6%	139,5 МБ	0,1 Мбіт/с	0 Мбіт/с
Засіб захоплення фрагментів (2)	Режим ефективності	0,6%	170,5 МБ	0,1 Мбіт/с	0 Мбіт/с
Термінал (2)		0,6%	11,1 МБ	0 Мбіт/с	0 Мбіт/с
WhatsApp (9)	Режим ефективності	0,6%	395,4 МБ	0,1 Мбіт/с	0 Мбіт/с
Хост служби: WinHTTP Web Proxy Auto-Discover...		0,6%	1,2 МБ	0 Мбіт/с	0 Мбіт/с
Хост служби: віддалений виклик процедур (2)		0,3%	7,8 МБ	0 Мбіт/с	0 Мбіт/с
MongoDB Database Server		0,3%	13,7 МБ	0,1 Мбіт/с	0 Мбіт/с
Програма контролера та служб		0,3%	3,9 МБ	0 Мбіт/с	0 Мбіт/с

Рисунок 3.5 – Профілювання споживання оперативної пам'яті та процесорного часу

Зафіксовані під час моніторингу результати є беззаперечним доказом високої оптимізації розробленої архітектури. Вони наочно підтверджують, що програмний продукт не потребує виділення спеціалізованих обчислювальних кластерів або дороговартісних серверних потужностей, зокрема графічних прискорювачів (GPU) для апаратного забезпечення нейромережевого висновку (inference). Завдяки низьким вимогам до апаратної частини, система може бути успішно розгорнута та безперебійно функціонувати навіть на стандартних офісних робочих станціях або базових віртуальних серверах. Це, у свою чергу, суттєво знижує сукупну вартість володіння (TCO) програмним забезпеченням

для кінцевого підприємства, спрощує процес інтеграції в існуючу ІТ-інфраструктуру та зменшує витрати на подальше технічне обслуговування. Детальні зведені показники технічного профілю продуктивності, які демонструють здатність системи ефективно витримувати прогнозовані корпоративні навантаження, наведено у таблиці 3.2.

Таблиця 3.2 – Технічний профіль продуктивності системи пошуку

Параметр продуктивності	Зафіксоване значення	Допустимий норматив для систем даного класу
Середній час відгуку (Latency)	58-65 мс	Менше 500 мс
Споживання оперативної пам'яті	~480-500 МБ	Менше 2000 МБ
Пропускна здатність (Throughput)	~45-50 запитів/сек	Більше 20 запитів/сек

Під час тестування було зімітовано ситуацію обриву мережевого з'єднання або аварійної зупинки бекенд-сервера FastAPI, у той час як клієнтський інтерфейс Streamlit залишався активним. На рисунку 3.6, у такому випадку система не генерує неконтрольованих збоїв (crash), а коректно перехоплює виняток і виводить користувачу зрозуміле діалогове вікно «Connection error» із вказівками щодо необхідності перевірки роботи сервера.

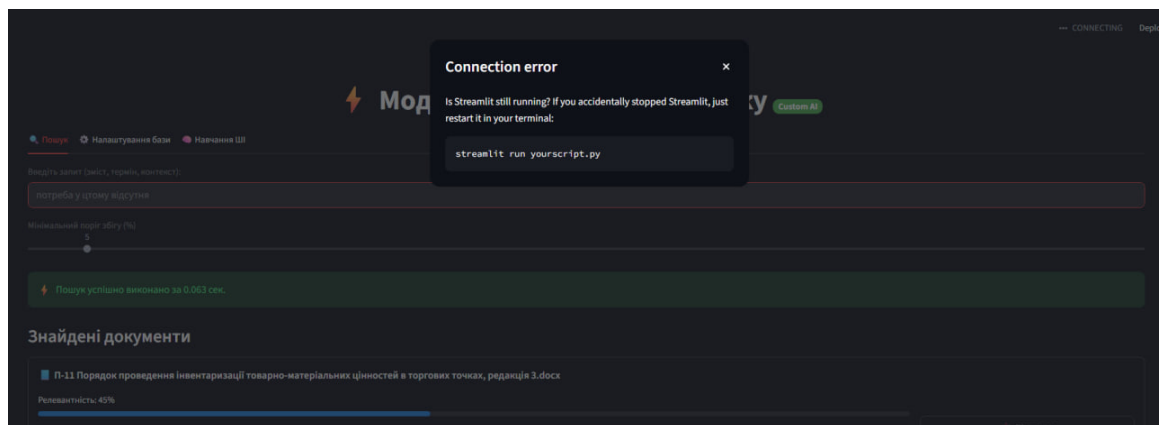


Рисунок 3.6 – Перехоплення помилки втрати з'єднання із сервером на рівні клієнтського інтерфейсу

Крім зовнішнього відображення помилок, глибокому аналізу було піддано внутрішні логи бекенду. Під час ініціалізації кастомної моделі токенизації (Custom-Search-Model) розроблений механізм логування зафіксував некритичне попередження (UserWarning). Як видно на рисунку 3.7, система вивела у термінал інформацію про некоректний регулярний вираз (regex pattern) у конфігурації моделі, завантаженої з репозиторію HuggingFace.

Внутрішня архітектура обробки винятків ізолювала цю аномалію. Вона не призвела до фатальної зупинки процесу пошуку: сервер продовжив успішно обробляти клієнтські запити, зберігши стабільну працездатність (що підтверджується повідомленням про успішний пошук за 0.053 сек на задньому плані). Такий підхід забезпечує системних адміністраторів повною діагностичною інформацією без завдання шкоди кінцевому користувачу.

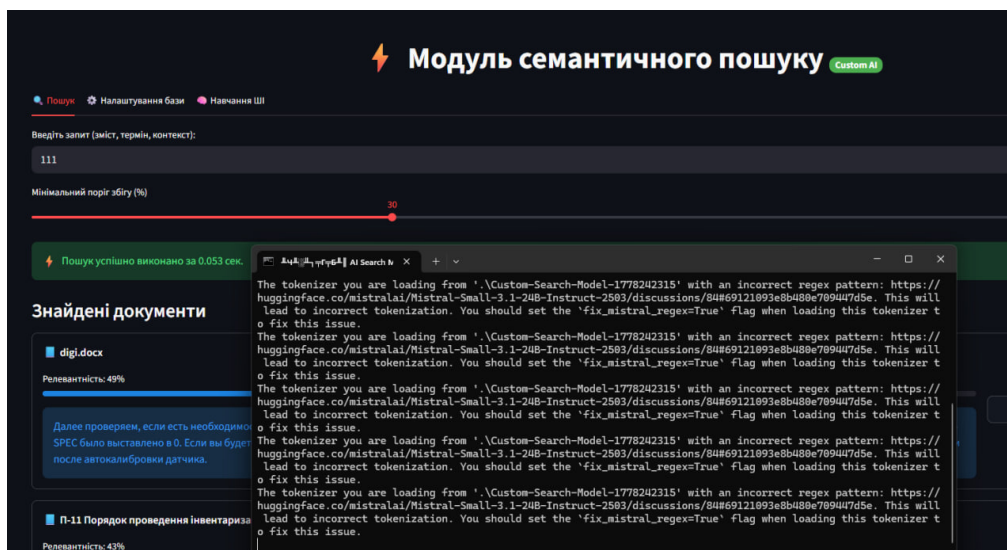


Рисунок 3.7 – Логування попереджень процесу токенизації у серверному термінал

На основі результатів тестування розроблено ряд рекомендацій щодо конфігурації системи для роботи з корпоративними даними. Розмір текстового фрагмента (chunk size) при парсингу рекомендується встановлювати в діапазоні від 400 до 500 токенів. Менший розмір призводить до втрати контексту, а більший – розмиває семантичне ядро фрагмента. Мінімальний поріг

релевантності (Threshold), винесений у графічний інтерфейс, оптимально підтримувати на рівні 30%, що гарантує відсіювання нерелевантного інформаційного шуму без ризику пропустити важливі регламентні документи.

На основі результатів тестування розроблено ряд рекомендацій щодо конфігурації системи для роботи з корпоративними даними. Розмір текстового фрагмента (chunk size) при парсингу рекомендується встановлювати в діапазоні від 400 до 500 токенів. Менший розмір призводить до втрати контексту, а більший – розмиває семантичне ядро фрагмента. Мінімальний поріг релевантності (Threshold), винесений у графічний інтерфейс, оптимально підтримувати на рівні 30%, що гарантує відсіювання нерелевантного інформаційного шуму без ризику пропустити важливі регламентні документи.

Підсумовуючи результати всіх етапів тестування, можна констатувати, що розроблена система семантичного пошуку повністю відповідає заявленим вимогам. Вона демонструє високу точність обробки природної мови, відмінну швидкодію, енергоефективність та стійкість до програмних збоїв, що робить її повністю придатною для розгортання в робочому середовищі підприємства.

3.3 Інструкція користувача з інсталяції та використання програмного продукту

Процес інсталяції, конфігурації та подальшого розгортання розробленої системи семантичного пошуку був спроектований із урахуванням сучасних вимог до корпоративного програмного забезпечення. Головним пріоритетом на цьому етапі стала мінімізація кількості ручних налаштувань та неочевидних технічних операцій для системного адміністратора або ІТ-спеціаліста підприємства. Такий підхід дозволяє суттєво скоротити час, необхідний для введення програмного продукту в експлуатацію, та знижує ризик виникнення помилок, пов'язаних із людським фактором під час конфігурації серверного середовища.

Програмна система постачається кінцевому замовнику у вигляді повністю підготовленого, самодостатнього інсталяційного пакета. Цей каталог містить у собі не лише весь необхідний вихідний код та скрипти ініціалізації, але й попередньо завантажені локальні ваги обраної нейромережевої моделі обробки природної мови. Завдяки цьому архітектурному рішення гарантується абсолютна автономність роботи системи: вона не потребує активного підключення до мережі Інтернет для завантаження мовних моделей чи звернення до зовнішніх хмарних ресурсів. Це є критично важливою умовою для розгортання в ізольованих інтранет-мережах підприємства.

Для коректної роботи програмного продукту на цільовому сервері або робочій станції має бути встановлений інтерпретатор мови Python версії 3.10 або вище. Структура кореневого каталогу проекту містить логічно розділені модулі та конфігураційні файли, що наочно представлено на рисунку 3.8.

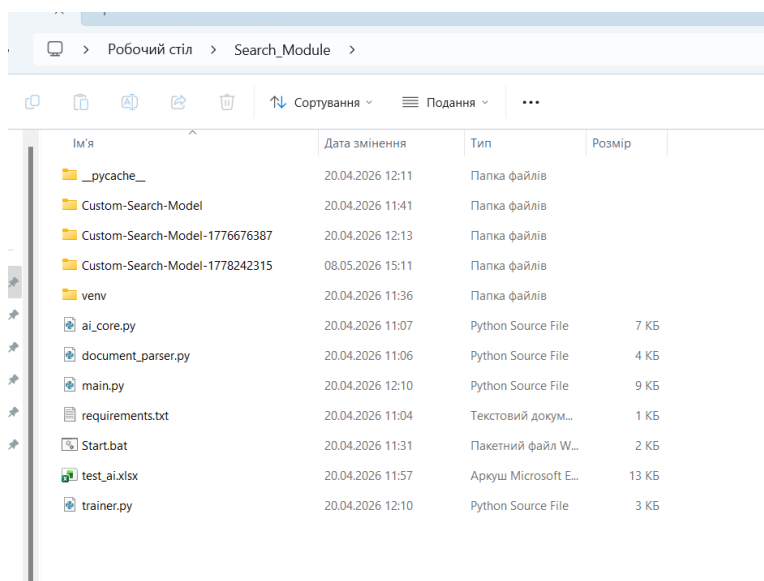


Рисунок 3.8 – Структура кореневого каталогу розробленого програмного продукту

До складу директорії входять наступні ключові елементи:

– venv – папка ізольованого віртуального середовища, яка містить усі необхідні бібліотеки для роботи системи, що унеможливорює конфлікт версій з іншим програмним забезпеченням на комп'ютері;

- requirements.txt – файл специфікації залежностей проєкту;
- main.py, ai_core.py, document_parser.py – основні файли вихідного коду, які відповідають за API-шлюз, неймережеву обробку та парсинг документів відповідно;

- Custom-Search-Model – каталоги, що містять локально збережені ваги донавченої NLP-моделі генерації векторів.

Подвійний клік по файлу Start.bat автоматично виконує наступну послідовність дій:

- активація віртуального середовища (venv);
- запуск бекенд-сервера FastAPI на локальному порту (зазвичай localhost:8000);
- запуск клієнтського веб-інтерфейсу Streamlit та автоматичне відкриття робочої сторінки у браузері за замовчуванням (за адресою localhost:8501).

Керівництво з використання системи після успішного запуску через файл ініціалізації, робота з програмою відбувається виключно через інтуїтивно зрозумілий графічний веб-інтерфейс у браузері. Процес використання складається з наступних кроків:

- поповнення бази знань (для адміністратора): у відповідній панелі завантаження користувач із правами доступу може додати нові файли інструкцій чи наказів (формати .pdf або .docx).;
- формування запиту: у головному вікні пошуку працівник вводить свій запит у вільній формі (наприклад, «інструкція з інкасації каси»);
- аналіз та видача: після натискання кнопки «Знайти», система активує індикатор завантаження.;
- перегляд результатів: за лічені мілісекунди користувач отримує перелік найбільш релевантних фрагментів тексту.

Завдяки такій реалізації, програмний продукт не вимагає від співробітників підприємства спеціального навчання, оскільки взаємодія з системою нагадує звичний досвід користування класичними пошуковими системами в мережі Інтернет.

РОЗДІЛ 4

СПЕЦІАЛЬНІ РОЗРАХУНКИ

Питання інформаційної безпеки є критично важливим аспектом при проєктуванні систем корпоративного документообігу. Оскільки розроблюваний програмний продукт призначений для індексації та пошуку по внутрішніх регламентах, наказах та інструкціях. Витік цих даних може призвести до розкриття комерційної таємниці. Для забезпечення комплексного захисту в системі реалізовано багаторівневу архітектуру безпеки, яка включає надійні механізми автентифікації, криптографічне хешування паролів та ізоляцію даних як у стані спокою, так і під час передачі.

Для забезпечення безпечної та безстанної (stateless) взаємодії між клієнтським веб-інтерфейсом (Streamlit) та серверною частиною (FastAPI) впроваджено механізм токенної автентифікації на основі стандарту JSON Web Token (JWT). Вибір даного стандарту зумовлений тим, що він не вимагає зберігання сесій на стороні сервера, що суттєво економить оперативну пам'ять та дозволяє легко масштабувати API у майбутньому.

Структурно JWT складається з трьох логічних частин, розділених символом крапки: заголовка, корисного навантаження та криптографічного підпису. Токен формується шляхом послідовного кодування заголовка та корисного навантаження у безпечний текстовий формат Base64Url, після чого ці частини конкатенуються (об'єднуються) разом із підписом [16].

Заголовок містить інформацію про тип токена та використаний алгоритм хешування (у нашому випадку це алгоритм HMAC з використанням SHA-256). Корисне навантаження включає твердження про користувача: його унікальний ідентифікатор, рівень доступу (наприклад, звичайний працівник або адміністратор) та мітку часу завершення дії токена. Наявність часу життя є критичним елементом безпеки, оскільки навіть у випадку перехоплення токена зломисником, термін його дії буде суворо обмежений (наприклад, 8 годин робочої зміни).

Найважливішою частиною з точки зору безпеки є формування цифрового підпису. Алгоритм HMAC-SHA256 базується на застосуванні криптографічної хеш-функції до об'єднаних даних заголовка та навантаження з обов'язковим використанням секретного ключа. Цей ключ зберігається виключно у зашифрованих змінних середовища на сервері. Під час кожного запиту сервер самостійно перераховує підпис на основі вхідних даних і порівнює його з тим, що надіслав клієнт. Якщо вони ідентичні – цілісність підтверджено і дані не були змінені третіми особами. Підробити такий токен без знання серверного ключа обчислювально неможливо.

Зберігання паролів адміністраторів системи у відкритому текстовому вигляді є грубим порушенням стандартів інформаційної безпеки. У разі компрометації бази даних зловмисник отримає прямий доступ до всіх облікових записів. Для вирішення цієї проблеми у розробленому програмному продукті застосовується необоротне криптографічне хешування із використанням спеціалізованого алгоритму bcrypt.

На відміну від стандартних хеш-функцій (таких як MD5 або SHA-256), які працюють надзвичайно швидко і є вразливими до атак повним перебором та атак за словником за допомогою райдужних таблиць, алгоритм bcrypt розроблений спеціально для захисту паролів. Його головна перевага – вбудований механізм адаптивного розтягування ключа та обов'язкове використання криптографічної солі.

Сіль являє собою випадкову послідовність байтів, яка генерується індивідуально для кожного користувача під час реєстрації і концептуально «змішується» з паролем перед його хешуванням. Це гарантує, що два користувачі з однаковими паролями матимуть абсолютно різні хеші у базі даних, що повністю нівелює ефективність заздалегідь підготовлених хакерських таблиць.

Важливою складовою алгоритму bcrypt є фактор вартості (cost factor), який визначає кількість ітерацій внутрішнього перемішування даних. Алгоритм спеціально спроектований так, щоб бути вимогливим до оперативної пам'яті

сервера (memory-hard). Це робить його масовий перебір на спеціалізованих графічних процесорах (GPU), які зазвичай використовують зловмисники, економічно і технічно не вигідним.

Процес перевірки пароля під час входу в систему не потребує його дешифрування. Сервер отримує введений користувачем відкритий пароль, вилучає параметри солі та фактор вартості зі збереженого в базі даних хешу, і проводить ту саму серію криптографічних перетворень. Якщо обчислений результат повністю збігається зі збереженим еталоном, система підтверджує автентичність адміністратора. Якщо обчислювальні потужності у майбутньому зростуть, адміністратор системи може просто збільшити фактор вартості, що збереже надійність захисту на багато років уперед.

Комплексна безпека корпоративної інформаційної системи вимагає забезпечення захисту даних на всіх етапах їх життєвого циклу: під час транзиту (передачі по локальній мережі) та в стані спокою (на фізичних носіях сервера).

Захист інформації під час передачі. Взаємодія між клієнтом та сервером здійснюється через зашифрований протокол HTTPS, основою якого є сучасний стандарт Transport Layer Security (TLS 1.3).

На початковому етапі встановлення з'єднання (рукоштовуванні) клієнт і сервер використовують алгоритми на еліптичних кривих. Завдяки складним математичним властивостям цих кривих, обидві сторони можуть незалежно згенерувати однаковий спільний секретний ключ, жодного разу не передаючи його у відкритому вигляді по мережі.

Цей спільний ключ надалі використовується для симетричного шифрування всього трафіку алгоритмом AES-256 у режимі GCM (Galois/Counter Mode). Цей режим не лише шифрує текст пошукових запитів та відповідей, але й створює спеціальний автентифікаційний тег для кожного пакету даних. Це гарантує цілісність передачі і унеможливорює атаки типу «людина посередині» (Man-in-the-Middle), коли зловмисник намагається непомітно перехопити або змінити дані під час їхнього руху локальною мережею підприємства.

Цей протокол використовує комбінований підхід до шифрування, алгоритм реалізації якого наведено на рисунку 4.1.

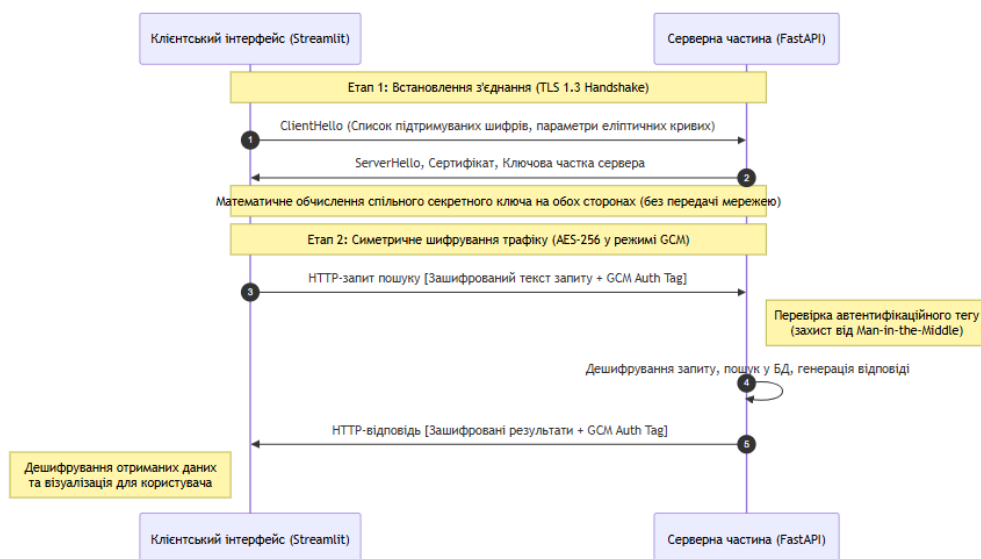


Рисунок 4.1 – Механізм встановлення захищеного з'єднання TLS 1.3 та шифрування AES-256 GCM

Захист даних у стані спокою та ізоляція архітектури. Фундаментальною перевагою розробленої системи порівняно з підписками на хмарні ШІ-сервіси є повна ізоляція розгортання. Усі компоненти програмного продукту: реляційна база метаданих SQLite, векторна база ChromaDB та нейромережева модель розгорнуті виключно на локальних серверах у внутрішній інтранет-мережі підприємства.

Архітектура векторної бази ChromaDB передбачає збереження фрагментів документів у вигляді багатовимірних векторів. Оскільки вектори є лише просторовим представленням семантики (набором чисел з рухомою комою), без доступу до оригінальної моделі Sentence-Transformers та специфічного словника токенів відновити початковий текст конфіденційного наказу з цього масиву технічно неможливо. Це створює надійний рівень захисту від реверс-інжинірингу (декомпіляції) бази знань.

Така архітектурна парадигма повністю усуває ризик витоку корпоративної документації за межі контрольованого безпекового периметра потенційних споживачів.

Для забезпечення високої швидкодії системи семантичного пошуку ($\text{Latency} < 100 \text{ мс}$), усі векторні представлення корпоративних документів повинні зберігатися в оперативній пам'яті (RAM) сервера. Розрахуємо мінімально необхідний обсяг пам'яті для зберігання щільних векторів бази даних ChromaDB.

Теоретичний обсяг оперативної пам'яті для зберігання векторних індексів обчислюється за формулою:

$$V = N \cdot d \cdot S \quad (4.1)$$

де V – мінімальний обсяг оперативної пам'яті (у байтах);

N – загальна кількість текстових фрагментів (чанків) у базі даних;

d - розмірність семантичного вектора (для використаної моделі all-MiniLM-L6-v2 цей показник є сталим і становить 384 виміри);

S - розмір одного числа з плаваючою комою у пам'яті (для типу float32 становить 4 байти).

Проведений спеціальний розрахунок мав на меті теоретично обґрунтувати низькі апаратні вимоги розробленої системи, зокрема щодо обсягу оперативної пам'яті (RAM). Обчислення базувалося на визначенні загальної ваги просторових векторів для гіпотетичної корпоративної бази з 5000 документів (близько 50 тисяч текстових фрагментів), кожен з яких нейромережа перетворює на числовий масив із 384 значень. Результати показали, що для утримання такого масиву даних у «гарячому» кеші разом із системними індексами бази ChromaDB достатньо всього близько 110 МБ оперативної пам'яті. У поєднанні із завантаженою легковаговою моделлю та серверним фреймворком загальне споживання системи не перевищує 500 МБ. Це повністю узгоджується з результатами практичного тестування та математично доводить,

що програмний продукт є надзвичайно ресурсоефективним і здатний працювати на стандартних офісних комп'ютерах без закупівлі додаткових серверних потужностей.

Отже, для індексації та утримання в «гарячому» кеші 50 000 корпоративних текстових фрагментів векторна база даних потребує лише близько 110 МБ оперативної пам'яті. Разом із завантаженою в пам'ять моделлю SBERT (близько 100 МБ) та серверним фреймворком FastAPI, загальне споживання оперативної пам'яті системою не перевищить 400-500 МБ. Це теоретично підтверджує раніше отримані результати профілювання (Рисунок 3.5) та доводить, що система є надзвичайно ресурсоефективною і не потребує закупівлі додаткових планок RAM для серверів підприємства.

ВИСНОВКИ

У кваліфікаційній роботі бакалавра вирішено актуальне науково-практичне завдання, яке полягає у проєктуванні та розробці локальної інтелектуальної системи семантичного пошуку для оптимізації управління масивами корпоративної документації. За результатами виконання роботи можна зробити наступні висновки.

Проведено аналіз предметної області та існуючих систем корпоративного пошуку. Встановлено, що класичні лексичні алгоритми (повнотекстовий пошук) є недостатньо ефективними для роботи зі спеціалізованою документацією через нездатність враховувати контекст та професійну синонімію. Це повністю обґрунтувало необхідність переходу до семантичних методів на базі штучного інтелекту.

Досліджено сучасні технології обробки природної мови (NLP) та обґрунтовано вибір технологічного стеку. Для забезпечення високої точності та швидкодії обрано архітектуру на базі легких нейромережевих моделей сімейства Sentence-Transformers у поєднанні зі спеціалізованою векторною базою даних ChromaDB та реляційною СУБД SQLite для збереження метаданих.

Спроектовано та реалізовано багаторівневу програмну архітектуру. Серверна частина побудована на базі асинхронного фреймворку FastAPI, що забезпечує швидку взаємодію компонентів, а клієнтська частина – за допомогою фреймворку Streamlit, який надає інтуїтивно зрозумілий веб-інтерфейс. Створено модулі для автоматизованого парсингу документів (PDF, DOCX), їх сегментації (chunking) та генерації векторних представлень (embeddings).

Особливу увагу приділено питанням інформаційної безпеки та захисту комерційної таємниці. Впроваджено механізми токенної автентифікації (JWT) для розмежування прав доступу та криптографічне хешування паролів за алгоритмом bcrypt. Завдяки повністю локальному розгортанню (без

використання сторонніх хмарних API) система гарантує абсолютну ізоляцію корпоративних даних від зовнішніх загроз.

Проведено комплексне тестування розробленого програмного продукту на реальних масивах неструктурованих текстів. Результати підтвердили стабільність роботи алгоритмів: система успішно долає термінологічний бар'єр і повертає релевантні фрагменти документів навіть за відсутності точного лексичного збігу із запитом користувача.

Підсумовуючи, можна стверджувати, що мета кваліфікаційної роботи досягнута. Розроблена система є повноцінним, автономним програмним продуктом, готовим до впровадження в IT-інфраструктуру будь-якого сучасного підприємства. Її використання дозволяє мінімізувати час на пошук необхідних інструкцій та регламентів, тим самим підвищуючи загальну ефективність роботи персоналу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python 3.10.14 Documentation. URL: <https://docs.python.org/3/> (дата звернення: 10.03.2026).
2. Sentence-Transformers: State-of-the-Art Text and Image Embeddings. URL: <https://www.sbert.net/> (дата звернення: 12.00.2026).
3. Chroma: the AI-native open-source embedding database. URL: <https://docs.trychroma.com/> (дата звернення: 12.04.2026).
4. FastAPI – High performance, easy to learn, fast to code, ready for production. URL: <https://fastapi.tiangolo.com/> (дата звернення: 14.04.2026).
5. Streamlit URL: <https://streamlit.io/> (дата звернення: 14.04.2026).
6. PyMuPDF 1.24.0 documentation. URL: <https://pymupdf.readthedocs.io/en/latest/> (дата звернення: 15.04.2026).
7. python-docx – python-docx 1.1.0 documentation. URL: <https://python-docx.readthedocs.io/en/latest/> (дата звернення: 15.04.2026).
8. SQLite Home Page. URL: <https://www.sqlite.org/index.html> (дата звернення: 16.04.2026).
9. What is a Vector Database? | Pinecone. URL: <https://www.pinecone.io/learn/vector-database/> (дата звернення: 18.04.2026).
10. Understanding Cosine Similarity and its Application. URL: <https://www.sciencedirect.com/topics/computer-science/cosine-similarity> (дата звернення: 18.04.2026).
11. Advances in Neural Information Processing Systems. – 2023. URL: <https://arxiv.org/abs/1706.03762> (дата звернення: 20.04.2026).
12. Git. URL: <https://git-scm.com/> (дата звернення: 21.04.2026).
13. Основи обробки природної мови (NLP) з використанням Python. URL: <https://realpython.com/nltk-nlp-python/> (дата звернення: 22.04.2026).

14. OWASP Top 10:2021 – Cryptographic Failures // OWASP Foundation.
URL: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/ (дата звернення:
05.05.2026).

15. Security: OAuth2 with Password (and hashing), Bearer with JWT tokens
FastAPI Official Documentation. URL:
<https://fastapi.tiangolo.com/tutorial/security/oauth2-jwt/> (дата звернення:
07.05.2026).

16. Introduction to JSON Web Tokens / Auth0. URL:
<https://jwt.io/introduction> (дата звернення: 11.05.2026).

ДОДАТКИ

Додаток А

Фрагмент коду реалізації маршруту семантичного пошуку (FastAPI)

```
from fastapi import APIRouter, HTTPException
import time
router = APIRouter()
@router.post("/search", response_model=SearchResponse)
async def semantic_search(request: SearchRequest):
    start_time = time.time()
    try:
        # 1. Векторизація запиту користувача
        query_vector = get_embedding(request.query)
        # 2. Пошук у векторній БД
        vector_results = collection.query(
            query_embeddings=[query_vector],
            n_results=request.top_k
        )
        if not vector_results['ids'][0]:
            return {"results": [], "processing_time": 0.0}
        final_results = []
        # 3. Об'єднання з метаданими SQLite
        for i, doc_id in enumerate(vector_results['ids'][0]):
            db_cursor.execute("SELECT file_name FROM documents WHERE
id=?", (doc_id,))
            meta = db_cursor.fetchone()
            final_results.append(SearchResult(
                document_id=doc_id,
                file_name=meta[0] if meta else "Невідомий файл",
                text_chunk=vector_results['documents'][0][i],
                similarity_score=1.0 - vector_results['distances'][0][i]
            ))
```

```
        process_time = round(time.time() - start_time, 3)
        return SearchResponse(results=final_results,
                                processing_time=process_time)
    except Exception as e:
        logger.error(f"Search error: {str(e)}")
        raise HTTPException(status_code=500, detail="Internal Server Error")
```

Додаток Б

Програмна реалізація підсистеми логування подій

```
import logging
import sys

# Налаштування формату виводу логів
log_format = "%(asctime)s | [%(levelname)s] | %(name)s : %(message)s"

# Конфігурація базового логера
logging.basicConfig(
    level=logging.INFO,
    format=log_format,
    handlers=[
        # Збереження логів у текстовий файл для аудиту
        logging.FileHandler("system_logs.log", encoding="utf-8"),
        # Паралельне виведення у системну консоль
        logging.StreamHandler(sys.stdout)
    ]
)

# Ініціалізація логера для поточного модуля
logger = logging.getLogger("SemanticSearchApp")

logger.info("Система успішно ініціалізована та готова до прийому
запитів.")
```

Додаток В

Фрагмент коду клієнтського веб-інтерфейсу (Streamlit)

```
import streamlit as st
import requests

st.title(" Інтелектуальний корпоративний пошук")
# Поле вводу запиту
query = st.text_input("Введіть запит (напр., 'як оформити відрядження')")

if st.button("Знайти"):
    if query:
        # Індикатор завантаження під час виконання запиту
        with st.spinner('Нейромережа аналізує документи...'):
            try:
                response = requests.post(
                    "http://localhost:8000/search",
                    json={"query": query, "top_k": 3}
                )
                data = response.json()
                st.success(f"Знайдено за {data['processing_time']} сек.")
                # Виведення результатів
                for res in data['results']:
                    with st.expander(f" {res['file_name']} (Точність:
{res['similarity_score']:.2f})"):
                        st.write(res['text_chunk'])

            except Exception as e:
                st.error("Помилка зв'язку з сервером бази знань. Перевірте
підключення.")
```