Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

Бакалаврська дипломна робота на тему:

«РОЗРОБКА ДОДАТКУ АВТОМАТИЗОВАНОГО ПОШУКУ КОНТЕНТУ ВЕБСАЙТУ ЗА ДОПОМОГОЮ ВЕКТОРНИХ БАЗ ДАНИХ»

Виконав: студент IV курсу, групи 1АКІТ-20б
спеціальності 151 – Автоматизація та
комп'ютерно-інтегровані технології
(шифр і назва спеціальності)
Герасімов Є.Є.
(прізвище та ініціали)
Керівник: к.т.н., доцент кафедри АІІТ
Богач І. В.
(прізвище та ініціали)
«11» 06 2024 р.
Рецензент:д.т.н., професор кафедри КН
Іванчук Я.В.
(прізвище та ініціали)
«12» 06 2024 р.

Допущено до захисту
Зав. кафедри АІІТ

О.В.Бісікало
«13» 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Галузь знань 15 – Автоматизація та приладобудування
Спеціальність 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітньо-професійна програма Інтелектуальні комп'ютерні системи управління

ЗАТВЕРДЖУЮ
зав. кафедри АІІТ
_____ д.т.н., проф. Бісікало О. В.
«15» 03 2024 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Герасімову Євгену Євгеновичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка додатку автоматизованого пошуку контенту вебсайту за допомогою векторних баз даних»
керівник роботи Богач Ілона Віталіївна, к.т.н, доцент, доцент кафедри АІІТ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "11" березня 2024 року № 80
2. Термін подання студентом роботи 10.06.2024 р.
3. Вихідні дані до роботи:
: мова програмування – Python на будь-якій платформі, векторна база даних – Pinecone.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної галузі. Інформаційне забезпечення. Математичне забезпечення. Розробка програмного забезпечення.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Діаграма залежності модулів файлу main.py. Діаграма залежності модулів файлу setup.py. Діаграма послідовності. Блок-схема роботи веб-додатку

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Богач І. В., доц. каф. АІІТ		

7. Дата видачі завдання 12.03.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів бакалаврської дипломної роботи	Термін виконання		Примітка
		початок	закінчення	
1	Вибір, узгодження та затвердження теми БДР	01.03.2024	21.03.2024	
2	Аналіз літературних джерел	21.03.2024	15.03.2024	
3	Розробка основних розділів	16.03.2024	29.03.2024	
4	Аналіз методів та програмних рішень вирішення поставленої задачі	30.03.2024	10.04.2024	
5	Розробка програмного забезпечення	11.04.2024	17.06.2024	
6	Захист БДР	18.06.2024	18.06.2024	

Студент _____ Герасімов Є. Є.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Богач І. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 70 сторінок формату А4, в тому числі і додатки на яких є 19 рисунок, 2 формули, 3 таблиці, список використаних джерел містить 23 найменування.

У бакалаврській дипломній роботі розглянуто існуючі принципи пошуку контенту на вебсайті, принцип роботи векторного пошуку, застосування векторного пошуку для знаходження схожого контенту по запиту користувача. Система є зручною через легкість налаштування пошуку контенту, вона не вимагає попереднього встановлення параметрів та фільтрів пошуку. Векторна база даних Pinecone надає можливість автоматичного масштабування.

Ключові слова: векторний пошук, вбудування слів, Pinecone, Python, метрики відстані, векторні бази даних.

ABSTRACT

The bachelor's thesis consists of 70 pages of A4 format, which include 19 figures, 2 formulas, 3 tables and a list of references containing 23 titles.

The bachelor thesis examines the existing principles of content search on the website, the principle of operation of vector search, the application of vector search for finding similar content at the user's request. The system is convenient due to the ease of setting up content search, it does not require prior setting of parameters and search filters. Pinecone's vector database provides auto-scaling.

Keywords: vector search, word embeddings, Pinecone, Python, distance metrics, vector databases.

ЗМІСТ

ВСТУП

Актуальність роботи. В умовах стрімкого зростання обсягу веб-контенту користувачі стикаються з все більшими труднощами у пошуку точної та релевантної інформації. Кількість даних зростає експоненціально, і традиційні методи пошуку не справляються з цим завданням через свою обмеженість у швидкості, точності та масштабованості. В результаті користувачі витрачають багато часу на пошук потрібної інформації, що знижує ефективність їх роботи і задоволення від використання інтернет-ресурсів.

Розробка додатку, що забезпечить користувачам можливість швидкого і точного виявлення релевантної інформації з широкого спектру контенту вебсайту, є надзвичайно актуальною. Використання сучасних методів машинного навчання та обробки природної мови, особливо із застосуванням векторних баз даних [1, 12], дозволить значно підвищити якість пошуку. Такі технології зможуть автоматично відфільтровувати нерелевантний контент, надаючи користувачам виключно точну та актуальну інформацію, що значно зекономить їх час і підвищить ефективність роботи.

Забезпечення доступу до точної інформації в найкоротші терміни є важливим для різних сфер діяльності – від бізнесу до науки і освіти. Новий додаток, який не потребує попереднього налаштування, працює за запитом користувача та легко масштабується, стане незамінним інструментом для різних категорій користувачів. Це дозволить зменшити навантаження на користувачів, покращити їх досвід використання веб-ресурсів та підвищити загальну продуктивність.

Тому актуальність розробки додатку для автоматизованого та оптимізованого пошуку веб-контенту, що використовує передові технології, не викликає сумнівів. Такий додаток стане важливим внеском у розвиток інформаційних технологій і допоможе користувачам ефективніше взаємодіяти з величезними обсягами даних, які щоденно створюються в інтернеті.

Метою роботи є покращення сервісів автоматизації та оптимізації пошуку веб-контенту за рахунок векторних баз даних, що на відміну від існуючих методів

пошуку контенту не потребує попереднього налаштування, працює по запиту користувача та є легкомасштабованою.

Для розв’язання поставленої мети потрібно виконати **наступні задачі**:

1. Провести аналіз предметної області, а саме аналіз сучасних методів індексування та пошуку веб-контенту, функціональні та технічні вимоги.

2. Розробити архітектуру проєкту та обрати інструменти та програмні засоби для розробки.

3. Розробити серверну частину веб-додатку.

4. Інтегрувати векторну базу даних з додатком.

5. Протестувати готове рішення.

Об’єктом дослідження є процес пошуку та аналізу веб-контенту сторінки.

Предметом дослідження є методи та алгоритми векторного пошуку [2] і їх застосування для знаходження веб-контенту.

Методи дослідження: аналіз наукової літератури, моделювання систем, експериментальне програмування та аналіз даних для оцінки різних підходів.

Науково-практичний результат роботи полягає в розробці прототипу веб-додатку, який застосовує векторні бази даних для оптимізації пошуку веб-контенту, що допоможе покращити ефективність пошуку інформації і має потенціал для впровадження у комерційних та освітніх установах.

Результатом реалізації проекту стане не лише програмний продукт, але й методика його використання, що може бути застосована у різних сферах, де важливий швидкий доступ до точної інформації.

Апробація результатів роботи. Результати роботи було представлено на Всеукраїнській науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: Дослідження, проблеми, перспективи (МН-2024)» [3].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність та еволюція векторного пошуку

Еволюція та актуальність векторного пошуку в пошуку інформації набули особливого значення з експоненційним зростанням даних у різних цифрових форматах. Спочатку традиційних методологій пошуку, таких як відповідність ключових слів, було достатньо для керування та отримання інформації. Однак із різким розширенням баз даних та Інтернету ці методи почали відставати в ефективності та актуальності, що призвело до дослідження та впровадження більш складних технологій.

Векторний пошук, який використовує векторні просторові моделі для розуміння та виконання пошуку шляхом відображення текстових чи інших даних у вектори вбудувань [4], є проривом у роботі з величезними неструктурованими наборами даних. Цей підхід принципово відрізняється від традиційних методів пошуку, які покладаються на точні збіги ключових слів. Замість цього векторний пошук переводить запити та документи у багатовимірний простір, де близькість векторів визначає їх семантичну подібність [5]. Це не тільки покращує точність результатів, але й дозволяє системі розуміти контекст, нюанси та навіть синоніми без явної специфікації.

Актуальність векторного пошуку сьогодні підкреслюється його застосуванням у різних галузях, таких як електронна комерція, де він покращує рекомендації щодо продуктів і пошук, а також у сфері обслуговування клієнтів, де він покращує релевантність автоматизованих відповідей на запити користувачів. В академічних і наукових спільнотах векторний пошук дозволяє точніше шукати дослідницьку статтю завдяки розумінню глибшого змісту документів, а не лише ключових слів поверхневого рівня.

Крім того, інтеграція машинного навчання [6] та особливо глибокого навчання [7] у векторний пошук призвела до ще більш потужних інструментів пошуку. Такі методи, як вбудовування, згенеровані нейронними мережами,

забезпечують ще більш глибоке розуміння вмісту запитів і документів. Ці досягнення не тільки зробили пошук більш контекстно обізнаним, але й значно швидшим, оскільки вони можуть ефективніше впоратися з неоднозначністю та різноманітністю людської мови.

Оскільки обсяг даних продовжує зростати, а підприємства та установи шукають більш ефективні способи навігації цією інформацією, важливість технологій векторного пошуку стає все більш очевидною. Він є ключовим компонентом сучасних пошукових систем і систем рекомендацій, просуваючи інновації в пошуку та аналізі даних, які є важливими для епохи, керованої інформацією. Таким чином, безперервна еволюція технологій векторного пошуку є життєво важливою для того, щоб йти в ногу з цифровим ландшафтом, що постійно розширюється, що робить його критичною сферою вивчення та застосування в сучасному світі, що керується технологіями.

1.2 Майбутній розвиток векторного пошуку

Дивлячись у майбутнє, розвиток векторного пошуку має намір стати ще більш невід’ємною частиною системи пошуку інформації, що спонукає постійний прогрес штучного інтелекту та машинного навчання. Траєкторія векторного пошуку вказує на зрушення до все більш нюансованих і контекстно-залежних систем, які можуть легко інтерпретувати та обробляти людську мову в усій її складності.

Одним із найбільш перспективних напрямків розвитку векторного пошуку є інтеграція більш складних моделей нейронних мереж, таких як трансформатори, які вже зробили революцію в обробці природної мови [14]. Ці моделі сприяють глибшому розумінню контексту, настрою та семантичних зв’язків у текстах, дозволяючи інструментам векторного пошуку надавати не лише більш відповідні, але й більш глибокі результати пошуку.

Крім того, оскільки векторний пошук продовжує розвиватися, він, імовірно, стане більш персоналізованим. Майбутні системи зможуть динамічно адаптуватися

до індивідуальних уподобань користувачів та історії пошуку, пропонуючи індивідуальний пошук, який постійно вдосконалюється. Ця персоналізація виходить за межі простої взаємодії з користувачем, потенційно включаючи потоки даних у реальному часі для надання контекстуально релевантної інформації, яка узгоджується з безпосереднім середовищем і діяльністю користувачів.

Іншою важливою подією стане підвищення ефективності алгоритмів векторного пошуку. Оскільки набори даних зростають експоненціально, обчислювальна потреба в їх обробці за допомогою векторних моделей стає критичною проблемою. Дослідники постійно працюють над більш ефективними способами обчислення та отримання векторних вбудовань, можливо, за допомогою інновацій у квантових обчисленнях або більш удосконалених методів векторного квантування, які можуть значно скоротити час і обчислювальні ресурси, необхідні для пошуку.

Крім того, очікується, що крос-модальний пошук, коли система може отримувати інформацію з різних типів даних (наприклад, текст, зображення, аудіо), використовуючи уніфікований векторний підхід, стане більш поширеним. Це сприятиме більш інтегрованому та бездоганному цифровому досвіду, дозволяючи користувачам здійснювати пошук за різними типами медіа без потреби в окремих системах пошуку.

Майбутній розвиток векторного пошуку також може значно вплинути на конфіденційність і безпеку даних. Оскільки ці системи стають більш вправними у розумінні та прогнозуванні поведінки користувачів, забезпечення етичного використання даних і захисту конфіденційності користувачів стане першорядним. Удосконалені методи шифрування та об’єднані моделі навчання можуть зіграти вирішальну роль у захисті інформації користувача, одночасно використовуючи переваги персоналізованого векторного пошуку.

Загалом, майбутнє векторного пошуку полягає не лише у покращенні можливостей і швидкості пошукових технологій, а й у створенні більш інтелектуальних, оперативних і відповідальних систем, які можуть підтримувати та покращувати прийняття рішень людиною у все більш складному цифровому світі.

1.3 Типові проблеми при пошуку за векторами

Виконання векторного пошуку представляє ряд типових проблем, які можуть вплинути на його ефективність, точність і практичне використання. Ось деякі з поширених проблем, які виникають, а саме розмірність векторів, масштабованість, якість векторів, компроміс між швидкістю та точністю, неоднозначність запиту, розрідженість даних, оновлення моделі, упередженість і справедливість.

Розглянемо кожну з них більш детальніше.

Розмірність векторів: векторний пошук передбачає роботу з даними великої розмірності, що може призвести до неефективності обчислень і збільшення вимог до ресурсів. Це явище, яке часто називають «прокляттям розмірності», ускладнює ефективне керування великими наборами даних і пошук у них, оскільки відстань між точками стає менш значущою у просторі з великою розмірністю.

Масштабованість: із зростанням наборів даних підтримка продуктивності під час масштабування системи векторного пошуку стає складною. Індексація та пошук у мільйонах або навіть мільярдах векторів потребують значних обчислювальних ресурсів і інтелектуального структурування даних, щоб уникнути експоненціального зростання часу обробки.

Якість векторів: ефективність векторного пошуку значною мірою залежить від якості вбудовування вектора. Погано навчені моделі або вбудовування, які не вловлюють нюанси даних, можуть призвести до нерелевантних результатів пошуку. Це може бути особливо проблематично в областях, які вимагають глибокого розуміння контексту, наприклад, юридичні документи або технічні статті.

Компроміс між швидкістю та точністю: часто існує компроміс між швидкістю пошуку та точністю результатів пошуку. Методи, які забезпечують швидший час пошуку, такі як алгоритми Approximate Nearest Neighbor (ANN) [8], можуть пожертвувати точністю, тоді як точні методи пошуку, можуть бути надто повільними для практичного використання, особливо в програмах реального часу.

Неоднозначність запиту: моделі векторного пошуку зазвичай побудовані на алгоритмах машинного навчання, які інтерпретують семантику запиту на основі контексту в навчальних даних. Однак запити можуть бути неоднозначними або мати кілька значень, що призводить до того, що пошук дає менш релевантні результати, якщо контекст не добре зрозумілий моделі.

Розрідженість даних: у деяких випадках, особливо з текстовими даними, векторні представлення можуть страждати від проблем розрідженості, коли багато елементів у векторі є нулями. Це може погіршити продуктивність пошуку, оскільки вектори не фіксують інформацію, необхідну для точного порівняння.

Оновлення моделі: у динамічному середовищі, де постійно надходять нові дані, оновлювати модель векторного пошуку може бути проблемою. Моделі перепідготовки для включення нової інформації без втрати контексту зі старих даних вимагають складних стратегій для балансування між стабільністю та адаптивністю.

Упередженість і справедливість [9]: системи векторного пошуку, як і будь-який інший інструмент, керований штучним інтелектом, можуть успадковувати або навіть посилювати упередження, наявні в навчальних даних. Це може призвести до упереджених результатів пошуку, які можуть надавати перевагу певним групам або типам інформації, що потенційно може призвести до проблем із справедливістю.

1.4 Розгляд існуючих методів пошуку інформації на вебсайті

Розгляд існуючих методів пошуку інформації та їхніх особливостей - це важлива частина дослідження, яка дозволяє зрозуміти різноманітність та функціональні різних підходів. Далі наведено кілька прикладів найбільше популярних пошукових методів, а саме пошук за ключовими словами, фасетний пошук, пошук по повному тексту та їхні особливості.

1.4.1 Пошук за ключовими словами

Пошук за ключовими словами є основним і широко використовуваним методом пошуку інформації на веб-сайті або в будь-якому сховищі цифрового вмісту.

Принцип роботи:

1. Вхідні дані користувача: користувач вводить одне або кілька ключових слів або фраз у рядок пошуку на веб-сайті. Ці ключові слова представляють запит користувача або інформацію, яку він шукає.

2. Обробка запитів: пошукова система веб-сайту обробляє запити користувача, як правило, розбираючи та аналізуючи ключові слова, щоб зрозуміти наміри користувача. Цей крок може передбачати видалення загальних слів (стоп-слов), обробку спеціальних символів і застосування основних методів нормалізації запитів.

3. Пошук індексу: потім пошукова система шукає проіндексований вміст у базі даних веб-сайту або в сховищі вмісту. Покажчик містить інформацію про те, де у вмісті з’являється кожне ключове слово (наприклад, документи, статті, сторінки).

4. Зіставлення та ранжування: система визначає елементи вмісту (наприклад, веб-сторінки, документи), які містять ключові слова із запиту користувача. Залежно від алгоритму пошуку, який використовується, система може ранжувати ці елементи вмісту на основі релевантності, популярності, нещодавності або інших критеріїв.

5. Результати пошуку: нарешті, пошукова система надає користувачеві список результатів пошуку, як правило, у формі посилань, які можна натиснути, або фрагментів, які переглядають вміст. Користувачі можуть натиснути на результат, щоб отримати доступ до повного вмісту.

Переваги пошуку за ключовими словами:

− Знайомість і простота використання: користувачі знайомі з введенням ключових слів у рядки пошуку, що робить його інтуїтивно зрозумілим і зручним способом пошуку інформації.

− Проста реалізація: реалізація основної функції пошуку на основі ключових слів є відносно простою та не потребує складних алгоритмів чи технологій.

− Універсальність: пошук за ключовими словами може обробляти широкий спектр запитів, від простих пошуків за ключовими словами до більш складних запитів із кількома ключовими словами та операторами (наприклад, І, АБО, НІ).

− Швидкий результат пошуку: у багатьох випадках пошукові системи на основі ключових слів можуть швидко отримувати результати пошуку, особливо якщо проіндексований вміст добре організований, а інфраструктура пошуку оптимізована.

Недоліки пошуку за ключовими словами:

− Залежність від точної відповідності: пошук за ключовими словами базується на точних або часткових відповідностях між введеними користувачем ключовими словами та індексованим вмістом. Це може призвести до втрати результатів, якщо користувачі неправильно пишуть слова, використовують синоніми або фразові запити, які відрізняються від того, як вміст індексується.

− Обмежене розуміння контексту: системи пошуку на основі ключових слів зазвичай не мають внутрішнього розуміння контексту, семантики чи зв’язків між словами. У результаті вони можуть давати нерелевантні або менш точні результати, особливо для неоднозначних запитів.

− Перевантаження загальних ключових слів: загальні ключові слова або терміни можуть бути перевантажені багатьма значеннями, що призводить до неоднозначних результатів пошуку. Наприклад, пошук "Java" може стосуватися мови програмування, кави або індонезійського острова.

− Нездатність обробляти складні запити: хоча пошук за ключовими словами може ефективно обробляти основні запити, він може мати проблеми зі складними

запитами, які вимагають розуміння намірів користувача, контексту або нюансів природної мови.

1.4.2 Фасетний пошук

Фасетний пошук — це інтерактивний підхід до пошуку, який дозволяє користувачам уточнювати результати пошуку за допомогою вибору попередньо визначених фільтрів або фасетів. Ці фасети, такі як категорії, дати, автори або теги, дозволяють користувачам структуровано переміщатися та досліджувати вміст, покращуючи пошук, надаючи детальний контроль і персоналізовані результати.

Принцип роботи:

1. Початковий запит: подібно до пошуку за ключовими словами, користувач починає з введення запиту в рядок пошуку на веб-сайті. Цей запит може бути ключовим словом, фразою або комбінацією термінів.

2. Представлення фасетів: після того, як користувач надсилає запит, система фасетного пошуку представляє різні фасети або фільтри на основі атрибутів вмісту. Ці атрибути можуть включати категорії, теги, авторів, дати, типи файлів тощо.

3. Вибір фільтра: потім користувач може вибрати один або кілька аспектів, щоб звузити результати пошуку. Наприклад, під час пошуку статей користувач може вибрати аспект категорії, як-от «Технологія», і аспект дати, як-от «Останній місяць», щоб уточнити результати.

4. Динамічне фільтрування: коли користувач вибирає або скасовує вибір фасетів, результати пошуку динамічно оновлюються відповідно до уточнених критеріїв. Цей інтерактивний процес фільтрації дозволяє користувачам переглядати вміст ітеративно.

5. Ранжування та відображення: пошукова система також може застосовувати алгоритми ранжування, щоб упорядкувати результати пошуку на основі релевантності, популярності чи інших факторів. Уточнені результати зазвичай відображаються користувачеві, часто з опціями для подальшого вивчення чи коригування фасетів.

Переваги фасетного пошуку:

− Детальне уточнення: фасетний пошук дозволяє користувачам застосовувати кілька фільтрів одночасно, забезпечуючи детальний контроль над уточненням результатів пошуку на основі конкретних атрибутів або критеріїв.

− Розширення можливостей користувачів: користувачі можуть досліджувати вміст у структурований та інтуїтивно зрозумілий спосіб, використовуючи аспекти, які відповідають їхнім інформаційним потребам (наприклад, за категорією, датою, автором), що забезпечує більш персоналізований пошук.

− Гнучкий і масштабований: фасетний пошук може добре масштабуватися з великими сховищами вмісту, оскільки він покладається на метадані та атрибути, а не на весь вміст. Додати нові грані або змінити існуючі відносно просто.

− Зменшує когнітивне навантаження: надаючи параметри для фільтрації та категоризації, фасетний пошук допомагає зменшити когнітивне навантаження для користувачів, дозволяючи їм зосередитися на вдосконаленні свого пошуку, а не на формулюванні складних запитів.

Недоліки фасетного пошуку:

− Залежність від метаданих: Ефективний фасетний пошук покладається на добре структуровані метадані та тегування вмісту. Неточні або неповні метадані можуть призвести до неоптимальних результатів пошуку або відсутності відповідного вмісту.

− Знайомість користувача із системою: незважаючи на те, що для багатьох користувачів концепція фасетів і фільтрів є інтуїтивно зрозумілою, деякі можуть вважати її незнайомою, що потребує певного навчання, щоб повністю використовувати можливості системи.

− Управління складністю: зі збільшенням кількості аспектів і опцій управління складністю інтерфейсу багатогранного пошуку та забезпечення зручності використання стає все складнішим.

− Обмежене семантичне розуміння: Системи фасетного пошуку зазвичай зосереджуються на фільтрації за категоріями чи атрибутами й можуть не розуміти

семантичних зв’язків або контексту в запитах користувача поза вказаними фасетами.

1.4.3 Пошук по повному тексту

Повнотекстовий пошук – це комплексний пошуковий підхід, який індексує та шукає весь текстовий вміст на веб-сайті чи в базі даних. Це дозволяє користувачам шукати певні слова, фрази або терміни в документах, статтях або інших текстових даних.

Принцип роботи:

1. Індексація: перед виконанням пошуку система повнотекстового пошуку індексує весь текстовий вміст документів, статей або інших даних на веб-сайті. Цей процес індексування включає розбір, токенізацію та збереження слів або фраз разом із їхніми позиціями у вмісті.

2. Запит користувача: користувач вводить пошуковий запит, який може складатися з одного слова, кількох слів або фрази, у рядок пошуку на веб-сайті.

3. Обробка пошуку: система повнотекстового пошуку обробляє запит користувача, зіставляючи його з індексованим вмістом. Він враховує такі фактори, як близькість слів, синоніми, коріння (наприклад, зіставлення «бігти» з «біг») та інші мовні варіації, щоб знайти відповідні відповідники.

4. Ранжування: коли знайдено збіги, система ранжує результати пошуку на основі релевантності. Цей рейтинг може враховувати такі фактори, як частота появи ключових слів, розташування у вмісті та метадані, пов’язані з елементом вмісту.

5. Результати пошуку: система надає користувачеві список результатів пошуку, як правило, з фрагментами або попередніми переглядами, що підсвічують, де у вмісті з’являються ключові слова. Користувачі можуть натиснути на результат, щоб переглянути весь елемент вмісту.

Переваги пошуку по повному тексту:

− Комплексне охоплення пошуку: повнотекстовий пошук охоплює весь текстовий вміст, дозволяючи користувачам знаходити інформацію навіть у великих документах, статтях або базах даних.

− Гнучкість запитів: користувачі можуть вводити прості ключові слова, складні фрази або навіть використовувати логічні оператори (AND, OR, NOT), щоб уточнити свої запити, що робить повнотекстовий пошук універсальним для широкого діапазону пошукових потреб.

− Контекстна релевантність: алгоритми повнотекстового пошуку враховують контекст, у якому з'являються ключові слова, наприклад близькість слів і їхню співукладеність, що призводить до більш релевантних результатів пошуку.

− Підтримка ранжування: системи повнотекстового пошуку часто містять алгоритми ранжування, які визначають пріоритетність результатів пошуку на основі релевантності, допомагаючи користувачам швидко знаходити найбільш доречну інформацію.

Недоліки пошуку по повному тексту:

− Витратність ресурсів: індексування та виконання повнотекстового пошуку може потребувати ресурсів, особливо для великих обсягів текстового вмісту. Для цього може знадобитися ефективна інфраструктура та методи оптимізації.

− Складність обробки природної мови: Незважаючи на те, що повнотекстовий пошук є універсальним, йому може бути важко зрозуміти складні мовні запити, нюанси чи семантичні зв'язки між словами, що виходять за межі основних лінгвістичних правил.

− Шум і неоднозначність: повнотекстовий пошук може отримати результати з шумом або неоднозначністю, особливо якщо пошукові терміни є загальними словами, неоднозначними термінами або мають кілька значень.

− Проблеми масштабованості: Керування та масштабування систем повнотекстового пошуку може бути складним завданням, оскільки обсяг

індексованого вмісту зростає. Оптимізація продуктивності та релевантності стає вирішальною для бездоганної взаємодії з користувачем.

1.5 Висновки до першого розділу

Підсумовуючи, векторний пошук виступає як трансформаційна технологія в області пошуку інформації, яка відзначається своєю здатністю ефективно орієнтуватися та інтерпретувати величезні масиви даних. Еволюція векторного пошуку характеризується його зростаючою інтеграцією з передовими моделями машинного навчання та його зростаючою важливістю в різних програмах, від електронної комерції до академічних досліджень. Однак реалізація систем векторного пошуку не позбавлена проблем. Такі проблеми, як прокляття розмірності [10], проблема масштабованості та баланс між швидкістю та точністю, є серйозними перешкодами, які потребують постійної уваги.

Крім того, якість векторних вбудовувань і динамічний характер даних вимагають надійних і адаптивних рішень для підтримки релевантності та точності результатів пошуку. Занепокоєння щодо розрідженості даних і неоднозначності запитів ще більше підкреслюють потребу в складному навчанні та точному налаштуванні моделі. Етичні міркування, особливо щодо упередженості та справедливості, є вирішальними, оскільки ці системи стають все більш поширеними в процесах прийняття рішень.

Вирішення цих проблем вимагатиме узгоджених зусиль у дослідженнях і розробках, зосереджених на вдосконаленні алгоритмів, оптимізації обчислювальних стратегій і забезпеченні дотримання етичних стандартів. Майбутнє векторного пошуку багатообіцяюче з потенційним прогресом у крос-модальному пошуку та можливостях обробки в реальному часі. У міру розвитку технології вона, безсумнівно, відіграватиме вирішальну роль у формуванні наступного покоління пошукових систем і систем рекомендацій, розвиваючи наші можливості впоратися з інформаційним перевантаженням цифрової епохи.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Вхідними даними до системи є пошуковий запит користувача системи. Наприклад, "Технології майбутнього", "Ефективні методи боротьби зі стресом", або "Інновації в області освіти". Текст перетворюється в N-вимірний масив чисел за допомогою запиту до моделі вбудувань OpenAI [11] по якому знаходиться контент, який є найбільшим схожим до запиту користувача. Система передбачає можливість вводу пошукових запитів на різних мовах без додаткових перетворень.

2.2 Вихідні дані

Вихідними даними є список релевантних TED-виступів. Система ранжує їх за ступенем релевантності до запиту користувача, використовуючи векторний пошук з косинусною подібністю. Найбільш релевантні виступи розміщуються на вершині списку. Кожен виступ у списку супроводжується короткою інформацією для користувача.

Додатково для логування результатів пошуку записується рівень подібності знайденого контенту по пошуковому запиту. Ця міра може бути представлена у вигляді відсотка або числового значення.

2.3 Структура масивів інформації

Набір даних містить 5631 запис, тобто дані з 5631 сторінок. Детальний опис та структуру набору даних наведено в таблиці 2.1.

Таблиця 2.1 – Структура зібраного датасету

#	Column	Non-Null Count	Dtype
0	_id	5631 non-null	int64
1	duration	5631 non-null	int64
2	event	5630 non-null	object
3	likes	5631 non-null	object
4	page_url	5631 non-null	object
5	published_date	5631 non-null	object
6	recorded_date	5629 non-null	object
7	related_videos	5631 non-null	object
8	speakers	5631 non-null	object
9	subtitle_languages	5631 non-null	object
10	summary	5631 non-null	object
11	title	5631 non-null	object
12	topics	5631 non-null	object
13	transcript	4983 non-null	object
14	views	5631 non-null	int64
15	youtube_video_code	5462 non-null	object

Набір даних, використаний у цьому роботі, який містить інформацію про виступи TED, був отриманий за допомогою спеціального інструменту веб-збирання, розробленого на Python.

Web scraper було запрограмовано за допомогою Python із використанням таких бібліотек, як `requests` для HTTP-запитів, `BeautifulSoup` для аналізу вмісту HTML і `pandas` для маніпулювання даними та зберігання. Основною функцією скрейпера була навігація веб-сайтом TED, вилучення відповідних даних і систематичне їх зберігання.

Процес аналізу веб-сайту розпочався зі сторінок каталогу веб-сайту TED Talks [13]. На цих сторінках каталогу перелічено кілька доповідей, як правило, із розбивкою на сторінки для обробки великих колекцій. На кожній сторінці каталогу скрепер ідентифікував і витягував URL-адреси окремих сторінок доповідей TED. На кожній сторінці каталогу зазвичай містилося 32 виступи. Скрейпер

використовував цикл, для навігації за допомогою пагінації, ефективно збираючи URL-адреси всіх перелічених виступів TED. Цей систематичний підхід гарантував, що жодна розмова не буде пропущена, а набір даних буде вичерпним.

Маючи під рукою URL-адреси окремих сторінок обговорення TED, наступний етап включав отримання даних кожної з цих сторінок. Скрейпер перебирав зібрані URL-адреси, надсилаючи HTTP-запити для доступу до кожної доповіді TED. Після успішного отримання HTML-вмісту сторінка зберігалась локально як HTML файл задля подальшої розробки парсеру даних.

Скрейпер був запрограмований на вилучення різноманітних даних із кожної сторінки обговорення TED. Серед них:

1. _id: унікальний ідентифікатор TED доповіді.
2. duration: тривалість виступу у секундах.
3. event: конференція, на якій було виголошено доповідь.
4. likes: загальна кількість лайків, отриманих виступом TED, що вказує на його популярність і залучення аудиторії.
5. page_url: пряме посилання на сторінку виступу на веб-сайті TED.
6. published_date: дата, коли виступ було опубліковано онлайн.
7. recorded_date: дата, коли виступ було записано.
8. related_videos: рядок у форматі JSON зі списком _id інших виступів, пов’язаних із розглянутим, полегшуючи тематичні зв’язки.
9. speakers: міститься в рядку у форматі JSON, де надається інформація про доповідача(ів), зокрема їхні імена та професії.
10. subtitle_languages: список мов у форматі JSON, якими доступні субтитри для виступу.
11. summary: стислий опис змісту виступу із оглядом основних моментів, що обговорювалися.
12. title: офіційний заголовок доповіді.
13. topics: рядок у форматі JSON із переліком тем або тегів, пов’язаних з виступом, що сприяє тематичній категоризації.

14. transcript: повна стенограма виступу TED, цінна для детального текстового аналізу.

15. views: загальна кількість переглядів на час скрейпінгу.

16. youtube_video_code: унікальний ідентифікатор для відео виступу TED на YouTube.

За допомогою Python бібліотек matplotlib, seaborn, plotly, wordcloud було створено аналітичні графіки, які описують особливості колонок з текстовими даними.

Три гістограми на рисунку 2.1 представляють розподіл кількості слів для колонок title (заголовок), summary (опис) і transcript (транскрипт) виступів TED. Для описів кількість слів розподілена приблизно нормально, зосереджена навколо 60-70 слів, причому більшість становить від 30 до 100 слів. Є дуже мало описів, які містять менше 20 або більше 100 слів. Загалом описи є відносно лаконічними з чіткою центральною тенденцією, що свідчить про те, що вони написані коротко та по суті.

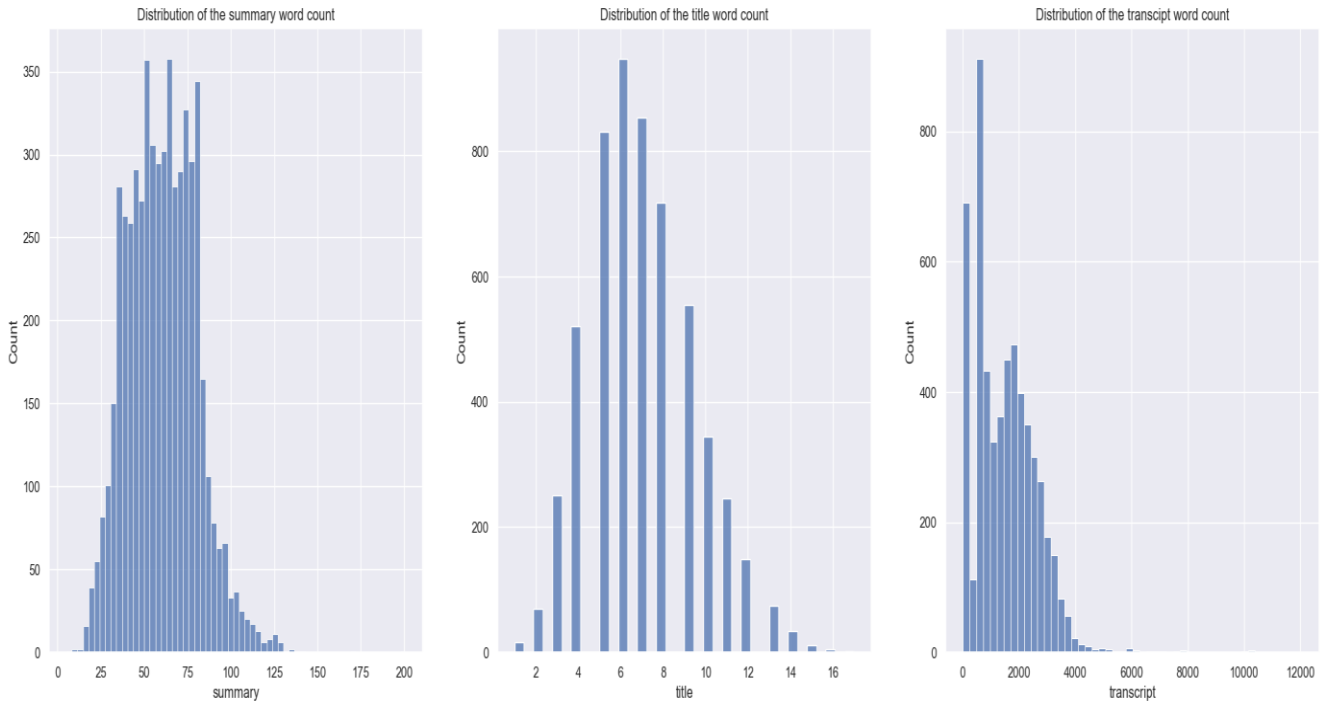

Рисунок 2.1 – Гістограми розподілу кількості слів в колонках summary, title, transcript

Кількість слів для заголовків спотворена в бік коротшої довжини з різким піком біля 5 слів. У більшості заголовків кількість слів становить від 4 до 8 слів, причому значно менше заголовків містить менше 2 або більше 10 слів, і дуже небагато слів перевищують 12 слів.

Заголовки дуже лаконічні, що очікувано, оскільки вони повинні бути помітними та інформативними в межах обмеженої кількості слів.

Кількість слів для транскриптів демонструє довгостроковий розподіл із піком близько 2000 слів. Більшість транскриптів коливається від 1000 до 4000 слів, з деякими викидами, що перевищують 6000 слів, а розподіл поширюється приблизно до 12000 слів, що вказує на значну різницю в довжині транскриптів. Транскрипти дуже різняться за довжиною, що відображає різноманітний характер доповідей TED, які можуть варіюватися від коротких презентацій до більш розширених дискусій.

Ці розповсюдження дають уявлення про типову довжину різних компонентів контенту TED talk, корисні для розуміння та покращення кураторства вмісту та стратегії презентації.

Рисунок 2.2 містить три діаграми, що представляють розподіл кількості слів для summary (описи), title (заголовків) і transcript (транскрипцій) виступів TED. Кожен графік надає візуальний підсумок розподілу даних, включаючи медіану, квартилі та статистичні викиди.

Коробковий графік для описів показує, що середня кількість слів становить близько 60 слів. Міжквартильний діапазон (IQR), який представляє середні 50% даних, охоплює приблизно від 40 до 80 слів. Вуса охоплюють приблизно від 20 до 100 слів, вказуючи на діапазон, куди потрапляє більшість описів. Є кілька викидів понад 100 слів, деякі з них перевищують 150 слів, а один помітний викид становить близько 200 слів.

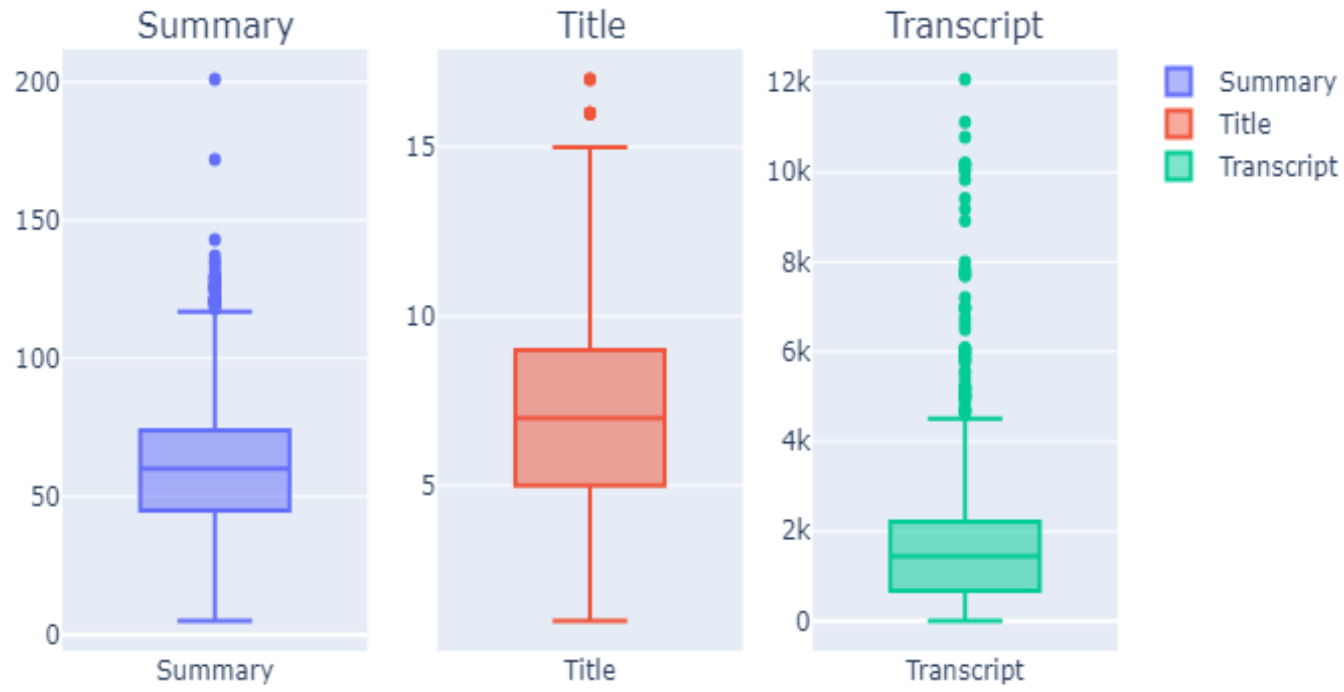

Рисунок 2.2 – Діаграма розподілу кількості слів в колонках summary, title, transcript

Для заголовків середня кількість слів становить близько 6 слів. IQR коливається приблизно від 4 до 8 слів, з вусами приблизно від 2 до 10 слів. Це свідчить про те, що більшість назв досить лаконічні. Є кілька викидів понад 10 слів, з максимальним викидом близько 15 слів.

Діаграма стенограми показує середню кількість слів близько 2000 слів. IQR коливається приблизно від 1000 до 4000 слів, що вказує на значний розкид серед 50% даних. Вуса мають приблизно від 500 до 6000 слів, демонструючи широкий діапазон довжин транскриптів. Існує багато викидів за межами 6000 слів, деякі з них досягають приблизно 12000 слів.

Рисунки 2.3, 2.4, 2.5 містять хмари слів, які представляють слова, які найчастіше зустрічаються в описах, заголовках і транскрипціях виступів TED. Хмари слів візуально підкреслюють слова залежно від їх частоти, при цьому більші слова з’являються в тексті частіше.

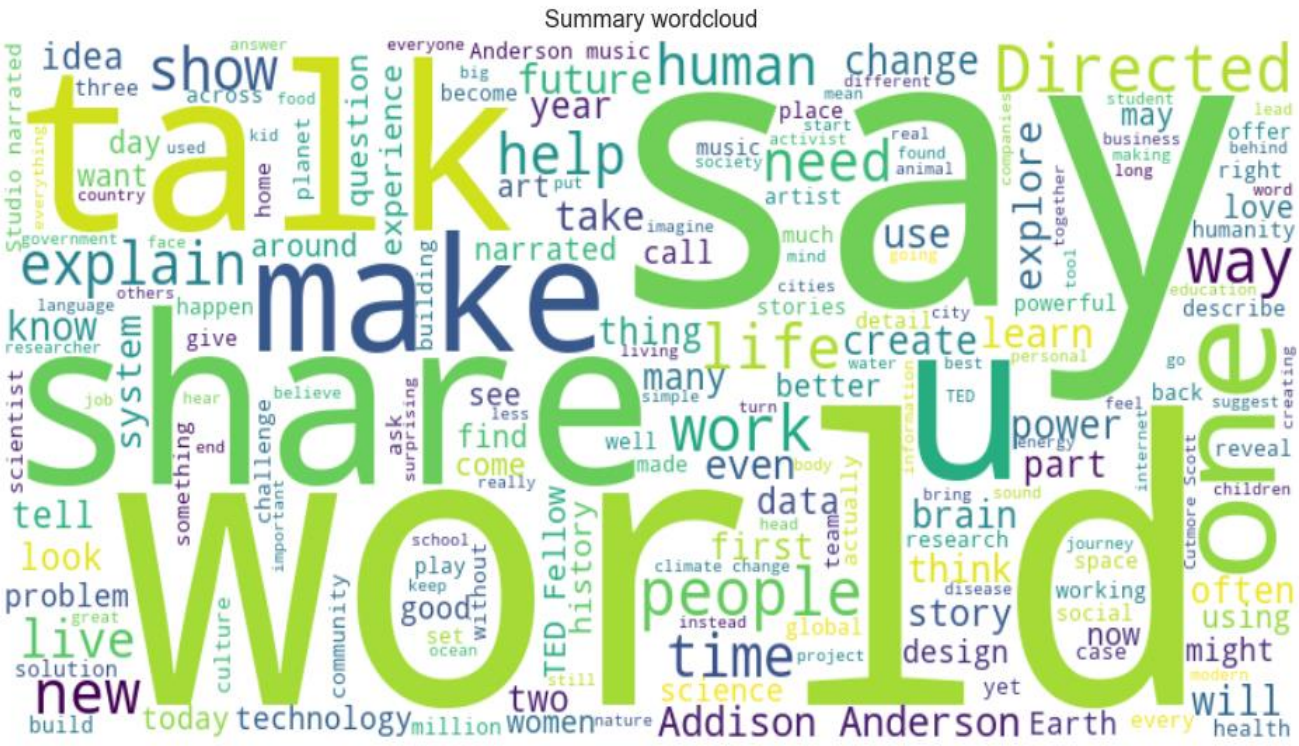

Рисунок 2.3 – Графік wordcloud текстової колонки summary

Рисунок 2.4 – Графік wordcloud текстової колонки title

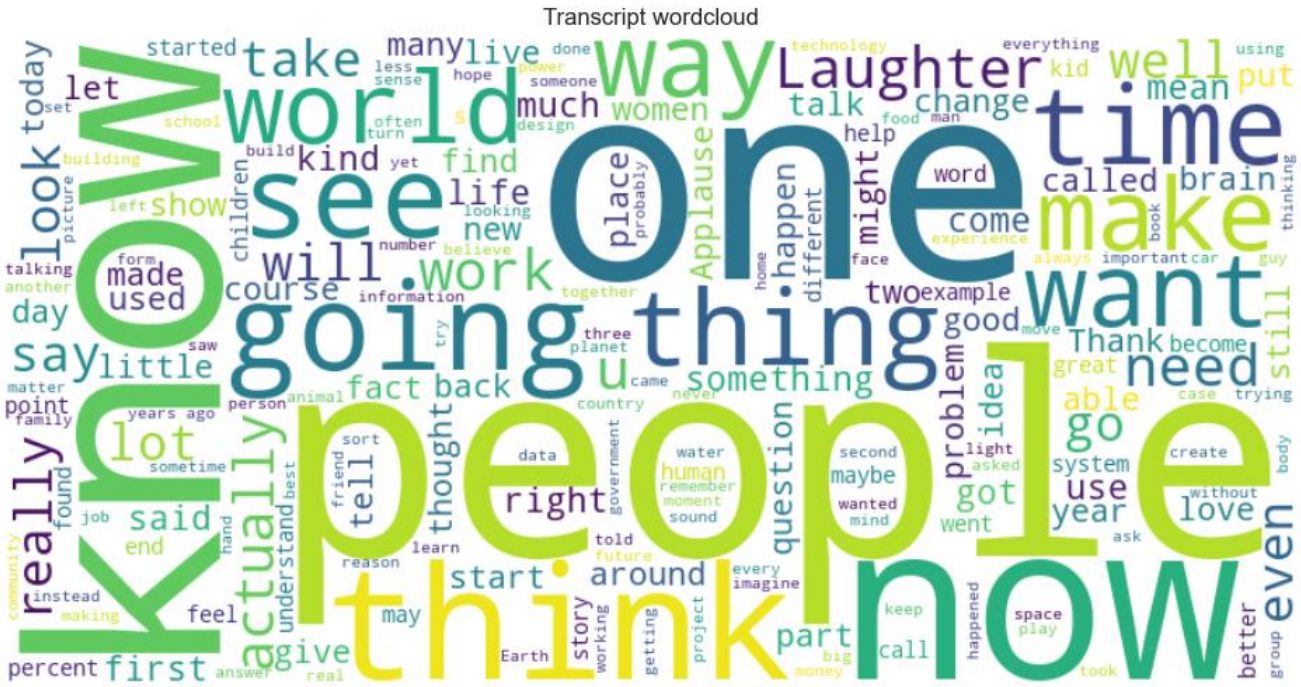

Рисунок 2.5 – Графік wordcloud текстової колонки transcript

У хмарі слів по колонці summary найпомітніші слова включають «talk», «world», «share», «make» та «say». Ці слова свідчать про те, що описи доповідей TED часто зосереджені на акті обміну ідеями, обговоренні глобальних проблем і висловленні точок зору. Інші слова, які часто зустрічаються, як-от «human», «life», «work», «learn» та «help», вказують на те, що у описах зазвичай розглядаються теми, пов’язані з людським досвідом, освітою та допомогою. Наявність таких слів, як «explain», «explore» та «create», підкреслює інформативний та дослідницький характер описів.

Хмара слів по колонці title показує, що найпоширенішими словами є «world», «future», «life», «make» та «human». Це свідчить про те, що в назвах доповідей TED часто наголошується на темах глобального значення, дискусіях, орієнтованих на майбутнє, і темах, пов’язаних з людиною. Такі слова, як «power», «work», «help», «new» і «way» додатково вказують на те, що назви часто пов’язані з розширенням можливостей, професійним життям, інноваціями та вирішенням проблем. Повторення таких слів, як «design», «need», «art» та «solve», відображає творчий характер багатьох виступів TED.

У хмарі слів по колонці transcript найпомітніші слова включають «people», «know», «one», «world» і «time». Це вказує на те, що транскрипти доповідей TED часто обговорюють людей, знання та плин часу в глобальному контексті. Інші часто зустрічаються слова, як-от «think», «make», «going», «work» та «want», свідчать про зосередженість на процесах мислення, діях і прагненнях. Наявність таких слів, як «see», «take», «thing», «life» та «right» підкреслює описовий і директивний характер промов.

Для даних, таких як related_videos, speakers, subtitle_languages, topics, які структуровані як складна вкладена інформація, скрейпер аналізував їх у рядки у форматі JSON. Цей структурований формат гарантував, що даними можна було легко маніпулювати та аналізувати їх на наступних етапах.

Після вилучення та аналізу даних з кожної сторінки обговорення TED скрейпер зберігав аналізовані дані у файл CSV. Цей формат файлу було обрано через його простоту та сумісність із різними інструментами аналізу даних. Кожен рядок у файлі CSV відповідав виступу TED, а стовпці представляли різні точки даних, отримані зі сторінок. Остаточний набір даних, зібраний таким чином, надав багате та багатогранне уявлення про виступи TED, охоплюючи кількісні показники, такі як перегляди та лайки.

Щоб забезпечити якість і цілісність набору даних, протягом процесу збирання було реалізовано кілька етапів перевірки. Були проведені перевірки узгодженості даних, щоб виявити та виправити будь-які розбіжності чи аномалії. Скрепер періодично тестувався та вдосконалювався для адаптації до будь-яких змін у структурі веб-сайту, забезпечуючи постійну точність і надійність зібраних даних.

2.4 Висновки до другого розділу

Отриманий набір даних містить всебічну інформацію про виступи, включно з тривалістю, кількістю переглядів, мовами субтитрів, змістовним описом, тощо. Дані для системи було отримано завдяки процесу веб скрейпінг, який надсилає

HTTP-запити до сторінки та парсить HTML код на наявність даних про TED виступ. Дані про виступ зберігаються у зручному форматі CSV.

Вхідними даними є пошукові запити користувачів природньою мовою, що містять інформацію про бажані TED виступи.

Вихідими даними є знайдені релевантні виступи з необхідною інформацію про сам виступ.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

Загальним завданням є пошук схожих доповідей TED по запиту користувача природною мовою. Першочерговим етапом є попередня обробка загального набору даних по якому буде проводитись векторний пошук. Це включає векторизацію текстових даних та наповнення векторної бази даних. Запит користувача автоматично проходить процес векторизації, тобто перетворення у N-вимірний вектор чисел з плаваючою комою.

Для вирішенні задачі векторизації даних та запитів використовується модель вбудувань Gecko [23] від Google. Завдяки передовим технікам донавчання та неймовірно великим обсягам тренувальних даних модель здатна чітко підхоплювати семантичне наповнення речень.

Запит користувача у векторній формі співсталяється з існуючими векторами у базі даних для знаходження найбільш подібних записів.

3.2 Математична постановка задачі

На вхід системи надходить запит користувача. Метою векторного пошуку є знаходження векторів, що мінімізують метрику відстані між вектором запиту та векторами у базі даних.

Введемо основні визначення:

- Векторний простір v, який містить дійсні числа, наприклад R^d, де d – це розмірність вектору.
- Вектор запиту q для якого виконується пошук подібності.
- Набір векторів $D = \{v_1, v_2, \dots, v_d\}$, що містяться у базі даних Pinecone.
- Косинусна подібність між векторами q і v визначається як:

$$\frac{q \cdot v}{||q|| ||v||}, \tag{3.1}$$

де $q \cdot v$ – скалярний добуток векторів, а $||q||$ і $||v||$ – довжини векторів.

В такому випадку задача полягає у визначенні векторів, які мають максимальну подібність до вектору запиту q. Тоді векторний пошук визначається як:

$$v' = argmax_{D \in v_1, v_2, \ldots, v_d} \left(\frac{q \cdot v}{||q|| ||v||} \right), \quad (3.2)$$

де v' – знайдений вектор з найбільшою подібністю до вектору запиту q.

3.3 Обґрунтування методу розв’язання

Для вирішення поставленої задачі, необхідно визначитись з моделлю вбудувань, яка з неструктурованих текстових даних буде створювати контекстуальні вектори. Для отриманих векторів треба знайти векторну базу даних, яка надасть можливість зберігання та пошуку. Задля ефективного та точного пошуку по векторам треба визначитись з метрикою подібності (відстані) [8].

3.3.1 Обґрунтування вибору моделі вбудувань

Для виконання векторного пошуку необхідно перевести всі неструктуровані текстові дані у формат N-розмірного вектора. Цей процес називається векторизацією даних. Розвиток моделей вбудувань можна поділити на три етапи: вбудовування на рівні слова, вбудовування на рівні підслова, контекстні вбудовування. Розглянемо кожен з них більш детальніше.

Вбудовування на рівні слова. Початковий прорив у вбудовуванні тексту відбувся з появою вбудовування на рівні слів, де кожне слово у словнику представлено як щільний вектор у безперервному векторному просторі. Ці вкладення фіксують синтаксичні та семантичні зв’язки слів на основі їхніх характеристик розподілу у великих текстових корпусах:

− One-Hot Encoding: найпростіша форма вбудовування слів, де кожне слово представлено розрідженим вектором із «1» у позиції, що відповідає слову в словнику, та «0» в іншому місці. Цей метод, однак, не в змозі охопити будь-які семантичні чи синтаксичні зв’язки, а просто вказує на наявність слова у реченні.

− Моделі векторного простору: такі методи, як латентний семантичний аналіз (LSA), використовують сингулярну декомпозицію значень на матрицях текстових документів, щоб зменшити розмірність, фіксуючи приховані семантичні шаблони, але все ще не маючи дрібнозернистих семантичних зв’язків.

− Нейронні вбудування: справжня зміна парадигми відбулася з такими моделями, як Word2Vec [16] і GloVe [15]. Word2Vec (рисунок 3.1) використовує неглибоку структуру нейронної мережі, навчаючись за допомогою моделей безперервного пакета слів (CBOW), що намагається передбачити слово беручи до уваги контекст (сусідні слова), та моделі пропуску грам (Skip-gram), що передчабає контекст з огляду на слово.

–

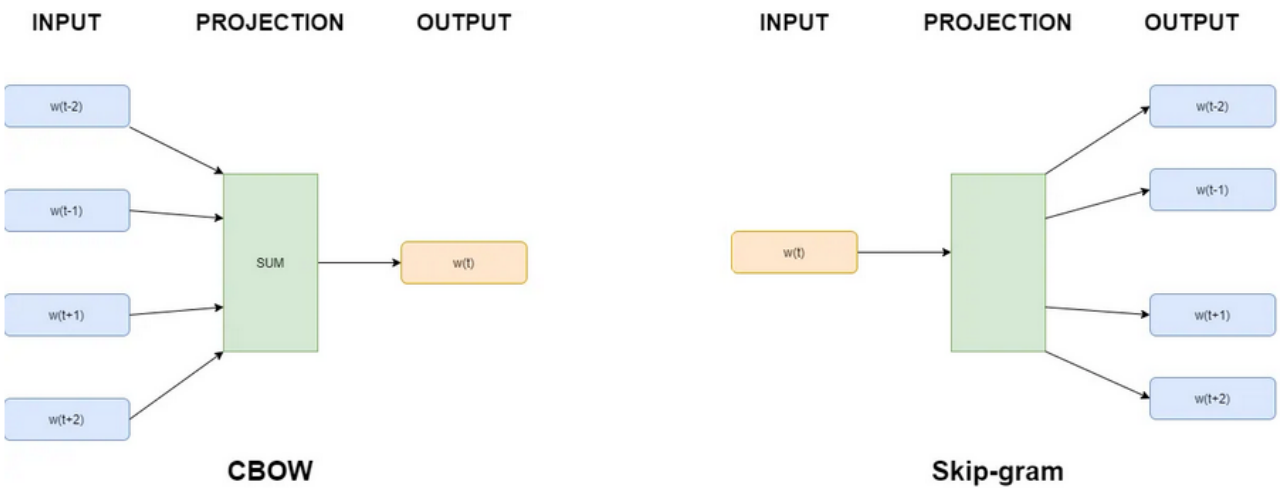

Рисунок 3.1 – Архітектура моделі вбудувань Word2Vec

GloVe (рисунок 3.2), з іншого боку, створює явну матрицю контексту слова (або спільної зустрічі слова) і розкладає її на множники, щоб отримати вбудовування, ефективно фіксуючи як глобальну статистику, так і локальний контекст.

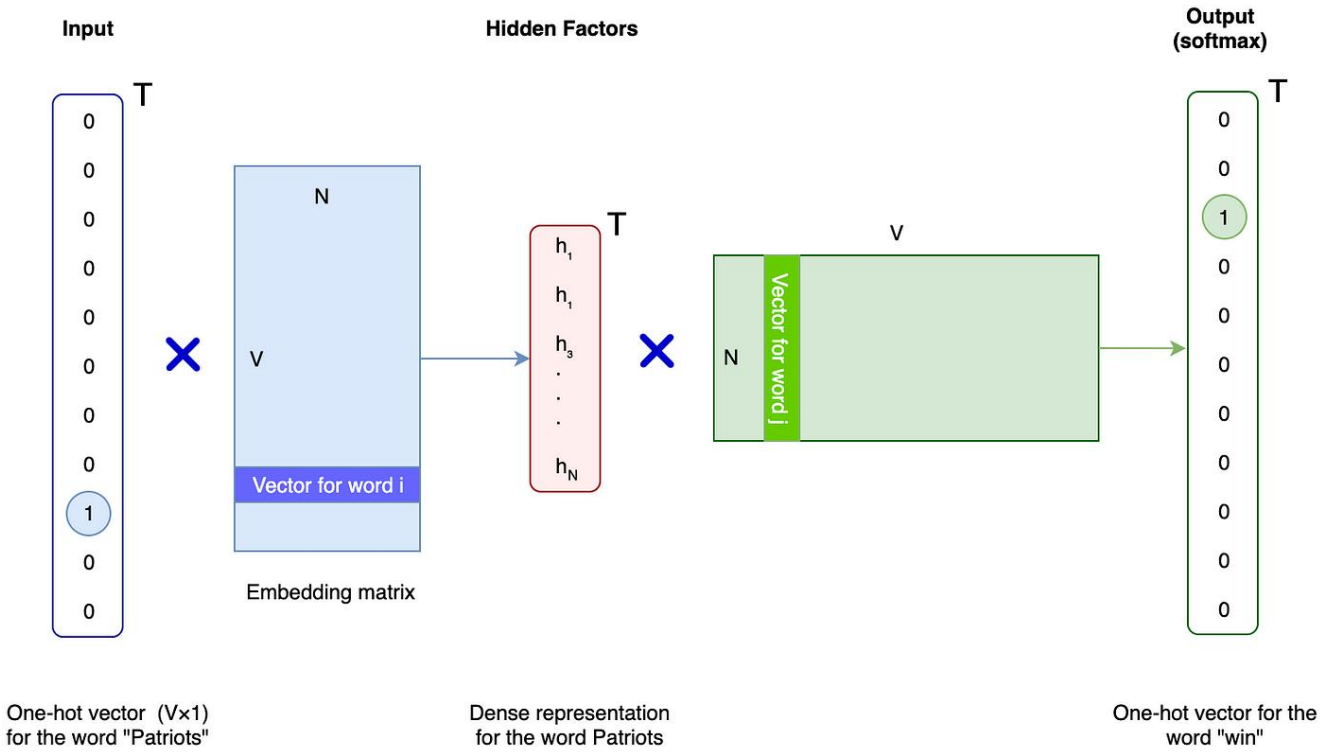

Рисунок 3.2 – Принцип роботи моделі GloVe

Ці моделі рівня слів суттєво покращили спосіб розуміння тексту машинами, уможлививши більш нюансовані програми NLP. Однак вони страждають від фундаментального обмеження: вони генерують єдине представлення для кожного слова, ігноруючи полісемію (слова, що мають кілька значень) і вплив контекстуальних нюансів.

Вбудовування на рівні підслова. Щоб усунути обмеження вбудовування на рівні слів, зокрема слів поза словниковим запасом (OOV), було розроблено вбудовування на рівні підслова.

– FastText: запроваджений Facebook AI Research, FastText [17] (рисунок 3.3) розширює модель skip-gram Word2Vec, розглядаючи кожне слово як складене з символьних n-грам. Це дозволяє моделі ділитися представленнями серед слів із загальними підрядками, таким чином дозволяючи їй краще обробляти рідкісні слова та генерувати вбудовування для слів OOV шляхом усереднення вбудовувань їхніх підслів.

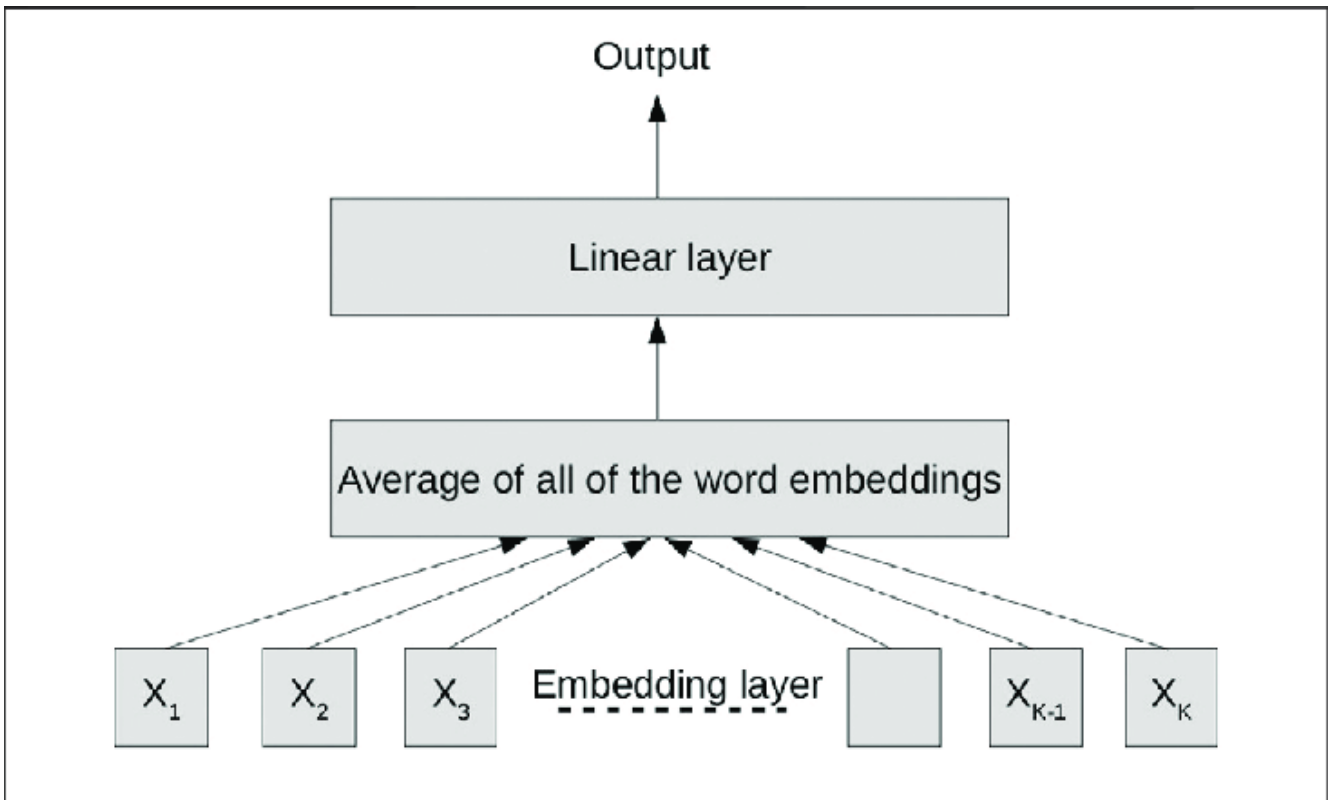

Рисунок 3.3 – Архітектура моделі вбудувань FastText

− Кодування пари байтів (BPE): Спочатку це була техніка стиснення даних, BPE була адаптована для NLP для керування одиницями підслів у нейронному машинному перекладі. Він поступово замінює найчастіші пари байтів у наборі даних на один невикористаний байт. У контексті вбудовування це дозволяє ефективно обробляти морфологічні варіації та неологізми, як показано в таких моделях, як токенизатор моделі BERT.

Контекстні вбудовування. Остання еволюція технологій вбудовування – це розробка контекстних моделей, які генерують динамічні вбудовування на основі контексту слів у реченні, таким чином ефективно вирішуючи проблему полісемії.

− ELMo (Embeddings from Language Models): ELMo [18] використовує глибоку, двонаправлену модель LSTM (рисунок 3.4), навчену на меті моделювання мови. Він генерує представлення слів, які є функціями всього вхідного речення, відрізняючись для того самого слова в різних контекстах.

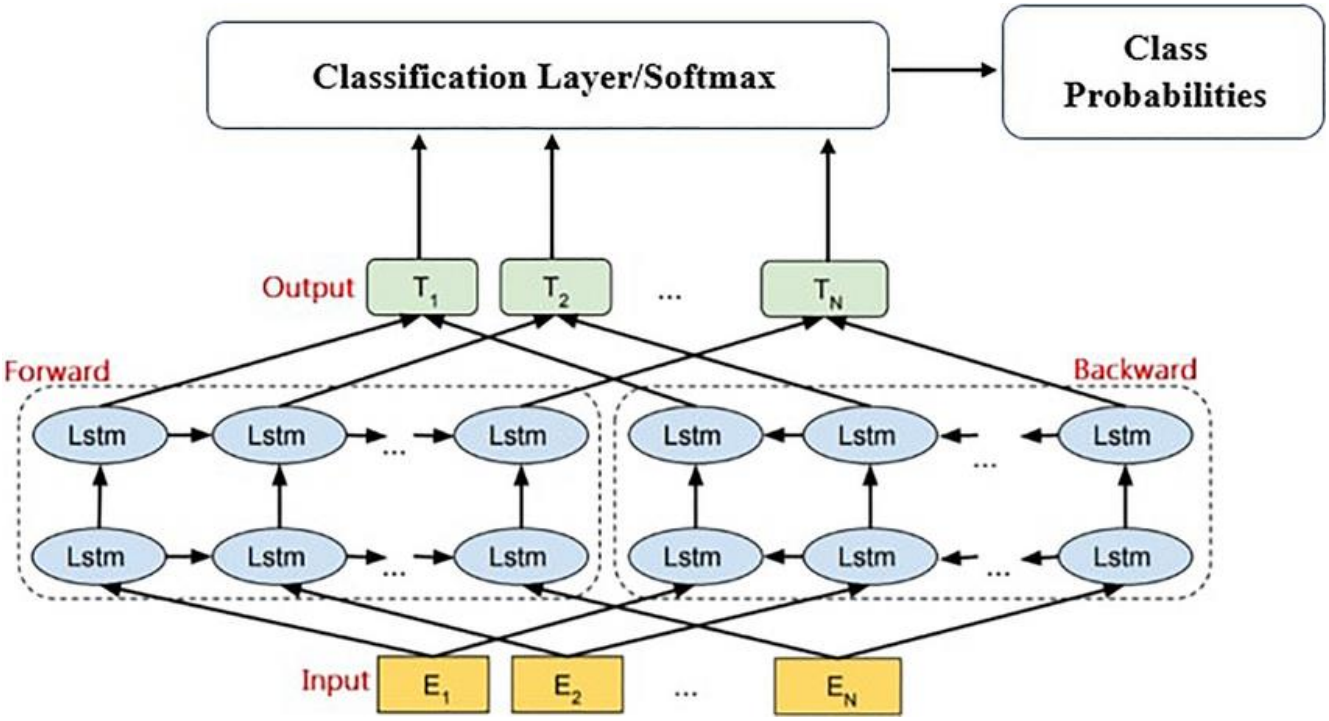

Рисунок 3.4 – Архітектура моделі ELMo

– Моделі на основі архітектури Transformer: впровадження архітектури Transformer [19] (рисунок 3.5) призвело до таких моделей, як BERT [20] та GPT [21], які використовують механізми уваги для зважування вплив різних слів у контексті. Ці моделі попередньо навчені на великих корпусах за допомогою таких завдань, як моделювання замаскованої мови та передбачення наступного речення, створюючи дуже залежні від контексту вбудовування.

– Моделі на основі архітектури Transformer називають Language Models (LM) або Large Language Models (LLM), залежно від кількості внутрішніх параметрів.

– Модель вбудовування Gecko від Google, яка базується на архітектурі Transformer, було обрано для створення вбудовувань для пошуку за семантичною подібністю завдяки її продуктивності, про що свідчить зайняте друге місце в рейтингу Massive Text Embedding Benchmark (MTEB) [22] у різних тестах завдання Semantic Text Similarity (STS). Високий рейтинг є показником здатності точно розуміти зміст, що робить його придатним для завдань, які вимагають точної оцінки подібності. Крім того, модель Gecko користується перевагами масштабних

досліджень і розробок Google у сфері мовних моделей, що гарантує, що вона є передовою та добре підтримується.

–

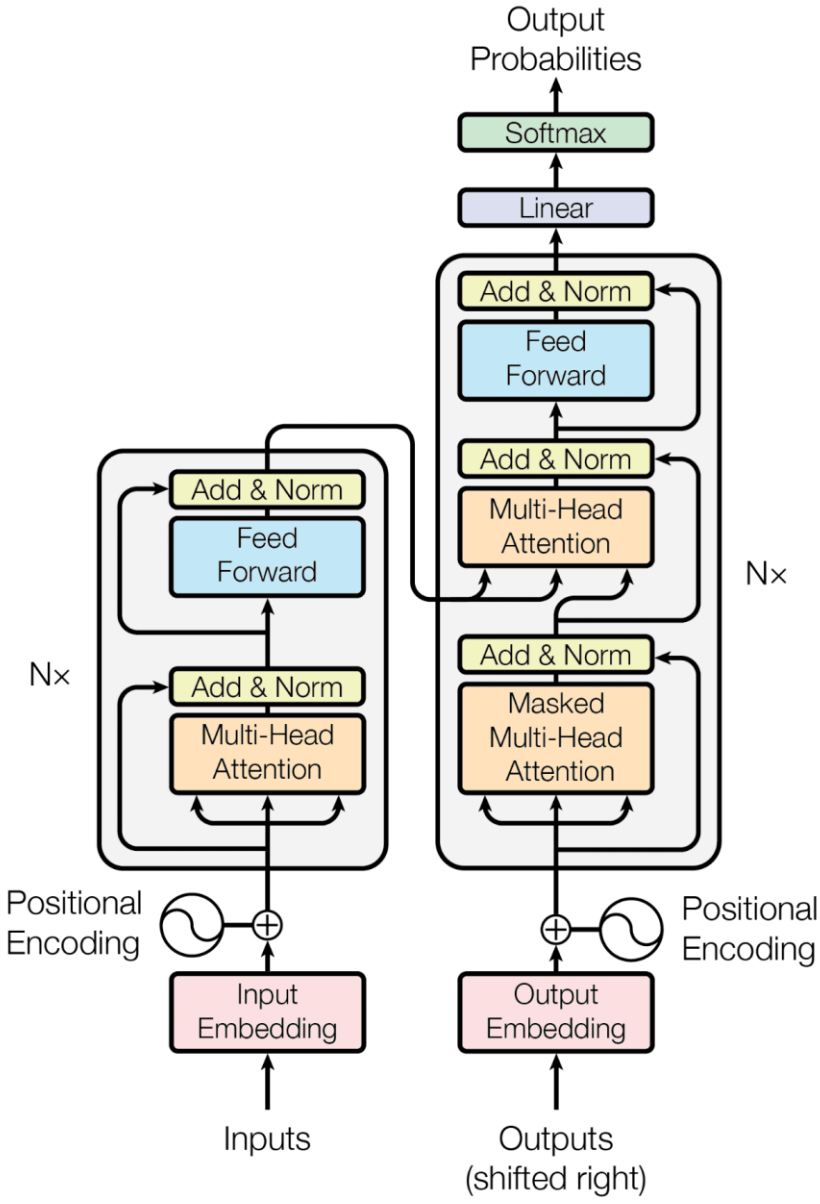

Рисунок 3.5 – Зображення архітектури Transformer

3.3.2 Обґрунтування вибору метрик подібності (відстані)

Коли ми маємо справу з пошуком векторної подібності в контексті машинного навчання та обробки природної мови, вибір метрики подібності є важливою складовою, оскільки він значно впливає на результати алгоритмів і їх

застосовність до різних типів даних і сценаріїв. Три загальні показники: евклідова відстань, косинусоїдна подібність і подібність скалярного добутку, кожен з яких має унікальні характеристики, які роблять їх придатними для конкретних завдань.

Евклідова відстань вимірює відстань по прямій лінії між двома точками у векторному просторі, як показано на рисунку 3.6. Цей метод інтуїтивно зрозумілий і ефективний у фізичному просторі, де важливі фактичні відстані. Однак у просторах великої розмірності, таких як ті, які часто зустрічаються в машинному навчанні, евклідова відстань може стати менш значущою. Це явище, відоме як «прокляття розмірності», призводить до сценарію, коли всі попарні відстані стають майже рівними, що ускладнює розрізнення між векторами лише на основі їх евклідової відстані.

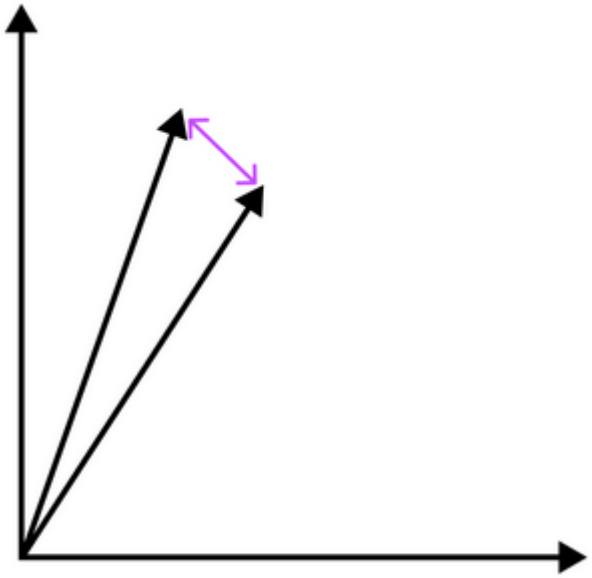

Рисунок 3.6 – Евклідова відстань

Косинусна подібність, навпаки, вимірює косинус кута між двома векторами, як показано на рисунку 3.7. Ця метрика особливо корисна, коли цікавить не величина векторів, а орієнтація. Наприклад, під час аналізу тексту надається перевага косинусній подібності, оскільки вона відображає, наскільки два документи схожі за змістом, незалежно від їх довжини. Порівнюючи кути, косинусна подібність ефективно нормалізує довжину документа, дозволяючи більш точно порівнювати їх вміст.

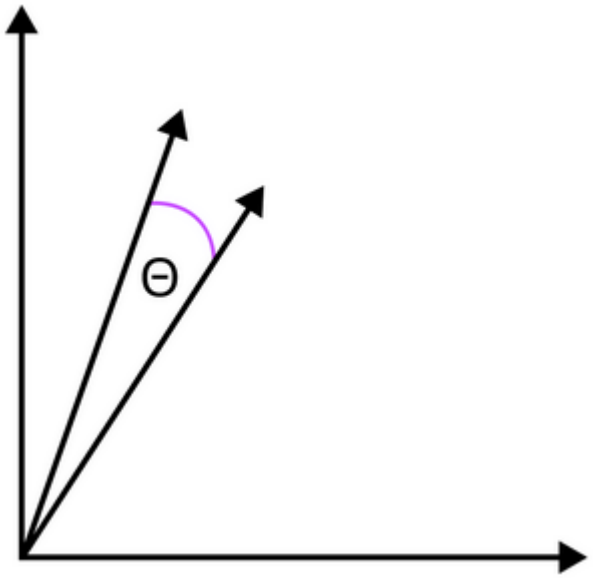

Рисунок 3.7 – Косинусна подібність

Подібність скалярного добутку, яка обчислює добуток величин двох векторів на косинус кута між ними, як показано на рисунку 3.8, тісно пов’язана з косинусною подібністю, але також враховує векторні величини. Ця метрика може бути корисною в ситуаціях, коли важливі як напрямок, так і величина векторів, наприклад, у системах рекомендацій, де як напрям (уподобання користувача), так і величина (сила уподобання) відіграють вирішальну роль.

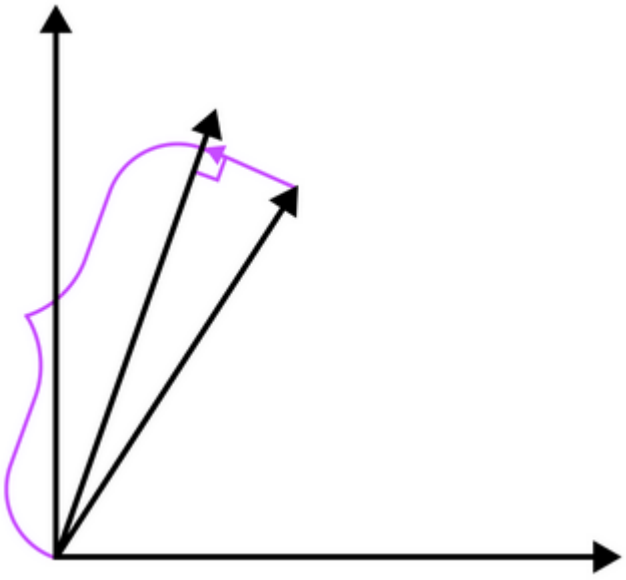

Рисунок 3.8 – Подібність скалярного добутку

Вибір косинусної подібності для векторного пошуку був обумовлений його здатністю забезпечити точнішу та значущу міру подібності в конкретному контексті різної довжини документа та простору ознак великої розмірності.

3.3.3 Обґрунтування вибору векторної бази даних

В пошуках векторної бази даних, яка задовольнить потреби проекту, було сформульовано технічні та операційні вимоги. Векторна база даних Pinecone повністю відповідає потребам та було обрана з наступних причин:

− Легкість інтеграції. Pinecone розроблено для простої інтеграції в існуючі системи, підтримуючи широкий спектр мов програмування та фреймворків. Ця сумісність значно скорочує час і зусилля, необхідні для інтеграції системи, дозволяючи розробникам швидко підключати Pinecone до програм і каналів даних. Наявність повної документації та зручного для розробників API ще більше полегшує бездоганну інтеграцію, що робить його ідеальним вибором для організацій, які прагнуть покращити свою інфраструктуру даних з мінімальними збоями.

− Висока щвидкодія. Pinecone забезпечує високу швидкість роботи, що особливо важливо для додатків, які вимагають швидкого пошуку даних і аналітики в реальному часі. Його ефективна обробка векторних пошуків і операцій гарантує безперебійну та ефективну роботу наших програм із мінімальною затримкою.

− Хмарний хостинг. На відміну від FAISS, яка в основному працює як бібліотека для ефективного пошуку подібності та кластеризації щільних векторів і вимагає ручного налаштування та обслуговування апаратного забезпечення для розгортання хмари, Pinecone пропонує повністю керований хмарний сервіс. Це не тільки зменшує накладні витрати на керування фізичним сервером, але й спрощує процес розгортання. Завдяки Pinecone масштабування та обслуговування здійснюються самою службою. Цей керований хмарний сервіс є особливо корисним для організацій, які шукають безпроблемне, масштабоване та легкодоступне рішення векторної бази даних.

− Низька операційна вартість. Pinecone має налаштування Serverless [6] індексу, де використовується принцип multitenancy. Він передбачає спільне розташування даних користувачів на одному кластері з повною ізоляцією доступу до даних. Цей підхід вирішує проблему, що не всі користувачі активно використовують сервісом. Також наявна функція автоматичного масштабування до нових даних, що значно скорочує операційні витрати та зусилля за рахунок автоматизації керування основною інфраструктурою, що дозволяє нам зосередитися на нашому основному бізнесі, не турбуючись про підтримку бази даних. Крім того, Pinecone пропонує безкоштовний рівень, який особливо корисний для невеликих проектів або компаній на ранніх стадіях розробки. Цей рівень дозволяє нам використовувати послугу без будь-яких початкових інвестицій, додатково мінімізуючи наші операційні витрати, водночас користуючись перевагами потужного рішення векторної бази даних.

3.4 Висновки до третього розділу

У даному розділі було сформульовано змістовну і математичну постановку задачі векторного пошуку. Для успішного вирішення задачі необхідно було обрати 3 компоненти: модель вбудувань для векторизації текстових даних, векторну базу даних для зберігання та пошуку за векторами, а також метрику подібності (відстані), яка оцінить схожість вектора запиту та існуючих векторів. Після проведення дослідження та аналізу наявних варіантів для зазначених компонентів, було обрано відповідні засоби та інструменти: модель вбудувань text-embedding-gecko-004, векторна база даних Pinecone та косинусна подібність, як метрика оцінки схожості векторів.

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Інструменти що використовуються при розробці

4.1.1 Мова програмування

Вибираючи мову програмування для розробки програми для пошуку веб-контенту за допомогою векторних баз даних, було обрано мову програмування Python, адже вона є потужною мова програмування, відома своєю простотою, гнучкістю та широким набором бібліотек і фреймворків. Ось кілька причин, чому Python було обрано для цього проекту:

- Простота й читабельність: Python має чіткий синтаксис і стисліcть, що робить код читабельним і легким для розуміння розробниками.
- Екосистема: Python має велику кількість бібліотек і фреймворків, які дозволяють швидко розробляти різноманітні веб-додатки, включаючи програми для роботи з векторними базами даних.
- Загальне використання: Python широко використовується в різних сферах, включаючи Data Science, штучний інтелект, веб-розробку тощо, що робить його популярним вибором серед розробників.
- Підтримка векторних баз даних: Python має бібліотеки, які дозволяють працювати з векторними базами даних, що робить його чудовим вибором для розробки програм, які вимагають пошуку вмісту за допомогою векторної технології.
- Підтримка спільноти: Python має велике та активне співтовариство розробників, яке надає підтримку, документацію та відповіді на запитання, які сприяють швидкому розвитку та вирішенню проблем.

Враховуючи ці фактори, вибір мови програмування Python є раціональним при розробці програми для пошуку веб-контенту за допомогою векторних баз даних.

4.1.2 Інтегроване середовище розробки

IDE (Інтегроване середовище розробки) є програмним забезпеченням, яке надає розробникам зручний інтерфейс для написання, редагування, компіляції та налагодження програмного коду. Воно поєднує в собі різні інструменти і функції, які полегшують роботу розробників і збільшують продуктивність.

Для розробки було обрано PyCharm для розробки цього проекту з декількох причин:

- Підтримка Python: PyCharm є одним з найпопулярніших інтегрованих середовищ розробки для Python. Воно надає повну підтримку Python, включаючи автодоповнення коду, підсвічування синтаксису, налагодження програми та інші корисні функції.
- Функціональність: PyCharm має широкий набір інструментів і функцій, які полегшують розробку, тестування та керування Python-проєктами. Це дозволяє ефективно працювати з кодом і зосередитися на розробці.
- Інтеграція із системою керування версіями: PyCharm має підтримку різних систем контролю версій, таких як Git, SVN, Mercurial. Це дозволяє зручно працювати з репозиторіями, відстежувати зміни та співпрацювати з іншими розробниками.
- Зручний інтерфейс: PyCharm має інтуїтивно зрозумілий інтерфейс користувача, що дозволяє легко навігувати по проєкту і взаємодіяти з інструментами розробки без зайвих складнощів.
- Підтримка веб-розробки: PyCharm має інтегровану підтримку для веб-розробки, включаючи роботу з HTML, CSS, JavaScript та іншими технологіями, що робить його відмінним вибором для розробки веб-додатків.

Загалом, PyCharm є потужним інструментом для розробки проектів на Python і надає розробникам зручне та продуктивне середовище для роботи над програмним кодом (рисунок 4.1).

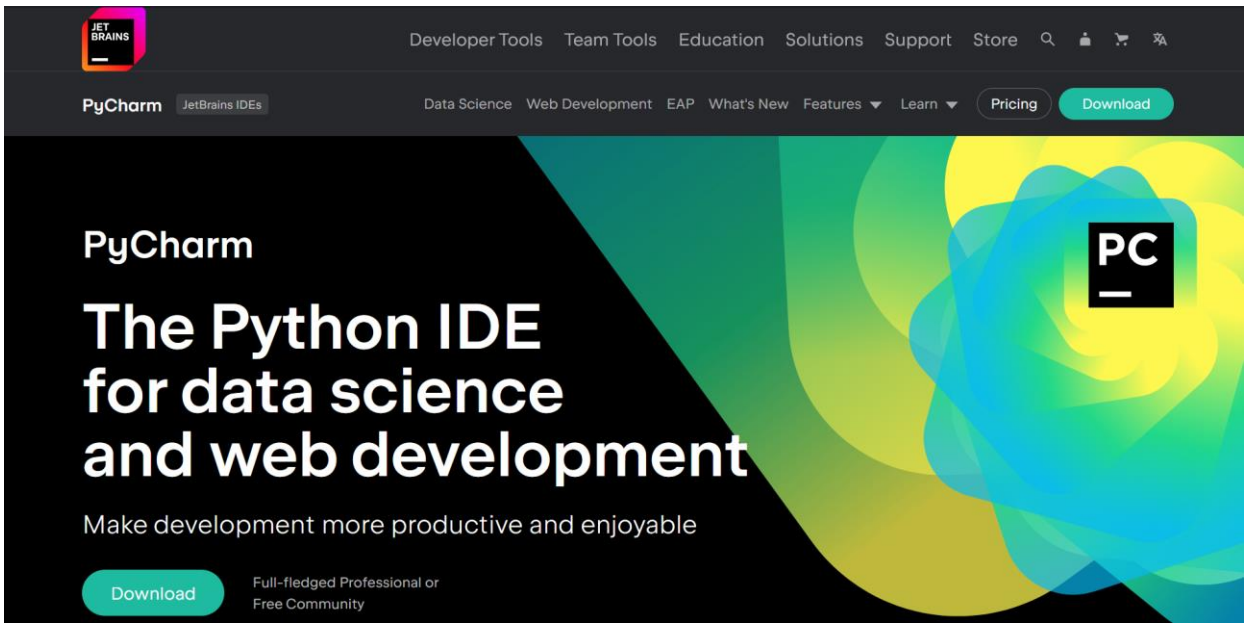

Рисунок 4.1 – Головна сторінка PyCharm

4.1.3 Встановлені Python бібліотеки

Для розробки додатку було встановлено наступні бібліотеки:

- pinecone-client – офіційна бібліотека Python для взаємодії з векторною базою даних Pinecone, яка використовується для створення та керування векторним індексом у програмах машинного навчання, що включають пошук подібності в просторі вбудовування. Він надає функції для взаємодії зі службою Pinecone, дозволяючи користувачам ефективно додавати, оновлювати, видаляти та запитувати векторні дані.

- Streamlit – це популярна бібліотека Python з відкритим вихідним кодом, яка використовується для створення спеціальних веб-програм для машинного навчання та аналізу даних. Вона призначена для швидкого й легкого створення прототипів інтерфейсів, які можна використовувати для демонстрації роботи алгоритмів машиного навчання або ж візуалізації даних.

- requests – це проста бібліотека, що надає зручний доступ до створення HTTP запитів.

- beautifulsoup4 – бібліотека для парсингу HTML та XML документів. Воно створює деревоподібну структуру з коду вебсайту, що надає можливість

ефективного та швидкого знаходження інформації по класам, тегам, тощо. В даному проекті її було використано для парсингу коду сторінок каталога та індивідуальних сторінок доповідей TED.

- pandas – швидка та потужна бібліотека для створення, маніпуляції та збереження табличних даних. Використовується для проміжного процесингу та чистки даних.

- tqdm – легка та мінімалістична бібліотека, що дозволяє огортати ітератори для динамічного відліку виконання коду.

- python-dotenv – маленька бібліотека, яка зчитує пари ключ-значення з файлу налаштування середовища. Використовується для скривання конфіденційної інформації, такої як: паролі, ключі API, значення змінних.

- llama-index – комплексна бібліотека, призначена для полегшення процесу інтеграції та роботи великих мовних моделей. Вона містить різноманітні інструменти, такі як з'єднувачі даних для прийому даних із багатьох джерел, індекси даних для структурування даних у форматах, оптимізованих для продуктивності, і механізми, які дозволяють здійснювати запити та взаємодію за допомогою запитів природною мовою.

- llama-index-vector-stores-pinecone – модуль для екосистеми llama-index, що служить сполучною ланкою між LlamaIndex і Pinecone, векторною системою баз даних. Це дозволяє LlamaIndex використовувати Pinecone для керування векторними даними, необхідними для завдань, які включають пошук подібності в просторах вбудовування.

- llama-index-embeddings-vertex – черговий модуль LlamaIndex зосереджений на інтеграції та управлінні вбудовуваннями з Google Cloud Vertex AI, керованою платформою машинного навчання Google. Абстрагує функціонал аутентифікації з Google Cloud, відсилання запитів на обробку вбудувань за високорівневими функціями.

Завантаження модулів, фреймворків, бібліотек виконувалось за допомогою вабсайту PyPI, що є головним репозиторієм бібліотек Python (рисунок 4.2).

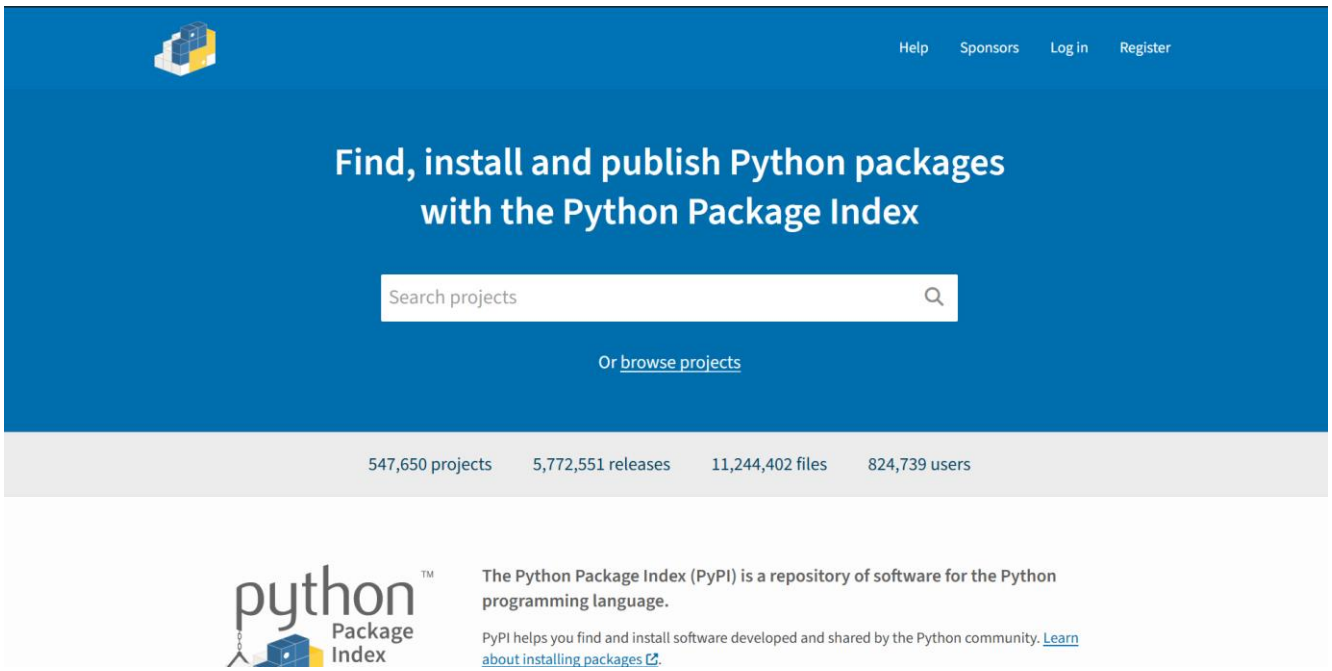

Рисунок 4.2 – Головна сторінка PyPI

4.2 Архітектура програмного забезпечення

Опис архітектури є важливою складовою розуміння функціоналу додатку. Вподальшому надано опис директорій проекту для орієнтуванні, таблицю назв функцій та опис їх функціоналу.

4.2.1 Структура директорій проекту

Структуру директорій додатку описано на рисунку 4.3.

У корневій папці проекту знаходяться наступні директорії:

1. *data* – у цій директорії зберігаються усі дані або файли необхідні для коректного функціонування додатку. Вона містить декілька піддиректорій:

- *pipeline_storage* – каталог для зберігання даних та кешу пов’язаних із додаванням записів у векторну базу даних.

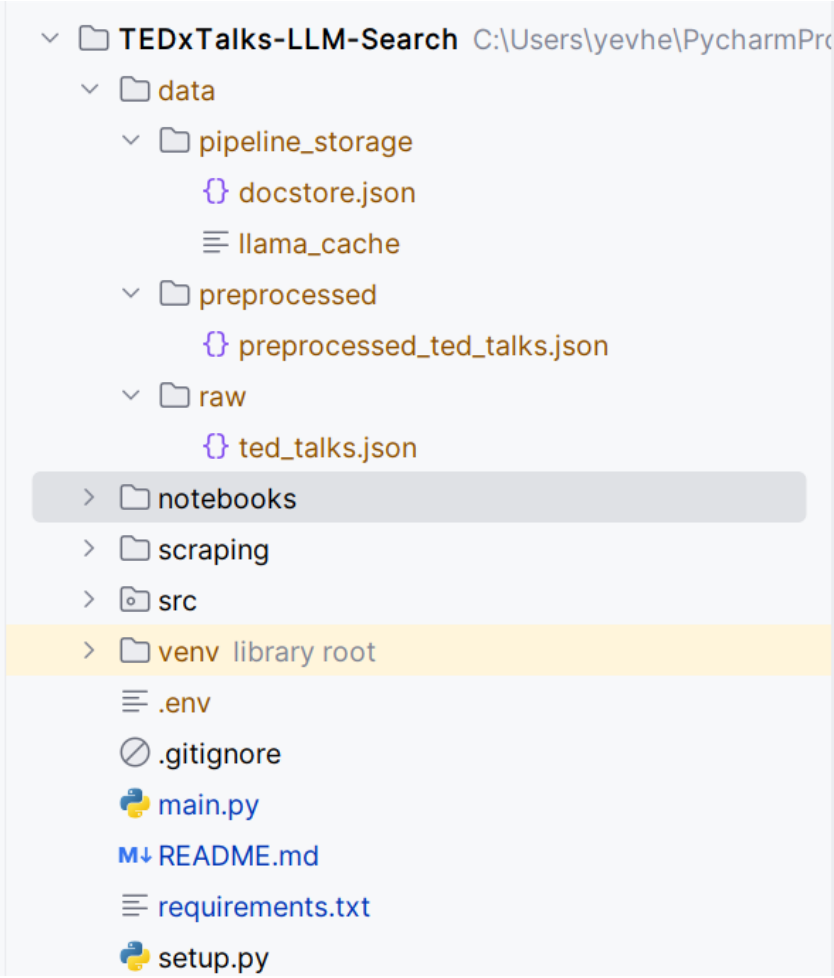

Рисунок 4.3 – Структура директорій готового проекту

- *preprocessed* – каталог, що містить дані, які пройшли чистку та обробку.
- *raw* – каталог, що містить необроблені дані.

2. *notebooks* – директорія у якій знаходяться jupyter ноутбуки, корисні для швидкої прототипізації.
3. *scraping* – каталог де знаходяться функції для веб скрейпінгу даних з вебсайту TED.
4. *src* – каталог, що містить програмну логіку веб-додатку
5. *venv* – каталог віртуального середовища, де встановлено залежності Python, для ізольоції від глобальних пакетів Python.

4.2.2 Специфікація функцій програмного забезпечення

Таблиця 4.1 – лістинг функцій веб-додатку та опис їх функціоналу

Назва	Опис
setup_environment	Налаштовує та активує віртуальне середовище.
check_prerequisites	Встановлює необхідні Python пакети з файлу requirements.txt у активоване віртуальне середовище.
save_dataframe_as_json	Зберігає pandas dataframe у вигляді JSON файлу.
load_df_from_gdrive	Завантажує pandas dataframe з CSV файлу на Google Drive та зберігає його у пам'яті.
convert_df_to_documents	Перетворює pandas dataframe на список об'єктів Document з llama-index.
create_index	Створює безсерверний індекс Pinecone.
populate_pinecone_db	Заповнює індекс Pinecone даними про TED talks.
create_pinecone_retriever	Створює векторний ретрівер.
remove_bracketed_text	Видаляє текст у дужках з вхідного тексту.
remove_special_chars	Видаляє спеціальні символи з вхідного тексту.
collapse_multiple_chars	Згладжує множинні повторення символів у вхідному тексті.
clean_text_column	Об'єднує функції очищення тексту.
process_rel_videos_column	Перетворює списки в колонці 'related_videos' у рядки, розділені комами.
process_speakers_column	Виділяє та розділяє дані 'speakers' у колонки 'speakers_names' і 'speakers_occupation', потім видаляє колонку 'speakers'.

Продовження таблиці 4.1

process_sub_lang_column	Перетворює 'subtitle_languages' зі списку словників у рядки, розділені комами.
process_topics_column	Розділяє дані 'topics' на колонки 'topics_names' і 'topics_ids', потім видаляє колонку 'topics'.
preprocess_raw_dataset	Попередньо обробляє сирий набір даних і зберігає його у вигляді JSON файлу.
retrieve_similar_talks	Отримує схожі виступи TED на основі запиту.
display_cards	Відображає знайдені виступи TED.
main	Головна функція для запуску Streamlit додатку.

Лістинг функцій додатку описано в таблиці 4.1.

4.3 Приклад роботи додатку

Користувацький інтерфейс було розроблено за допомогою бібліотеки streamlit. Вона надає доступ до готових елементів, що прискорює розгортання та створення функціональних прототипів для проектів машинного навчання та візуалізації даних. На рисунку 4.4 реалізовано поле вводу, що служить пошуковою строкою, де користувач може зазначити який виступ він намагається знайти.

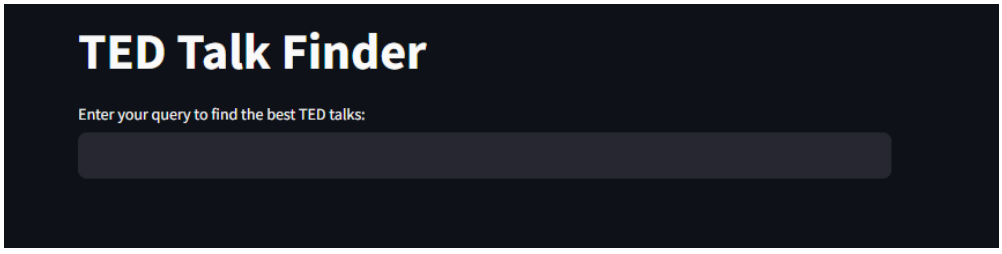

Рисунок 4.4 – Пошукова строка при вході на веб-додаток

Після надсилання запиту він обробляється серверною частиною, де текст перетворюється на вектор за допомогою моделі вбудовування тексту gecko-texm-embedding-004 від Google. Це перетворення є ключовим, оскільки воно переводить

текстові дані у формат, придатний для векторного пошуку. Процес вбудовування фіксує семантичне значення запиту, гарантуючи, що механізм пошуку може зрозуміти та точно відповідати намірам користувача.

Векторизований запит порівнюється з попередньо векторизованими виступами TED, які знаходяться у базі даних Pinecone, що підтримує автоматичне індексування та оптимізована для швидкого пошуку схожих елементів. Для цього обчислюється відстань між вектором запиту та векторами TED, віддаючи пріоритет розмовам із найменшими відстанями (найбільша подібність).

Записи, які вважаються найбільш схожими, динамічно вибираються та відображаються на екрані користувача. Кожен результат містить назву розмови та короткий опис, це можна побачити на рисунках 4.5 та 4.6

Рисунок 4.5 – Приклад знаходження виступів по запиту «russian aggression»

Рисунок 4.6 – Приклад знаходження виступів по запиту «How to beat procrastination»

4.4 Порівняння додатку з аналогами

Таблиця 4.2 містить порівняльну характеристику досліджуваного методу з аналогами.

Таблиця 4.2 – Порівняння функцій додатку з аналогами

Особливість	Пошук за ключовими словами	Фасетний пошук	Повнотекстовий пошук	Векторний пошук
Складність	Низька	Середня	Середня	Висока
Точність	Помірна	Висока	Помірна	Дуже висока

Продовження таблиці 4.2.

Швидкість	Висока	Середня	Середня	Змінна
Вимоги до вводу користувача	Конкретні ключові слова	Вибір із попередньо визначених категорій	Текстовий запит	Текстовий запит
Масштабованість	Низька	Низька	Помірна	Висока
Обробка синонімів	Немає	Немає	Немає	Присутня
Релевантність результатів	Помірна	Помірна	Помірна	Висока

4.5 Висновки до четвертого розділу

У даному розділі описано інструменти, бібліотеки та середовище програмування, яке використовувалось під час розробки програмної частини веб-додатку. Надано детальний опис директорій програми та функцій, які використовуються.

Продемонстровано результати роботи у вигляді користувацького інтерфейсу, де можна ввести запит англійською та отримати найбільш схожі за семантичним значення виступи TED, показуючи успішне відпрацювання алгоритму векторного пошуку.

ВИСНОВКИ

В бакалаврської дипломної роботи було досягнуто поставлену мету - покращено сервіси автоматизації та оптимізації пошуку веб-контенту за рахунок векторних баз даних, розроблено додаток автоматичного пошуку контенту вебсайту за допомогою векторних баз даних, що на відміну від існуючих методів пошуку контенту не потребує попереднього налаштування, працює по запиту користувача та є легкомасштабованою..

В ході розробки було використано низку бібліотек мови програмування Python, такі як: llama-index, що значно полегшив роботу пов’язану з пошуком за векторами та ефективною векторизацією даних, запитів користувачів, бібліотека pinecone в поєднанні з конекторами для llama-index розширили функціонал та надали можливість використовувати високорівневі класи та функції для легкої інтеграції, додавання записів у векторну базу даних та векторного пошуку.

Для простого налаштування та низьких операційних витрат рекомендується використовувати векторну базу даних Pinecone. Як хмарне рішення воно має чудову швидкодію, легку інтеграцію та автоматичне масштабування у випадку зростання кількість даних або ж кількості запитів.

Проведені дослідження по темі роботи показали актуальність впровадження подібних систем через стрімке зростання об’ємів інформації та складності налаштування пошуку контенту за допомогою звичайних фільтрів.

Порівнюючи запропоноване рішення із іншими методами пошуку інформації на вебсайтах, запропоноване рішення надає можливість точніше підхоплювати ідеї в складних пошукових запитах, тим самим покращуючи досвід користування вебсайтом.

Поставлене завдання є вирішеним, що підтверджується прикладами застосування розробленого веб-додатку для пошуку контенту на вебсайті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge URL: https://arxiv.org/pdf/2310.11703 (дата звернення 21.05.2024)

2. Vector Search with OpenAI Embeddings: Lucene Is All You Need URL: https://arxiv.org/pdf/2308.14963 (дата звернення 21.05.2024)

3. Порівняльний аналіз метрик відстані для векторного пошуку / Герасімов Є.Є., Богач І.В. // Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)» 2024», Вінниця, 2024. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21756 (дата звернення 21.05.2024)

4. Word Embeddings: A Survey URL: https://arxiv.org/pdf/1901.09069 (дата звернення 21.05.2024)

5. Calculating the similarity between words and sentences using a lexical database and corpus statistics URL: https://arxiv.org/pdf/1802.05667v2.pdf (дата звернення 21.05.2024)

6. A Research on Machine Learning Methods and Its Applications URL: https://www.researchgate.net/publication/327547625_A_Research_on_Machine_Learning_Methods_and_Its_Applications (дата звернення 21.05.2024)

7. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions URL: https://doi.org/10.1007/s42979-021-00815-1 (дата звернення 21.05.2024)

8. Approximate Nearest Neighbor Search in High Dimensions URL: https://arxiv.org/abs/1806.09823 (дата звернення 21.05.2024)

9. Bias in Word Embeddings URL: https://www.researchgate.net/publication/339612860_Bias_in_Word_Embeddings (дата звернення 21.05.2024)

10. The Curse of Dimensionality: Inside Out URL: https://www.researchgate.net/publication/327498046_The_Curse_of_Dimensionality_Inside_Out (дата звернення 21.05.2024)

11. OpenAI Embeddings Guide URL: https://platform.openai.com/docs/guides/embeddings (дата звернення 21.05.2024)

12. Vector database management systems: Fundamental concepts, use-cases, and current challenges URL: https://www.researchgate.net/publication/374157740_Vector_database_management_systems_Fundamental_concepts_use-cases_and_current_challenges (дата звернення 21.05.2024)

13. TED Talks Website URL: https://www.ted.com/talks (дата звернення 21.05.2024)

14. Natural Language Processing: State of The Art, Current Trends and Challenges URL: https://www.researchgate.net/publication/319164243_Natural_Language_Processing_State_of_The_Art_Current_Trends_and_Challenges (дата звернення 21.05.2024)

15. GloVe: Global Vectors for Word Representation URL: https://nlp.stanford.edu/pubs/glove.pdf (дата звернення 25.05.2024)

16. Efficient Estimation of Word Representations in Vector Space URL: https://arxiv.org/abs/1301.3781 (дата звернення 25.05.2024)

17. Enriching Word Vectors with Subword Information URL: https://arxiv.org/abs/1607.04606 (дата звернення 25.05.2024)

18. Deep contextualized word representations URL: https://arxiv.org/abs/1802.05365 (дата звернення 25.05.2024)

19. Attention Is All You Need URL: https://arxiv.org/abs/1706.03762 (дата звернення 25.05.2024)

20. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding URL: https://arxiv.org/abs/1810.04805 (дата звернення 25.05.2024)

21. Improving Language Understanding by Generative Pre-Training URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (дата звернення 25.05.2024)

22. MTEB: Massive Text Embedding Benchmark URL: https://arxiv.org/pdf/2210.07316 (дата звернення 25.05.2024)

23. Gecko: Versatile Text Embeddings Distilled from Large Language Models URL: https://arxiv.org/pdf/2403.20327 (дата звернення 25.05.2024)

ДОДАТКИ

Додаток А

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ДОДАТКУ АВТОМАТИЗОВАНОГО ПОШУКУ КОНТЕНТУ ВЕБСАЙТУ ЗА ДОПОМОГОЮ ВЕКТОРНИХ БАЗ ДАНИХ

Діаграма залежності модулів файлу main.py

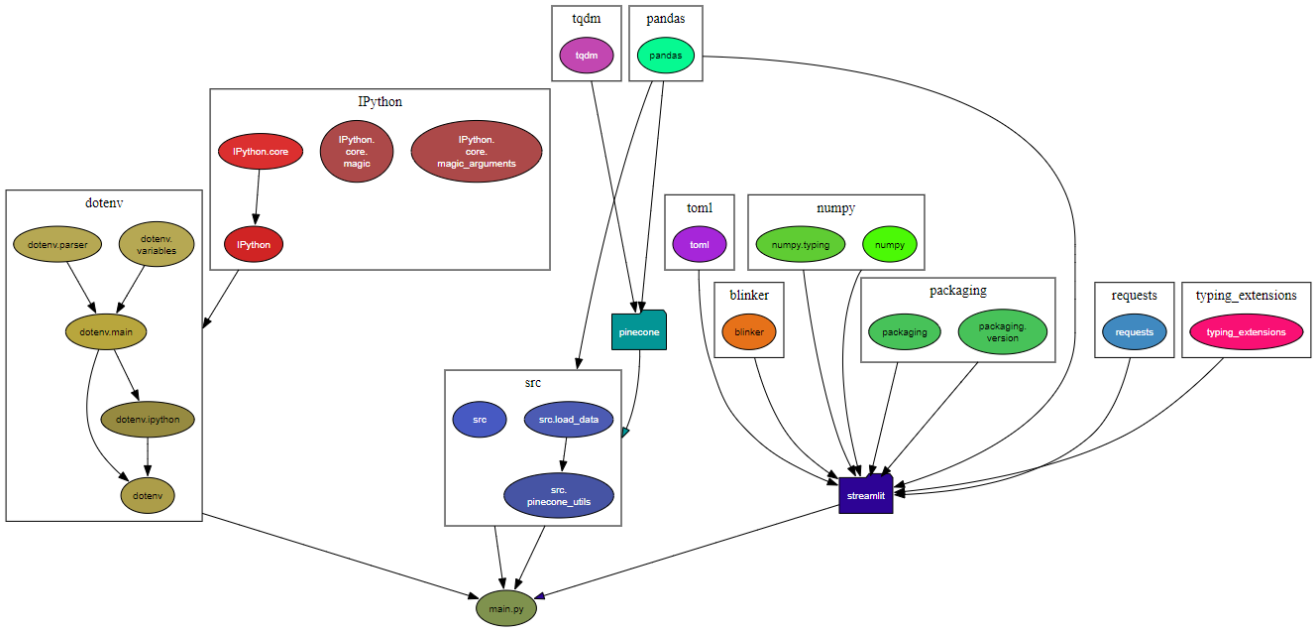

Рисунок А.1 – Діаграма залежності модулів файлу main.py

Діаграма залежності модулів файлу setup.py

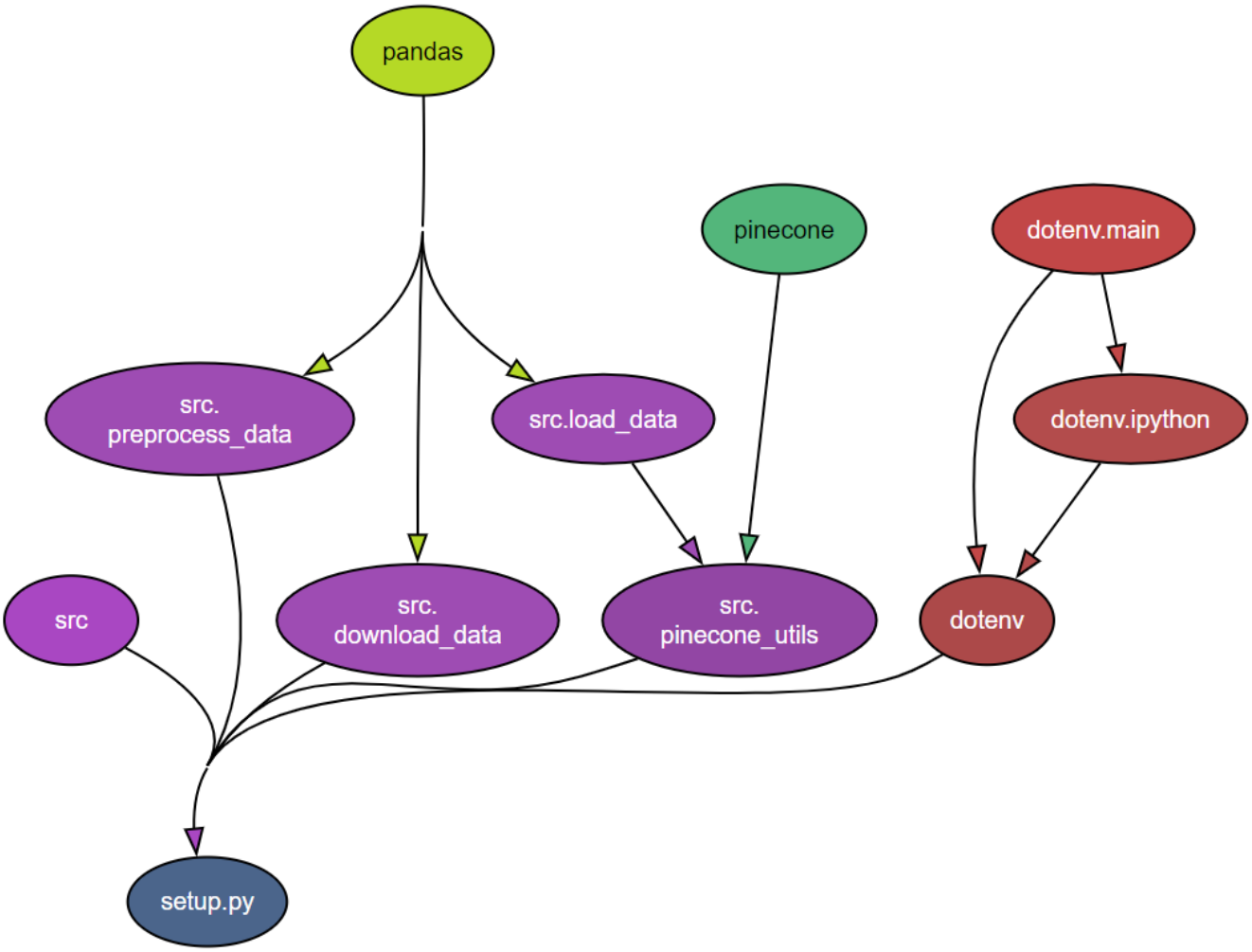

Рисунок А.2 – Діаграма залежності модулів файлу setup.py

Діаграма послідовності

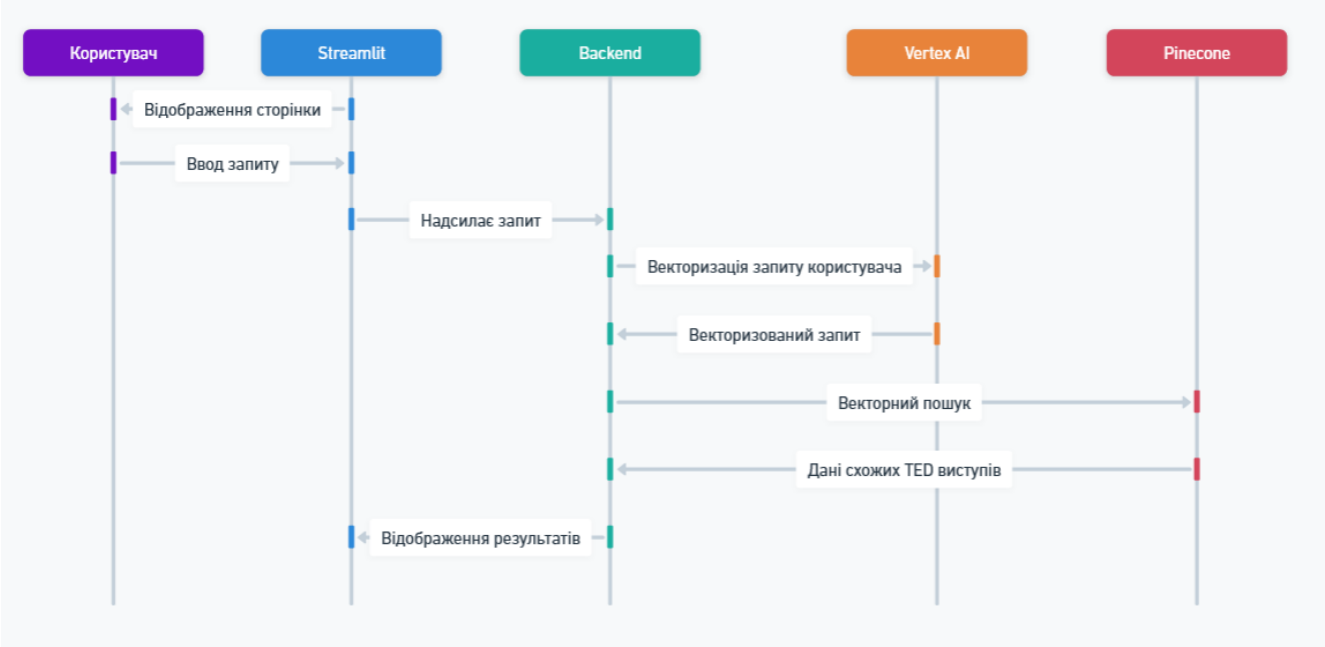

Рисунок А.3 – Діаграма послідовності

Блок-схема роботи веб-додатку

Рисунок А.4 – Блок-схема роботи веб-додатку

Додаток Б

(обов'язковий)

Лістинг основних функцій програми

#setup.py

```
import os
import subprocess
import sys
from dotenv import load_dotenv
from src.pinecone_utils import create_index, populate_pinecone_db
from src.download_data import load_df_from_gdrive, save_dataframe_as_json
from src.preprocess_data import preprocess_raw_dataset

load_dotenv(".env")


def setup_environment():
    """
    Set up and activate a virtual environment.
    """
    if not os.path.exists("venv"):
        subprocess.run([sys.executable, "-m", "venv", "venv"])
    # Activate the virtual environment
    activate_script = (
        os.path.join("venv", "Scripts", "activate_this.py")
        if os.name == "nt"
        else os.path.join("venv", "bin", "activate_this.py")
    )
    with open(activate_script, "rb") as file:
        exec(file.read(), dict(__file__=activate_script))

    print("Virtual environment successfully setup and activated.")


def check_prerequisites():
    """
    Install necessary Python packages from the requirements.txt file into the activated virtual environment.
    """
    subprocess.run([sys.executable, "-m", "pip", "install", "-r", "requirements.txt"])
    print("Prerequisites successfully installed.")


if __name__ == "__main__":
```

```
    setup_environment()
    check_prerequisites()

    raw_data_path = os.path.join("data", "raw", "ted_talks.json")
    if not os.path.exists(raw_data_path):
        df = load_df_from_gdrive()
        save_dataframe_as_json(df, raw_data_path)
    preprocess_raw_dataset(raw_data_path)

    create_index(dims=768, metric="cosine")
    populate_pinecone_db(
        os.path.join("data", "preprocessed", "preprocessed_ted_talks.json")
    )
```

```
#download_data.py

import os
import ast
import pandas as pd


def save_dataframe_as_json(df: pd.DataFrame, output_file_path: str):
    """
    Save a pandas dataframe as a JSON file.

    :param df: The pandas dataframe to save as JSON file.
    :param output_file_path: The path to save JSON file.
    """
    if not os.path.exists(os.path.dirname(output_file_path)):
        os.makedirs(os.path.dirname(output_file_path), exist_ok=True)

    df.to_json(output_file_path, orient="records")
    print("Dataset downloaded and saved as JSON file.")


def load_df_from_gdrive(file_id: str = "1AjkMy6kjvYGgFKivRpXaxV5e6FTo7HxN")
-> pd.DataFrame:
    """
    Loads a pandas dataframe from CSV file in Google Drive and stores it in-memory.

    :param file_id: The ID of the file to download.
    :return: A pandas dataframe.
    """
    URL = f"https://drive.google.com/uc?export=download&id={file_id}"
    try:
```

```
        df = pd.read_csv(
            URL,
            converters={
                'related_videos': ast.literal_eval,
                'speakers': ast.literal_eval,
                'subtitle_languages': ast.literal_eval,
                'topics': ast.literal_eval
            }
        )
    except Exception as e:
        print(f"Download failed: {e}")

    return df
```

#load_data.py

```
import pandas as pd
from llama_index.core import Document


def convert_df_to_documents(df_path: str) -> list[Document]:
    """
    Convert a pandas dataframe to a list of llama-index Document objects.
    :param df_path: path to local pandas dataframe
    :return:
    """
    df = pd.read_json(df_path)

    documents = [
        Document(
            # @TODO: Consider using other columns
            text=row["summary"],
            metadata={
                "page_url": row["page_url"],
                "title": row["title"],
                "duration": int(row["duration"]),
                "topics_list": row["topics_names"].split(", "),
            }
        )
        for _, row in df.iterrows()
    ]

    return documents
```

#pinecone_utils.py

```
import os
from pinecone import Pinecone, ServerlessSpec
from llama_index.core.ingestion import IngestionPipeline
from llama_index.core.storage.docstore import SimpleDocumentStore
from llama_index.vector_stores.pinecone import PineconeVectorStore
from llama_index.core.indices import VectorStoreIndex
from llama_index.core.retrievers import VectorIndexAutoRetriever,
VectorIndexRetriever
from llama_index.embeddings.vertex import VertexTextEmbedding,
VertexEmbeddingMode
from src.load_data import convert_df_to_documents


def create_index(dims: int, metric: str = "cosine"):
    """
    Create a serverless Pinecone index.
    :param dims: Number of dimensions of the vectors to be stored.
    :param metric: Distance metric used for similarity search.
    """
    pc = Pinecone(api_key=os.environ["PINECONE_API_KEY"])
    pc.create_index(
        name=os.environ["PINECONE_INDEX_NAME"],
        dimension=dims,
        metric=metric,
        spec=ServerlessSpec(cloud="aws", region="eu-west-1"),
    )
    print("Pinecone index is set up and ready.")


def populate_pinecone_db(json_path: str):
    """
    Populate a Pinecone index with TED talks data.

    :param json_path: Path to local JSON file containing TED talks data.
    """
    vector_store = PineconeVectorStore(
        api_key=os.environ["PINECONE_API_KEY"],
        index_name=os.environ["PINECONE_INDEX_NAME"],
    )
    pipeline = IngestionPipeline(
        transformations=[
            VertexTextEmbedding(
                project=os.environ.get("GCP_PROJECT_ID"),
                location=os.environ.get("GCP_LOCATION"),
                model_name="text-embedding-004",
```

```
                embed_mode=VertexEmbeddingMode.SEMANTIC_SIMILARITY_MODE,
            )
        ],
        vector_store=vector_store,
        docstore=SimpleDocumentStore(),
    )

    documents = convert_df_to_documents(json_path)
    pipeline.run(documents=documents, show_progress=True)

    # Persist the pipeline for future use, avoids duplicates when inserting new data
    pipeline.persist(os.path.join("data", "pipeline_storage"))

    print("Data has been successfully populated into the Pinecone database.")


def create_pinecone_retriever(top_k) -> VectorIndexRetriever:
    """
    Create a vector retriever.

    :return: VectorIndexRetriever object.
    """
    vector_store = PineconeVectorStore(
        api_key=os.environ["PINECONE_API_KEY"],
        index_name=os.environ["PINECONE_INDEX_NAME"]
    )
    index = VectorStoreIndex.from_vector_store(vector_store)
    retriever = VectorIndexRetriever(
        index=index,
        similarity_top_k=top_k,
        embed_model=VertexTextEmbedding(
            project=os.environ.get("GCP_PROJECT_ID"),
            location=os.environ.get("GCP_LOCATION"),
            model_name="text-embedding-004",
            embed_mode=VertexEmbeddingMode.SEMANTIC_SIMILARITY_MODE,
        )
    )

    return retriever

#preprocess_data.py

import re
import os
from ast import literal_eval
```

```
import pandas as pd


def remove_bracketed_text(input_text):
    """

    :param input_text:
    :return:
    """
    # Regex to find text inside round and square brackets
    pattern = r"[\(\[]([^()\[\]]*)[\)\]]"
    result_text = re.sub(pattern, "", input_text)

    return result_text


def remove_special_chars(input_text):
    """
    Remove special characters from input text.
    :param input_text:
    :return:
    """
    result_text = re.sub(r"\r|\n", "", input_text)

    return result_text


def collapse_multiple_chars(input_text):
    """
    Collapse multiple occurrences of characters in input text.
    :param input_text:
    :return:
    """
    # Collapse repeating whitespaces
    result_text = re.sub(r"\s+", " ", input_text)
    # Collapse repeating dashes
    result_text = re.sub(r"\-+", "-", result_text)
    # Collapse repeating dots
    result_text = re.sub(r"\.+", ".", result_text)

    return result_text


def clean_text_column(input_text):
    """
```

```
    Combine text cleaning functions.
    :param input_text: text to clean
    :return: cleaned text
    """
    input_text = remove_special_chars(input_text)
    input_text = remove_bracketed_text(input_text)
    input_text = collapse_multiple_chars(input_text)

    return input_text


def process_rel_videos_column(df: pd.DataFrame) -> pd.DataFrame:
    """
    Converts lists in 'related_videos' column to comma-separated strings.

    :param df: DataFrame with 'related_videos' lists.
    :return: DataFrame with updated 'related_videos' as strings.
    """
    df["related_videos"] = df["related_videos"].apply(lambda x: ", ".join(x))

    return df


def process_speakers_column(df: pd.DataFrame) -> pd.DataFrame:
    """
    Extracts and splits 'speakers' data into 'speakers_names' and 'speakers_occupation'
columns, then removes
    'speakers' column.

    :param df: DataFrame with 'speakers' data.
    :return: Updated DataFrame with new columns and without 'speakers'.
    """
    speakers_list = df["speakers"].to_list()
    speakers_names_list = []
    speakers_occupation_list = []
    for data in speakers_list:
        names_str = ", ".join([item["name"] for item in data])
        occupations_str = ", ".join([item["occupation"] for item in data])
        speakers_names_list.append(names_str)
        speakers_occupation_list.append(occupations_str)
    df["speakers_names"] = speakers_names_list
    df["speakers_occupation"] = speakers_occupation_list
    df.drop(columns=["speakers"], inplace=True)

    return df
```

```
def process_sub_lang_column(df: pd.DataFrame) -> pd.DataFrame:
    """
    Converts 'subtitle_languages' from list of dictionaries to comma-separated strings.

    :param df: DataFrame with 'subtitle_languages'.
    :return: DataFrame with 'subtitle_languages' as strings.
    """
    subtitle_list = df["subtitle_languages"].to_list()
    languages_list = []
    for data in subtitle_list:
        languages_str = ", ".join([item["name"] for item in data])
        languages_list.append(languages_str)
    df["subtitle_languages"] = languages_list

    return df


def process_topics_column(df: pd.DataFrame) -> pd.DataFrame:
    """
    Splits 'topics' data into 'topics_names' and 'topics_ids' columns, then removes 'topics'.

    :param df: DataFrame with 'topics' data.
    :return: Updated DataFrame with new columns and without 'topics'.
    """
    topics_list = df["topics"].to_list()
    topics_ids_list = []
    topics_names_list = []
    for data in topics_list:
        ids_str = ", ".join([item["id"] for item in data])
        names_str = ", ".join([item["name"] for item in data])
        topics_ids_list.append(ids_str)
        topics_names_list.append(names_str)
    df["topics_names"] = topics_names_list
    df["topics_ids"] = topics_ids_list
    df.drop(columns=["topics"], inplace=True)

    return df


def preprocess_raw_dataset(raw_json_path: str, output_dir: str = os.path.join("data",
"preprocessed")):
    df = pd.read_json(raw_json_path)
```

```
    df = process_rel_videos_column(df)
    df = process_speakers_column(df)
    df = process_topics_column(df)
    df = process_sub_lang_column(df)

    df["summary"] = df["summary"].apply(clean_text_column)

    df["transcript"] = df["transcript"].fillna("")
    df["transcript"] = df["transcript"].apply(clean_text_column)

    os.makedirs(output_dir, exist_ok=True)
    df.to_json(os.path.join(output_dir, "preprocessed_ted_talks.json"), orient="records")

    print("Data successfully preprocessed and saved.")
```

```
#main.py

import streamlit as st
from dotenv import load_dotenv
from src.pinecone_utils import create_pinecone_retriever

load_dotenv(".env")
retriever = create_pinecone_retriever(top_k=3)


def retrieve_similar_talks(query):
    results = retriever.retrieve(query)

    response = [
        {
            "title": result.metadata.get("title"),
            "page_url": result.metadata.get("page_url"),
            "summary": result.get_text()
        }
        for result in results
    ]
    return response


def display_cards(talks):
    # Create a grid of cards to display the talks
    cols = st.columns(3)
    for col, talk in zip(cols, talks):
        with col:
            st.subheader(f"[{talk.get('title')}]({talk.get('page_url')})")
```

```
            st.write(talk.get("summary"))


def main():
    st.title("TED Talk Finder")

    user_query = st.text_input("Enter your query to find the best TED talks:", "")

    if user_query:
        result_talks = retrieve_similar_talks(user_query)
        display_cards(result_talks)

if __name__ == "__main__":
    main()
```

70

Додаток В

(обов'язковий)

ПРОТОКОЛ

ПЕРЕВІРКИ ДИПЛОМНОЇ РОБОТИ

НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка додатку автоматизованого пошуку контенту вебсайту за допомогою векторних баз даних»

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ: кафедра АІІТ, ФІІТА, 1АКІТ-20б
(кафедра, факультет, навчальна група)

Науковий керівник: Богач І. В., доц. каф. АІІТ
(прізвище, ініціали, посада)

Показники звіту подібності Unicheck

Оригінальність 99.21 % Схожість 0.79 %

Аналіз звіту подібності (відмітити потрібне)

■ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень

Особа, відповідальна за перевірку _____ Овчинников К. В.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи

Автор _____ Герасімов Є. Є.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Богач І. В.
(підпис) (прізвище, ініціали)