ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Оптимізація задачі дедуплікації сайтів на основі аналізу метаданих»

на здобуття освітнього ступеня магістра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Костянтин САЛЮК
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Костянтин САЛЮК
група КНДМ-62

Керівник: Оксана ЗОЛОТУХІНА
науковий ступінь, вчене звання к.т.н., доцент

Рецензент:
науковий ступінь, вчене звання _____
(Ім'я, ПРІЗВИЩЕ)

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність 122 Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Салюку Костянтину Володимировичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Оптимізація задачі дедуплікації сайтів на основі аналізу метаданих

керівник кваліфікаційної роботи Оксана ЗОЛОТУХІНА к.т.н., доцент,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» 10.2023р. №145

2. Строк подання кваліфікаційної роботи «29» грудня 2023р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, документація бібліотек python.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області для задачі дедуплікації веб-ресурсів.
Огляд існуючих методів збору, обробки та порівняння веб-сайтів.
Розробка методики дедуплікації веб-сайтів на основі аналізу метаданих
Оцінка якості роботи розробленої системи

5. Перелік графічного матеріалу: *презентація*

1. Етапи вирішення задачі дедуплікації сайтів
2. Загальна архітектура системи
3. Архітектура мікросервісів системи
4. Оцінка продуктивності роботи системи
5. Оцінка якості визначення дублікатів

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз особливостей та вимог до вирішення задачі дедуплікації веб-сайтів	19.10-30.10.23	
2	Аналіз технологій агрегації даних веб-сайтів	30.10-13.11.23	
3	Аналіз методів вилучення та обробки інформації з веб-сайтів	03.11-07.11.23	
4	Аналіз методів порівняння контенту веб-сайтів з використанням метаданих	07.11-12.11.23	
5	Розробка методики дедуплікації сайтів на основі аналізу метаданих	12.11-15.11.23	
6	Розробка архітектури та програмних засобів системи для автоматизованого порівняння веб-сайтів на основі аналізу метаданих	15.11-08.12.23	
7	Експериментальні дослідження та оцінка результатів роботи	08.12-11.12.23	
8	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.23	
9	Розробка демонстраційних матеріалів	21.12-29.12.23	

Здобувач вищої освіти _____ Костянтин САЛЮК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Оксана ЗОЛОТУХІНА
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 59 стор., 2 табл., 26 рис., 31 джерел.

Мета роботи – оптимізація задачі дедуплікації веб-сайтів за рахунок автоматизації визначення подібності веб-сайтів на основі аналізу метаданих.

Об'єкт дослідження – процес визначення подібності веб-сайтів.

Предмет дослідження – методи та засоби вилучення та обробки інформації з веб-сайтів.

Короткий зміст роботи: У роботі проведено олгяд існуючих підходів до агрегації даних веб-сайтів. Проведено дослідження методів аналізу контенту веб-сайтів та визначення подібності цього контенту. Розроблено метод дедуплікації веб-сайтів на основі метаданих. На основі розробленої методики створено систему, яка виконує повний цикл аналізу подібності списку веб-сайтів починаючи з агрегації даних і закінчуючи фінальним результатом. Описано архітектурні особливості створеної системи, які дозволяють оптимізувати використання обчислювальних ресурсів та масштабувати систему для скорочення часу аналізу.

КЛЮЧОВІ СЛОВА: KEYWORDS EXTRACTION, NLP, NATURAL LANGUAGE PROCESSING, TEXT PROCESSING, TEXT VECTORIZATION, WEB-SITE SIMILARITY, МЕТАДАНІ, ОБРОБКА ПРИРОДНОЇ МОВИ

ABSTRACT

Text part of the master's qualification work: 59 pages, 2 tables, 26 pictures, 31 sources.

The purpose of the work – optimizing the web site deduplication task by automating the determination of website similarity based on metadata analysis.

Object of research – the process of determining the similarity of websites.

Subject of research – methods and tools of extracting and processing information from websites.

Summary of the work: The master's thesis thoroughly analyzes the problems that arise when solving the task of website deduplication. It also explores methods for optimizing time- and resource-intensive subtasks, which are the stages in determining the similarity coefficient between a pair of websites.

Existing approaches to website data aggregation have been evaluated as part of the research. The analysis showed the advantages and disadvantages of each method, identifying key aspects that should be considered when developing an effective deduplication mechanism.

In this master's thesis, a review of methods for analyzing website content and determining the similarity of this content was conducted. As a result, the key parameters and characteristics that should be considered when creating an effective metadata-based deduplication algorithm were identified.

Based on the research conclusions, a website deduplication methodology was developed based on the use of metadata to analyze and compare web resources. The developed methodology was implemented in a system that includes the entire cycle of analyzing the similarity of a list of websites, starting with data aggregation and ending with the final results.

The architectural features of the created system aimed at optimizing the use of computing resources and the ability to scale the system are covered in detail. This allows to achieve maximum productivity and efficiency when analyzing the similarity of large amounts of website data, reducing analysis time and resource consumption. And, in fact, it removes restrictions on the amount of input data.

The feasibility of using a microservice architecture was assessed and the extent to which such an architecture can reduce the processing time of various data sets was measured. The correlation between the processing speed and the number of handlers at each step of the system was also studied.

In the master's thesis, experimental tests were conducted to evaluate the results of the developed system. The quality of the site deduplication system was determined using the F-measure method. The obtained value of the F-measure (from 0.74 to 0.83, depending on the selected boundary value) indicates the success of using the developed system to solve the problem of website deduplication.

KEYWORDS: NLP, NATURAL LANGUAGE PROCESSING, NER, NAMED ENTITY RECOGNITION, DEFINITION OF NAMED ENTITIES, METADATA.

ЗМІСТ

ВСТУП

В сучасному інформаційному суспільстві проблема дублювання та схожості веб-сайтів виявляється актуальною і важливою. Задача дедуплікації, або виявлення схожих сайтів, набуває значущості у зв'язку із зростанням обсягів веб-контенту у геометричній прогресії. Зростаюча кількість сайтів, спрямованих на різні аудиторії, призводить до важливості виявлення та об'єднання схожих ресурсів для забезпечення користувачам якісного та унікального контенту.

Задача дедуплікації сайтів є нетривіальною через велику різноманітність вмісту та структур сайтів. Пошук схожості вимагає розробки ефективних методів аналізу та порівняння вмісту, враховуючи різноманітність взаємодії користувачів із веб-ресурсами. Крім того, неоднозначність визначення схожості та великий обсяг інформації вимагають удосконалення методів та підходів до розв'язання цієї проблеми.

У сучасному бізнесі, де відображення компаній у віртуальному просторі відіграє ключову роль, задача дедуплікації набуває особливого значення в галузі обробки даних CRM систем. У ситуаціях, де компанія має декілька різних сайтів, може виникнути проблема роздрібнення клієнтської інформації. Для CRM систем, які ведуть облік клієнтів, це може призвести до неправильного аналізу та стратегій, оскільки окремі сайти можуть розглядатись, як окремі компанії, тоді як насправді це єдиний суб'єкт.

Проведення дедуплікації сайтів у фінансовій сфері має безпосередній вплив на якість аналізу ринків, ефективність управління портфелем, а також на виявлення ризиків та ухвалення інвестиційних рішень. Збирання дубльованої чи неоднозначної інформації може призвести до невірного аналізу фінансових показників, що потенційно порушує нормальне функціонування фінансових установ та інвестиційних фондів.

Однією з основних стратегій при вирішенні завдання дедуплікації є аналіз текстового контенту. Цей метод базується на обробці та порівнянні текстової інформації, яка може включати в себе заголовки, мета-теги та основний текст

сторінок. Аналіз текстового контенту дозволяє визначити схожі теми, ключові слова та структуру тексту між різними веб-ресурсами.

У зв'язку із зростанням обсягу веб-даних та поширенням інтернет-ресурсів, вирішення проблеми ідентифікації та об'єднання подібних веб-сайтів стає нагальною вимогою. Таким чином, завдання дедуплікації веб-сайтів, що базується на метаданих, стає необхідним та актуальним в сучасному інформаційному середовищі.

Мета роботи – оптимізація задачі дедуплікації веб-сайтів за рахунок автоматизації визначення подібності веб-сайтів на основі аналізу метаданих.

Об'єкт дослідження – процес визначення подібності веб-сайтів.

Предмет дослідження – методи та засоби вилучення та обробки інформації з веб-сайтів.

Методи дослідження: математичні: статистичні методи; емпірико-теоретичні: абстрагування, аналіз, синтез, емпіричні методи, евристики, візуалізація; методи проектування та розробки програмного забезпечення; методи обробки текстів природною мовою.

Практична значущість результатів полягає в використанні розробленого програмного комплексу для аналізу списку веб-сайтів на наявність дублікатів.

Для досягнення мети вирішувались наступні завдання.

1. Аналіз методів збору інформації з великої кількості веб-сайтів.
2. Дослідження методів обробки інформації з веб-сайтів.
3. Аналіз підходів до екстракції ключових слів з веб-сайту.
4. Розробка методу визначення дублікатів веб-сайту на основі аналізу ключових слів та прийняття рішення на основі кордонного значення для коефіцієнту подібності сайтів.
5. Розробка програмного продукту, який буде реалізовувати розроблений метод. Програмний продукт має бути оптимізованим та масштабованим.
6. Оцінка продуктивності розробленої системи та якості результатів.

Основна частина роботи складається з трьох розділів, вступу та висновків.

У першому розділі роботи виконано аналіз наявних методів збору інформації в умовах, коли потрібно опрацювати величезний обсяг даних. Розглянуті різноманітні підходи до збору та обробки інформації з веб-сайтів, а також методи подальшого порівняння отриманої інформації.

Другий розділ розглядає розробку власної методики обробки контенту для ефективного вирішення проблеми дедуплікації списку веб-сайтів. У цьому розділі аналізуються, як аспекти обробки та аналізу текстової інформації, так і архітектурна оптимізація програмного продукту для досягнення результату за мінімальний час і з оптимальним використанням обчислювальних ресурсів.

Третій розділ присвячено оцінці результатів функціонування розробленої системи. В цьому розділі здійснюється оцінка не лише якості отриманих результатів, але й ефективності витраченого часу та обчислювальних ресурсів для отримання результатів порівняння веб-сайтів.

1 АНАЛІЗ МЕТОДІВ ТА ТЕХНОЛОГІЙ ДЕДУПЛІКАЦІЇ САЙТІВ

1.1 Загальна постановка задачі дедуплікації сайтів

Проблема дедуплікації сайтів представляє собою важливий аспект в області обробки та аналізу веб-даних. Відокремлення унікальних та схожих сторінок є важливим етапом у побудові надійних баз даних та виведення корисної інформації. Застосування методів машинного навчання, таких як кластеризація чи векторизація, може допомогти вирішити цю завдання, забезпечуючи точне і систематичне виділення надмірно схожих сайтів.

Ключовими викликами вирішення цієї задачі є висока різноманітність структури веб-сайтів та нестабільність форматів контенту. Для коректного аналізу і оцінки подібності веб сайтів потрібно враховувати, що сторінки веб-сайтів містять велику кількість HTML/CSS/JavaScript коду. Цей код з однієї сторони не містить суттєво важливої інформації для аналізу тексту, а з іншої - може впливати на визначення важливості того, чи іншого шматка тексту. Наприклад на деяких веб-сайтах додають невидемі блоки з інформацією, які можуть змінити результати аналізу. Власне кажучи такі блоки додаються для введення в оману пошукових систем, які також аналізують контент сайту.

Ще одним викликом є великий обсяг даних. Кожен сайт може мати необмежену кількість сторінок. Тож для аналізу подібності веб-сайтів система аналізу повинна обирати найбільш значущі сторінки з веб сайту.

Враховуючи обсяг даних, з яким буде працювати система, кожен з етапів повинен бути максимально оптимізованим при використанні ресурсів. А всю систему загалом має бути легко масштабувати відносно обсягу даних, який треба проаналізувати.

1.2 Аналіз технологій агрегації даних з веб сайтів

1.2.1 Бібліотека requests

Бібліотека requests для мови програмування Python є важливим інструментом, що дозволяє взаємодіяти з HTTP-протоколом та ефективно завантажувати дані з веб-сайтів.

Requests забезпечує простий та інтуїтивно зрозумілий інтерфейс для виконання HTTP-запитів, що спрощує процес взаємодії з веб-ресурсами. Вона позбавлена складних абстракцій, роблячи її привабливою для початківців та забезпечуючи зручність та гнучкість використання для досвідчених розробників.

Програмний код, що використовує бібліотеку requests, продемонстровано на рисунку 1.1.

```python
import requests
import json

from lxml import html
from pathlib import Path

DATA_PATH = Path(__file__).parent
URL = 'http://example.com'


def main():
    response = requests.get(URL)
    page_content = response.text

    tree = html.fromstring(page_content)
    title = tree.xpath('//title/text()')[0]

    with (DATA_PATH / 'data.json').open(mode='wt') as fp:
        json.dump(
            {
                'url': URL,
                'title': title,
            },
            fp,
        )

    with (DATA_PATH / 'content.html').open(mode='wt') as fp:
        fp.write(page_content)


if __name__ == '__main__':
    main()

```

Рисунок 1.1 – Приклад роботи бібліотеки requests

Цей код призначений для завантаження головної сторінки веб-сайту та дістає вміст тегу 'title'. Результат виконання зберігається у файлі. Весь контент сторінки також зберігається в окремому файлі.

Однією з переваг requests є його широка підтримка та активний розвиток, що робить її стандартом для використання в середовищі Python. Вона легко інтегрується з іншими бібліотеками та фреймворками, що полегшує створення комплексних програмних рішень.

Важливо враховувати, що requests не забезпечує обробку JavaScript, тому вона не може бути використана для веб-сторінок, які динамічно завантажують контент за допомогою JavaScript.

Додатково, при завданнях, які передбачають обробку великої кількості запитів або потребують асинхронної обробки виникає проблема з продуктивністю. Тому що ця бібліотека використовує синхронний підхід до роботи. Це можна дещо оптимізувати використовуючи обробку у потоках. Але мові програмування Python підтримка багатопоточності є лімітованою (порівняно з іншими мовами програмування). В один момент часу Python може виконувати тільки один потік. Використання requests у потоках дає приріст продуктивності порівняно з однопоточним використанням. Але при неправильному використанні цей приріст продуктивності може нівелюватися витратами на переключення між потоками.

1.2.2 Бібліотека aiohttp

Бібліотека aiohttp є важливим інструментом у мові програмування Python для взаємодії у сучасному веб-просторі. Вона базується на асинхронному підході та спрямована на ефективне використання ресурсів для одночасної обробки багатьох запитів, роблячи її особливо корисною для завдань, де швидкість та продуктивність є ключовими факторами.

Однією з головних переваг асинхронного підходу є покращення часу виконання завдань, оскільки можна продовжувати виконання інших частин

програми під час очікування завершення мережевих запитів. Це особливо корисно в сценаріях, де час реакції на дані важливий для досягнення ефективності.

Бібліотека aiohttp доволі універсальна і призначена для великого обсягу різноманітних задач. З її допомогою можна побудувати сервер, який буде приймати запити від інших сервісів. Ця бібліотека може виконувати роль фреймворка для побудови сучасного веб-сайту. Aiohttp можна також використовувати для мережевої взаємодії (асинхронна альтернатива бібліотеці requests). Завдяки своє модульності можна обрати для себе лише одну із функції цієї бібліотеки і не витрачати ресурси на завантаження та роботу непотрібного у даний момент коду.

На рисунку 1.2 представлений програмний код, що використовує бібліотеку aiohttp. Основна мета - продемонструвати, як за допомогою aiohttp завантажити головну сторінку веб-сайту та отриманні вмісту тегу 'title'. Результат виконання зазначеного коду зберігається у відповідні файли.

```python
import asyncio
import aiohttp
import json

from lxml import html
from pathlib import Path

DATA_PATH = Path(__file__).parent
URL = 'http://example.com'


async def main():
    async with aiohttp.ClientSession() as session:
        async with session.get(URL) as response:
            page_content = await response.text()

    tree = html.fromstring(page_content)
    title = tree.xpath('//title/text()')[0]

    with (DATA_PATH / 'data.json').open(mode='wt') as fp:
        json.dump(
            {
                'url': URL,
                'title': title,
            },
            fp,
        )

    with (DATA_PATH / 'content.html').open(mode='wt') as fp:
        fp.write(page_content)

if __name__ == '__main__':
    asyncio.run(main())

```

Рисунок 1.2 – Приклад роботи aiohttp

Перевагою використання aiohttp є також вбудована підтримка кукі, можливість налаштовування параметрів HTTP-запитів, вбудована підтримка WebSockets та інші зручні функції, що спрощують роботу з мережею і роблять її близькою до оптимальної.

Однак важливо враховувати, що aiohttp може вимагати певного рівня досвіду в асинхронному програмуванні, щоб оптимально використовувати його можливості.

1.2.3 Фреймворк Scrapy

Scrapy є високорівневим фреймворком для веб-парсингу, побудованим на мові програмування Python. Його архітектурна концепція базується на асинхронному та багатозадачному підході, що дозволяє ефективно обробляти численні запити та рекурсивно обходити сторінки веб-сайтів.

Однією з особливостей Scrapy є вбудована інтеграція із селекторами CSS та XPath, що дозволяє точно визначати та витягувати необхідні елементи з HTML-коду сторінок. Це спрощує процес екстракції структурованої інформації і дозволяє легко адаптувати код для різних веб-сайтів.

Важливою особливістю Scrapy є підтримка пайплайнів. Пайплайни використовуються для реалізації різноманітних завдань, таких як фільтрація, очистка, агрегація та збереження інформації. Один з головних принципів роботи пайплайнів - це послідовна обробка отриманих даних, що дозволяє ефективно взаємодіяти з результатами скрапінгу на різних етапах обробки. Правильне використання такого підходу дозволяє зробити функціонал системи лекорозширюваним. Все це надає розробникам велику гнучкість у створенні та вдосконаленні складних потоків обробки даних.

Приклад коду із застосуванням Scrapy представлено на рисунку 1.3. Цей код завантажує головну сторінку сайту та дістає контент тегу title. Результат роботи зберігається у файл. Також у файл зберігається весь контент сторінки.

```python
import scrapy
import json

from pathlib import Path
from typing import Any

from scrapy.http import Response


DATA_PATH = Path(__file__).parent


class ExampleSpider(scrapy.Spider):

    name = 'example_myspider'

    start_urls = ['http://example.com']

    def parse(self, response: Response, **kwargs: Any) -> Any:
        title = response.xpath('//title/text()').extract_first()

        with (DATA_PATH / 'data.json').open(mode='wt') as fp:
            json.dump(
                {
                    'url': self.start_urls,
                    'title': title,
                },
                fp,
            )

        with (DATA_PATH / 'content.html').open(mode='wt') as fp:
            fp.write(response.text)

```

Рисунок 1.3 – Приклад роботи Scrapy

Незважаючи на численні переваги, Scrapy не позбавлений певних недоліків, таких як відносна складність для новачків та обмежена підтримка JavaScript, що може вплинути на здатність фреймворку парсити динамічний контент.

Також Scrapy більше призначений для роботи із сайтами із заздалегідь відомою структурою. Коли потрібно із одноманітно-структурованих сайтів отримати певну інформацію. У випадку, коли потрібно просто зберігати весь контент сторінки із мінімальними маніпуляціями із контентом, використовувати досить велику бібліотеку не використовуючи більшу частину її можливостей може бути нераціонально.

1.2.4 Бібліотека BeautifulSoup

BeautifulSoup є потужною бібліотекою для обробки HTML та XML-документів у середовищі Python. Вона використовується для ефективного витягування інформації з веб-сторінок шляхом обробки HTML-коду та створення структурованого дерева об'єктів, яке спрощує навігацію та взаємодію з різними елементами сторінки.

Код, що використовує бібліотеку BeautifulSoup, можна побачити на рисунку 1.4. У даному прикладі відбувається завантаження головної сторінки веб-сайту, витягування контенту з тегу 'title', та збереження результату у файлі. Також весь вміст сторінки також фіксується у відповідному файлі.

```python
import requests
import json
from bs4 import BeautifulSoup
from pathlib import Path

DATA_PATH = Path(__file__).parent
URL = 'http://example.com'


def main():
    response = requests.get(URL)
    html_content = response.text

    soup = BeautifulSoup(html_content, 'html.parser')

    title = soup.select_one('title').text

    with (DATA_PATH / 'data.json').open(mode='wt') as fp:
        json.dump(
            {
                'url': URL,
                'title': title,
            },
            fp,
        )

    with (DATA_PATH / 'content.html').open(mode='wt') as fp:
        fp.write(html_content)


if __name__ == '__main__':
    main()

```

Рисунок 1.4 – Приклад роботи BeautifulSoup

Основна особливість BeautifulSoup полягає в зручному та інтуїтивно зрозумілому API, яке дозволяє швидко та легко здійснювати обхід та витягування даних із сторінок. Його функціональність допомагає виконувати операції з пошуку, фільтрації та маніпулювання елементами HTML зі зручністю.

Варто зазначити, що BeautifulSoup не має вбудованих засобів мережевої взаємодії. Тобто для скачування сторінки можна використовувати що завгодно. BeautifulSoup допомагає легко витягувати дані у структурованому вигляді.

Однак, при всіх перевагах, важливо враховувати, що BeautifulSoup пристосований для витягування даних із сторінок, але не призначений для складних завдань, де потрібна взаємодія з JavaScript-генерованим контентом. Також, він може бути менш ефективним у порівнянні з іншими інструментами для скрапінгу у випадках великої кількості даних.

1.3 Аналіз методів обробки інформації та порівняння веб сайтів

1.3.1 Метод хешування

Хеш-функції представляють собою алгоритми, які перетворюють вхідні дані різної довжини в фіксований хеш-код. Використання хеш-функції для аналізу контенту веб-сайтів дозволяє використовувати ресурси ефективно і підвищує швидкість аналізу. Основна ідея полягає в перетворенні великих об’ємів даних у короткі хеш-значення і подальше їх порівняння.

У роботі "Detecting Near-Duplicates for Web Crawling" [1] автори досліджують методи визначення схожості веб-сторінок для оптимізації процесу веб-індексації. Застосування хеш-функцій, таких як MinHash, дозволяє врахувати схожість без необхідності повного порівняння вмісту. Текст розбивається на множину слів, і для кожної множини обчислюється хеш-код. Визначаючи мінімальний хеш-код для кожної сторінки, можна ефективно визначити схожість між ними [2].

В контексті задачі дедуплікації веб-сайтів також використовують метод SimHash. Кожен термін у документі отримує свій вектор ваг. Хеш-функції використовуються для отримання хеш-коду для кожного вектора ваг, і схожість визначається на основі різниці хеш-кодів. SimHash використовує хеш-функції для створення "центроїду" (centroid) об'єкта. Спільно зберігає інформацію про схожість, але зменшує розмірність даних, використовуючи вектор з невеликою кількістю бітів.

Можна відзначити, що для методу SimHash характерна висока стійкість до невеликих змін у документах.

Ще одним підходом є використання Locality-Sensitive Hashing (LSH). У цьому методі документи представляються як вектори хеш-кодів, створених за допомогою різних хеш-функцій. Пари документів з подібними хеш-кодами розглядаються як кандидати на схожість [3]. Особливістю аналізу веб-сайтів за допомогою цього методу є використання локально чутливих хеш-функцій. Метод LSH є ефективним у великих наборах даних, оскільки дозволяє швидко виключити несхожі об'єкти та фокусуватися лише на потенційно схожих.

1.3.2 Метод TF - IDF

Аналіз частоти слів є підходом, спрямованим на вивчення та порівняння використання слів у текстах для визначення їх схожості чи відмінності. Цей метод стає особливо ефективним у виявленні дублікатів вмісту веб-сайтів, оскільки дозволяє визначити схожість не лише за структурою, але й за змістом тексту.

Аналіз частоти слів застосовується для побудови профілів вмісту веб-сайтів, визначення ключових слів та їх частоти в кожному тексті. Порівняння цих профілів дозволяє виявити подібність між веб-сайтами. Наукові дослідження вказують на успішність цього підходу, особливо в контексті дедуплікації текстового вмісту [2].

Серед методів, які використовуються для аналізу частоти слів одним із ключових методів аналізу є TF-IDF, що визначає важливість кожного слова у

документі в контексті колекції документів [24]. Цей метод став необхідним інструментом для розуміння та витягування смислової інформації з текстових даних.

TF-IDF складається з двох базових компонентів: Term Frequency (TF) та Inverse Document Frequency (IDF).

Term Frequency визначає частоту вживання конкретного терміну у конкретному документі. Розраховується, як кількість входжень терміну поділена на загальну кількість слів у документі.

Inverse Document Frequency вимірює, наскільки унікальним є термін у колекції документів. Вираховується як логарифм оберненої частини відношення загальної кількості документів до кількості документів, які містять термін.

Після цього значення TF-IDF обчислюється, як добуток двох множників: TF та IDF.

Такий показник вказує на вагомість терміну у конкретному документі та його важливість в рамках усієї колекції документів.

Метод TF-IDF знаходить широке застосування у визначенні схожості документів, ранжуванні пошукових результатів та кластеризації текстових даних [4]. Дослідження показують, що він ефективний для виділення ключових слів та виявлення схожих текстів у великих обсягах інформації [5].

Незважаючи на це метод TF-IDF має свої особливості і обмеження, такі як чутливість до довжини документу та відсутність розуміння семантичного змісту оброблюваного тексту. Виникає необхідність удосконалення алгоритмів для врахування цих факторів [6].

1.3.3 Метод Bag of Words (BoW)

Метод Bag of Words (BoW) є одним із фундаментальних підходів до векторизації текстових даних. Його основна ідея полягає в тому, щоб представити текстовий документ як набір ізольованих слів без урахування граматичних та синтаксичних відношень між ними.

Основні етапи векторизації за методом BoW:

1. Токенізація. Початковий текст розбивається на окремі слова або токени. Цей етап допомагає виокремити ключові одиниці тексту для подальшого аналізу. Можна використовувати будь-який спосіб токенізації вхідних даних (Whitespace Tokenization, Punctuation-based Tokenization, WordTokenizer, etc) [22].

2. Створення словника, який включає всі унікальні слова, що зустрічаються у текстах. Кожне унікальне слово буде мати власний індекс у цьому словнику.

3. Побудова векторів. Кожен документ (або речення) представляється вектором, де кожний елемент відповідає кількості входжень відповідного слова із словника. Значення вектору вказує на кількість входжень кожного слова у відповідному документі. Також можливе використання відносних частот або інших вагових коефіцієнтів.

Метод Bag of Words є ефективним засобом векторизації текстових даних, надаючи можливість представити документи у вигляді числових векторів. Його застосування знаходить широке застосування в сучасних системах обробки тексту та аналізу даних.

1.3.4 Doc2Vec

Метод Doc2Vec виступає як потужний інструмент у контексті векторизації текстових даних для задач обробки природної мови. Цей метод відрізняється від традиційних методів векторизації, таких як Bag of Words, оскільки враховує контекстуальні зв'язки слів у тексті. В основі Doc2Vec лежить ідея збереження не тільки самого слова, але й його контексту, що робить цей метод більш потужним у відтворенні семантичних зв'язків між словами та документами [18].

Архітектура моделі Doc2Vec базується на рекурентних нейронних мережах (RNN) і складається з двох основних моделей: DM (Distributed Memory) та DBOW (Distributed Bag of Words). У DM-моделі кожен документ та його слова

представлені у векторному просторі, в той час як DBOW модель зосереджена тільки на словах.

Основною концепцією є те, що кожен документ отримує свій унікальний вектор, який узагальнює його семантичний зміст. Цей процес відбувається шляхом навчання моделі на великій кількості текстів, враховуючи контекстуальні взаємодії між словами та документами.

Важливою характеристикою Doc2Vec є здатність генерувати вектори для нових документів, які модель не бачила під час навчання [23]. Це робить його ефективним для роботи з реальними текстовими даними, де набір документів постійно змінюється.

Doc2Vec також може бути використаний для вирішення задач класифікації, кластеризації та пошуку схожих документів. Він надає можливість визначати схожість між текстовими елементами в просторі векторів, враховуючи їхню семантику та структурні особливості.

Ключові переваги методу Doc2Vec:

1. Універсальність векторизації та врахування контексту. За допомогою нейронної мережі, модель намагається враховувати контекстуальні взаємодії слів та документів, що дозволяє універсально репрезентувати семантичний зміст, а не обмежуватися векторами окремих слів.

2. Можливість векторизації нових документів. Модель навчається генерувати вектори для нових документів, , які модель не “бачила” під час навчання. Це підвищує ефективність при роботі з реальними сценаріями обробки текстових даних, де набір документів постійно змінюється.

3. Завдяки своїй архітектурі, Doc2Vec може ефективно впоратися з великими обсягами даних.

Недоліки методу Doc2Vec:

1. Витрати обчислювальних ресурсів. Навчання моделі - це ресурсоємне завдання, особливо при великій кількості вхідних даних та великих розмірах векторів.

2. Для ефективного навчання та відтворення семантичних зв'язків потрібна значна кількість текстових даних.

3. Інтерпретаційна складність. Через внутрішню складність нейронної мережі важко пояснити і зрозуміти, як саме модель вивчає семантику. Тому задача інтерпретації і, особливо, пошуку причин і виправлення важливих помилок у результатах роботи стає важким завданням.

Незважаючи на ці недоліки, Doc2Vec залишається потужним інструментом для векторизації текстових даних та здатним до успішного застосування в різних завданнях обробки природної мови.

1.3.5 Метод FastText

Метод FastText представляє собою потужний метод векторизації текстових даних. Однією з ключових особливостей FastText є його спроможність генерувати вектори для слів та фраз, враховуючи морфологічні особливості. Цей метод використовує не тільки самі слова, але і їхні частини, що дозволяє ефективно враховувати семантичні зв'язки в текстах з різною лексичною структурою.

FastText базується на ідеї використання n-грам для подання слів, що дозволяє враховувати буквено-морфемні особливості та забезпечує унікальні вектори для кожної частини слова. Це особливо корисно для мов із багатою морфологією та складною структурою слів. Навчальний процес FastText здійснюється шляхом максимізації ймовірності наявності слів чи n-грам у контекстах, що дозволяє моделі вивчати семантичні зв'язки в корпусі текстів.

FastText є відносно новим методом векторизації тестових даних. Тож дослідження FastText активно продовжуються. Дослідження в галузі FastText продовжуються, і це підтверджується численними науковими роботами. Наприклад у статті “Enriching Word Vectors with Subword Information” [11] розглядаються деталі методу FastText та його застосування для класифікації текстів. А у статті “Bag of Tricks for Efficient Text Classification” [12] подаються

практичні поради з використання моделі. Після аналізу цих робіт можна виділити переваги і недоліки застосування FastText.

FastText володіє рядом переваг, які роблять його значущим інструментом у векторизації текстових даних:

1. Морфологічна чутливість. Завдяки використанню n-грами для кожного слова, FastText дозволяє враховувати морфологічні особливості даних. Це робить його особливо ефективним для мов, у яких одне слово може мати різні форми залежно від контексту, або для мов із різноманітною лексикою. Такий підхід сприяє більш точній векторизації слів та фраз, що важливо для багатьох завдань обробки природної мови

2. Робота з низькочастотними словами. FastText демонструє ефективність у векторизації низькочастотних слів, які можуть виявитися критичними у спеціалізованих або технічних текстах. Модель зберігає інформацію навіть для слів з обмеженим вживанням, що додає моделі гнучкості при роботі з рідкісними термінами і поліпшує результати роботи моделі для таких випадків.

3. Універсальність векторизації. Можливість генерації векторів для слів, які відсутні у навчальному наборі даних, розширює можливості моделі при роботі з новими даними. Це особливо корисно для завдань, де тексти постійно оновлюються.

4. Висока якість векторизації. FastText відомий своєю здатністю забезпечувати високу якість векторизації для різних слів та фраз, навіть у великих колекціях текстів. Це дозволяє ефективно використовувати його в широкому спектрі завдань обробки природної мови.

Незважаючи на багато позитивних аспектів, FastText також має свої обмеження:

1. Високі витрати обчислювальних ресурсів. Процес навчання FastText вимагає значних обчислювальних ресурсів, особливо при великих обсягах даних. Висока обчислювальна складність може бути обмеженням для застосування моделі на обмежених ресурсах.

2. Складність налаштування параметрів. Для досягнення оптимальної ефективності, модель потребує належного налаштування параметрів, таких як розмір векторів та глибина нейронної мережі. Налаштування може бути трудомістким завданням і вимагати експертного досвіду. А також можуть знадобитись додаткові ітерації навчання\перенавчання.

3. Менша інтерпретаційна зрозумілість. Інтерпретація отриманих векторів може бути складною через внутрішню складність моделі та велику кількість параметрів. Розуміння того, як саме вектори відображають семантику, є складним завданням.

4. Залежність від обсягу даних. Ефективність FastText сильно залежить від обсягу та якості навчального набору даних. У випадку обмежених даних це може вплинути на здатність моделі адекватно представляти семантичні зв'язки в текстах.

Зазначені переваги та недоліки варто розглядати в контексті конкретного завдання та обсягу даних, оскільки вони можуть впливати на ефективність та придатність методу FastText для конкретного використання.

1.3.6 Метод Latent Dirichlet Allocation (LDA)

Метод Latent Dirichlet Allocation (LDA) є потужним інструментом у сфері тематичного моделювання текстових даних. Його основна ідея полягає в тому, щоб розглядати кожний документ, як комбінацію різних тем і кожне слово в документі, як приналежне до певної теми. Зазвичай LDA застосовується для виявлення тем у колекції текстів. Проте, його можна використовувати і в контексті задачі дедуплікації веб-сайтів для виявлення схожих чи дубльованих контентів.

В статті "Latent Dirichlet Allocation" [9] розглядаються деталі моделі LDA. Документ розглядається, як мішанина тем, де кожна тема є розподілом ймовірностей по словах, і для кожного слова в документі обирається тема відповідно до цього розподілу. Автори визначають LDA, як генеративну ймовірнісну модель, яка описує, як документи створюються з комбінації тем.

Процес генерації дозволяє отримати як розподіл тем у документах, так і розподіл слів у кожній темі. Інше дослідження цього ж автор [10] висвітлює використання LDA для тематичного моделювання та виділяє його переваги у розумінні структури та змісту текстових колекцій.

LDA використовує математичні моделі, такі як розподіл Діріхле, для опису розподілу тем у документах та слів у темах. Це робить його ефективним інструментом для виявлення та аналізу тем великої кількості даних.

Результатами роботи моделі LDA є набір тем і відповідних ймовірностей входження слів у кожну тему. А кожен документ представлений, як розподіл ймовірностей по темах.

Для вирішення задачі дедуплікації веб-сайтів потрібно визначити схожість вхідних колекцій даних на основі результатів роботи моделі LDA. Можна порівняти розподіли тем для двох документів та визначити схожість між ними [31]. Якщо документи мають схожі розподіли тем, це може свідчити про їхню схожість або дублювання.

Серед недоліків використання LDA можна виділити складність параметризування моделі (кількість тем, кількість ітерацій, кількість слів у темі та ін). Для вирішення задачі дедуплікації невірно виставлені параметри сильно впливають на результат. Частково невірний результат на етапі роботи моделі ще більш буде помітно після порівняння кількох результатів та прийняття рішення на основі цього порівняння.

1.3.7 Jaccard Similarity

Jaccard Similarity дозволяє оцінити схожість вхідних даних і базується на вимірюванні схожості між двома множинами, використовуючи кількість їхніх спільних елементів відносно загальної кількості унікальних елементів. Цей метод знаходить широке застосування у веб-майнінгу, аналізі текстів, та інших областях, де важливо вимірювати ступінь схожості між двома наборами даних.

Важливою особливістю роботи цього алгоритму є представлення вхідних даних у вигляді множин. Для роботи з текстовими даними кожен документ може бути представлено як множину унікальних слів, чи окремих термінів, тощо.

У роботі “Similarity Measures for Text Document Clustering”[7] висвітлено математичні аспекти методу Jaccard Similarity та його застосування для аналізу текстових даних.

Jaccard Similarity (J) обчислюється за допомогою наступної формули (1.1):

$$J(A,\ B) = \frac{|A \cap B|}{|A \cup B|}, \tag{1.1}$$

де $|A \cap B|$ - кількість спільних елементів між множинами A і B, а $|A \cup B|$ - загальна кількість унікальних елементів у обох множинах.

Значення Jaccard Similarity може приймати значення в діапазоні від 0 до 1, де 0 вказує на повну відсутність спільних елементів, а 1 - на ідентичність множин.

Вивчаючи метод Jaccard Similarity була досліджена робота “Similarity Measures for Text Document Clustering” [8]. У цій роботі автор аналізує застосування різних мір схожості для кластеризації текстових документів, включаючи Jaccard Similarity. Jaccard Similarity розглядається, як один із ключових показників для порівняння текстових документів у визначенні їхньої схожості. Цей метод особливо корисний в задачах, де важлива не тільки наявність спільних слів, але й їхнє відношення до унікальних слів у кожному документі.

Такий підхід знаходить своє застосування в областях, де важливо враховувати контекстуальну схожість між документами. Наприклад, виявлення тематично схожих статей чи категоризація текстових документів за схожістю їхнього змісту.

У той же час Jaccard Similarity має певні недоліки:

1. Міра схожості не враховує кількість входжень конкретного елемента у вхідних множинах. Два набори можуть мати велику кількість спільних елементів, але сильно відрізнятися за кількістю повторень кожного з них.

2. Не враховується порядок елементів у множині. Це означає, що однакова схожість буде призначена двом множинам, незалежно від порядку їхніх елементів.

3. Враховує тільки кількість спільних елементів і загальну кількість унікальних елементів. Великі множини можуть мати меншу схожість за умови, що кількість спільних елементів не є великою відносно загальної кількості унікальних елементів. Тож якщо у множинах багато не значущої інформації, це може вплинути на результат

1.3.8 Cosine Similarity

Cosine Similarity є однією з ключових технік вимірювання схожості текстових даних. Цей підхід є ефективним і широко використовується у задачах порівняння та кластеризації текстової інформації [14].

Косинусна схожість визначає ступінь відповідності між двома векторами у просторі високої вимірності. В текстовому контексті кожен документ або текст представляється, як вектор, де кожна координата відповідає певному слову або терміну. Косинусна схожість визначає кут між цими векторами від 0 до 180 градусів [15].

Косинусна схожість обчислюється за формулою (1.2):

$$S_C(A, B) = \frac{A \times B}{||A|| \times ||B||}, \quad (1.2)$$

де $A \times B$ - скалярний добуток векторів A та B, $||A||$ - норма вектора A, $||B||$ - норма вектора B/

За результатами проведених досліджень [1], косинусна схожість виявляється надзвичайно корисною в різних сценаріях обробки текстової інформації. Зокрема, вона демонструє високу ефективність у визначенні ступеня подібності між текстовими документами, дозволяючи вирішувати складні завдання порівняння та кластеризації.

Основні переваги використання методу косинусної схожості:

- простота та швидкість (з точки зори обчислювальних ресурсів);
- хороший результат при роботі з розрідженими векторами;
- легкість інтерпретації результату;
- ефективність на великих обсягах даних.

Одним з головних недоліків є неефективність методу у роботі з короткими текстами, або поодинокими словами. Через обмежену інформативність векторів косинусна схожість втрачає інформативність.

Не зважаючи на ці недоліки, косинусна схожість залишається ефективним інструментом в багатьох застосуваннях, особливо там, де важливо визначення ступеня схожості між об'єктами.

2 РОЗРОБКА МЕТОДИКИ ДЕДУПЛІКАЦІЇ ВЕБ-САЙТІВ

2.1 Етапи вирішення задачі дедуплікації сайтів

Однією з основних особливостей задачі дедуплікації сайтів є масштабність та розмаїття даних, які доводиться обробляти. Велика кількість веб-сайтів, різноманітні структури сторінок та постійне змінювання контенту ускладнюють виявлення та обробку дублікатів.

Пошук та виявлення дублікатів стає ще складнішим завданням через різноманітність представлених даних. Врахування цієї різноманітності вимагає вдосконалення методів обробки та аналізу даних.

З огляду на об’єм необробленої інформації, порівняння весього контенту веб-сайту буде неефективне та може виникнути проблема великої обчислювальної складності, оскільки весь текст містить велику кількість слів, багато з яких можуть бути несуттєвими для виявлення схожості. Визначення ключових слів дозволяє зосередитися на основних елементах, що полегшує процес дедуплікації та покращує його точність.

Окрім того, для успішного розрахунку коефіцієнта подібності сайтів ключові слова краще представити у вигляді векторів. Векторизація дозволяє перетворити текстові дані у простір чисел, що полегшує порівняння та визначення схожості між сайтами. Також варто відзначити що методів порівняння числової інформації значно більше ніж текстової. А також ці методи є більш ефективними з точки зору використання обчислювальних ресурсів.

Після вивчення інформації про існуючі підходи до обробки великих об’ємів інформації з веб - сайтів, а також алгоритмів аналізу цієї інформації, можна виділити наступні етапи роботи системи для дедуплікації сайтів (рис. 2.1):

1. Завантаження головної сторінки веб сайту.
2. Фільтрація посилань, присутніх на головній сторінці сайту та обрання обмеженої кількості найінформативніших сторінок.
3. Скачування повного набору сторінок для кожного веб сайту.

4. Очищення сторінок від HTML розмітки та підготовка необроблених даних для подальшого аналізу.
5. Отримання списку ключових слів для кожного сайту.
6. Векторизація отриманих ключових слів.
7. Розрахування коефіцієнту подібності.

Рисунок 2.1 – Етапи вирішення задачі дедуплікації сайтів

2.2 Загальна архітектура системи

Важливою складовою майбутньої системи дедуплікації є оптимізація використання ресурсів та можливість швидкого та легкого масштабування системи. Оптимізація ресурсів дозволяє ефективно використовувати обчислювальну потужність та витрачати менше коштів на підтримку необхідної інфраструктури. А масштабованість системи дозволяє зменшити час виконання завдання та зняти обмеження на об’єм даних, які потрібно обробити.

У програмному забезпеченні види навантаження можна розділити на I/O bound та CPU bound операції. Для I/O bound операцій характерне те, що швидкість виконання задачі обмежено швидкістю операцій введення/виведення. Наприклад великою кількістю мережевих взаємодій. Для CPU bound операцій характерне те, що швидкість виконання задачі обмежено обчислювальною потужністю процесора. Ця проблема зазвичай є у задачах, в яких основна робота - це математичні обчислення.

Змішані навантаження, що поєднують як I/O, так і CPU bound операції, можуть призводити до складних викликів у розробці програмного забезпечення. Проблеми можуть виникати внаслідок неправильного балансу між різними видами операцій, що призводить до непродуктивного використання ресурсів.

Мікросервісна архітектура є оптимальним рішенням для вирішення проблем змішаних навантажень. Ця архітектурна парадигма дозволяє розділити систему на невеликі, незалежні мікросервіси, які можуть ефективно масштабуватися та оптимізувати використання ресурсів. Кожен мікросервіс може бути спрямований на вирішення конкретної задачі та взаємодіяти з іншими мікросервісами, оптимізуючи таким чином загальне використання ресурсів системи. Додатковою перевагою мікросервісної архітектури є легка масштабованість системи.

У контексті розробки системи для дедуплікації сайтів, на основі описаних етапів роботи, можна виділити наступні мікросервіси:

- entrypoint;
- pipeline controller;
- collector;
- text processor;
- keyword processor;
- vectorization processor;
- decision maker.

Схему взаємодії між сервісами зображено на рисунку 2.2.

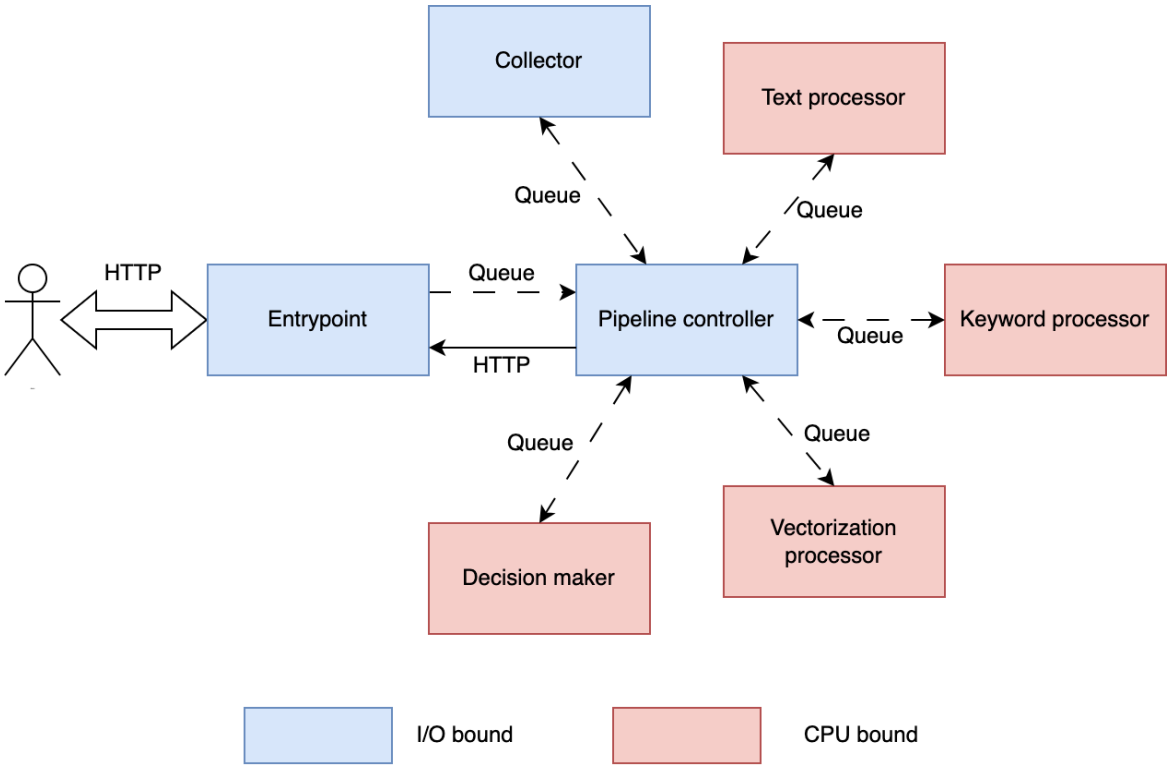


Рисунок 2.2 – Схема взаємодії мікросервісів системи

Кожен із описаних мікросервісів буде виконувати свою частину загальної задачі. Це дозволяє зробити систему максимально гнучкою і масштабованою. Також у майбутньому це дозволить за потреби міняти підхід до обробки даних на кожному окремому етапі не змінюючи інших етапів обробки, адже мікросервіси будуть максимально (наскільки це можливо) незалежними один від одного.

При наявності такої кількості незалежних сервісів у розподіленій системі важливим питанням є взаємодія між цими сервісами. У взаємодії між мікросервісами важливо розрізняти синхронні та асинхронні методи комунікації.

Синхронна взаємодія передбачає, що клієнтський сервіс чекає на відповідь від сервера перед продовженням роботи. Такий підхід характеризується тим, що запит та відповідь синхронізовані в часі. Важливо враховувати, що синхронна взаємодія може призводити до блокування ресурсів, особливо в великих розподілених системах.

З іншого боку, асинхронна взаємодія дозволяє клієнтському сервісу надсилати запити та не чекати миттєву відповідь. Це відкриває можливості для

продовження виконання інших завдань, підвищує швидкодію системи та реагує на зміни в навантаженні. Однак асинхронність може ускладнювати відлагодження коду та вимагати додаткових зусиль для контролю виконання задач.

У системі, що розробляється у цьому дипломному проекті буде використовуватись гібридна модель комунікації:

1. Синхронна модель з використанням HTTP протоколу для завдань в яких потрібна миттєва відповідь. Наприклад запуск роботи системи та отримання фінальних результатів користувачем буде відбуватись саме таким чином. Також HTTP буде використовуватись у випадках, коли потрібно зберегти інформацію про поточний прогрес виконання задачі.

2. Асинхронна модель з використанням черги повідомлень для обробки та переходу між внутрішніми етапами обробки даних. У якості черги повідомлень буде використовуватись RabbitMQ.

Також слід зазначити, що кожен сервіс повинен бути упакований у Docker-контейнер. Це спрощує запуск всієї системи незалежно від середовища. Єдина вимога до середовища - можливість запускати Docker-контейнери.

2.3 Сервіс entrypoint

Сервіс entrypoint є єдиним способом взаємодії користувача із системою. Сервіс буде реалізовано у вигляді API з використанням фреймворку FastAPI для мови програмування Python. Сервіс має відповідати наступним вимогам:

1. API повинно відповідати принципам REST. Це дозволить легко взаємодіяти із системою, за потреби інтегрувати систему дедуплікації доменів у інші системи, бо REST де-факто є стандартом при розробці сучасних API інтерфейсів.

2. Надавати вичерпну документацію про методи, доступні у API системи.

3. Валідувати вхідні дані, а також повертати користувачу інформативні повідомлення про помилки, якщо такі будуть.

4. Надавати користувачу системи інтерфейс для ініціювання обробки списку веб-сайтів і інтерфейс для отримання результатів роботи.

5. Надавати внутрішнім сервісам системи інтерфейси для збереження проміжних результатів роботи.

6. Зберігати дані у реляційній базі даних

На основі цих вимог розроблено вимоги до інтерфейсів, які мають бути у API сервісу entrypoint.

1. Ендпоінт POST /task. Цей ендпоінт буде дозволяти користувачу створювати нове завдання для системи та мати наступний формат вхідних даних (формат JSON):

- sites: список string, кожен елемент списку має бути валідним посиланням;
- use_archive_data: bool.

Формат валідної відповіді у форматі JSON: об’єкт із ключем “id” та значенням - ідентифікатором задачі у форматі UUID.

2. Ендпоінт GET /task/{task_id}?similarity_border={float}. Цей ендпоінт буде дозволяти користувачу отримати результати роботи системи для створеного завдання. Формат вхідних даних (вказується у посиланні методу):

- task_id: ідентифікатор завдання, string у форматі UUID;
- similarity_border: float

Формат відповіді у форматі JSON:

- status: ENUM (“in_progress”, “success”);
- created_at: дата та час створення завдання, string у форматі DATETIME;
- done_at (optional): дата та час завершення завдання, string у форматі DATETIME;
- results (optional): масив об’єктів, у якому ключ - це порівняні сайти, а значення - наскільки вони схожі.

3. Ендпоінт GET /private/task/{task_id}. За допомогою цього методу можна буде отримати інформацію про завдання за його ID. Цей метод повинен бути

захищено авторізацією, тому що він призначений лише для внутрішнього використовування. Вхідним параметром є ідентификатор завдання (string у форматі UUID)

Формат відповіді у форматі JSON:

- id: ідентифікатор завдання, string у форматі UUID;
- created_at: дата та час створення завдання, string у форматі DATETIME;
- done_at: дата та час завершення завдання, string у форматі DATETIME;
- sites: список сайтів для обробки, масив string.

4. Ендпоінт PATCH /private/task/{task_id}/done. За допомогою цього методу можна буде оновити інформацію про прогрес обробки конкретного завдання. Цей метод повинен бути захищено авторізацією, тому що він призначений лише для внутрішнього використовування. Вхідним параметром є ідентификатор завдання (string у форматі UUID).

5. Ендпоінт GET /private/site/{site_id}. За допомогою цього методу можна буде отримати інформацію про результати обробки конкретного сайту. Цей метод повинен бути захищено авторізацією, тому що він призначений лише для внутрішнього використовування. Вхідним параметром є ідентификатор сайту (string у форматі UUID).

Формат відповіді у форматі JSON:

- id: ідентифікатор сайту, string у форматі UUID;
- task_id: ідентифікатор завдання, string у форматі UUID;
- root_url: посилання на ініціюючу сторінку сайту, string;
- files_path: назва каталогу файлової системи, у якому зберігаються результати обробки, string;
- downloaded_at: дата та час завершення етапу завантаження сторінок, string у форматі DATETIME;
- preprocessed_at: дата та час завершення етапу обробки тексту, string у форматі DATETIME;

- extracted_at: дата та час завершення етапу визначення ключових слів, string у форматі DATETIME;
- vectorized_at: дата та час завершення етапу векторизації, string у форматі DATETIME;

6. Ендпоінт PUT /private/site/{site_id}. За допомогою цього методу можна буде оновити інформацію про прогрес обробки конкретного веб-сайту. Цей метод повинен бути захищено авторізацією, тому що він призначений лише для внутрішнього використовування.

Формат вхідних даних:

- site_id: ідентифікатор сайту, string у форматі UUID;
- task_id: ідентифікатор завдання, string у форматі UUID;
- root_url: посилання на ініціюючу сторінку сайту, string;
- files_path: назва каталогу файлової системи, у якому зберігаються результати обробки, string;
- downloaded_at: дата та час завершення етапу завантаження сторінок, string у форматі DATETIME;
- preprocessed_at: дата та час завершення етапу обробки тексту, string у форматі DATETIME;
- extracted_at: дата та час завершення етапу визначення ключових слів, string у форматі DATETIME;
- vectorized_at: дата та час завершення етапу векторизації, string у форматі DATETIME.

5. Ендпоінт PUT /private/compare_result/{site1_id}/{site2_id}. За допомогою цього методу можна буде оновити інформацію про прогрес обробки конкретного веб-сайту. Цей метод повинен бути захищено авторізацією, тому що він призначений лише для внутрішнього використовування.

Формат вхідних даних:

- site_id: ідентифікатор сайту, string у форматі UUID;
- task_id: ідентифікатор завдання, string у форматі UUID;
- similarity: коефіцієнт подібності сайтів, float.

Згідно з вимогами до сервісу, результати роботи повинні зберігатися у реляційній БД. Для цього було обрано СУБД PostgreSQL. Система управління базами даних PostgreSQL є однією з найпопулярніших та найпотужніших відкритих СУБД. Її архітектура базується на принципах реляційної моделі даних, що дозволяє ефективно керувати великими обсягами інформації та забезпечує високий рівень надійності та стабільності. СУБД також володіє високою мірою розширюваності, дозволяючи впровадження реплікації та кластеризації для забезпечення надійності та швидкодії.

На основі вимог розроблено схему бази даних, яка призначена для зберігання результатів роботи системи зображено на рисунку 2.3.

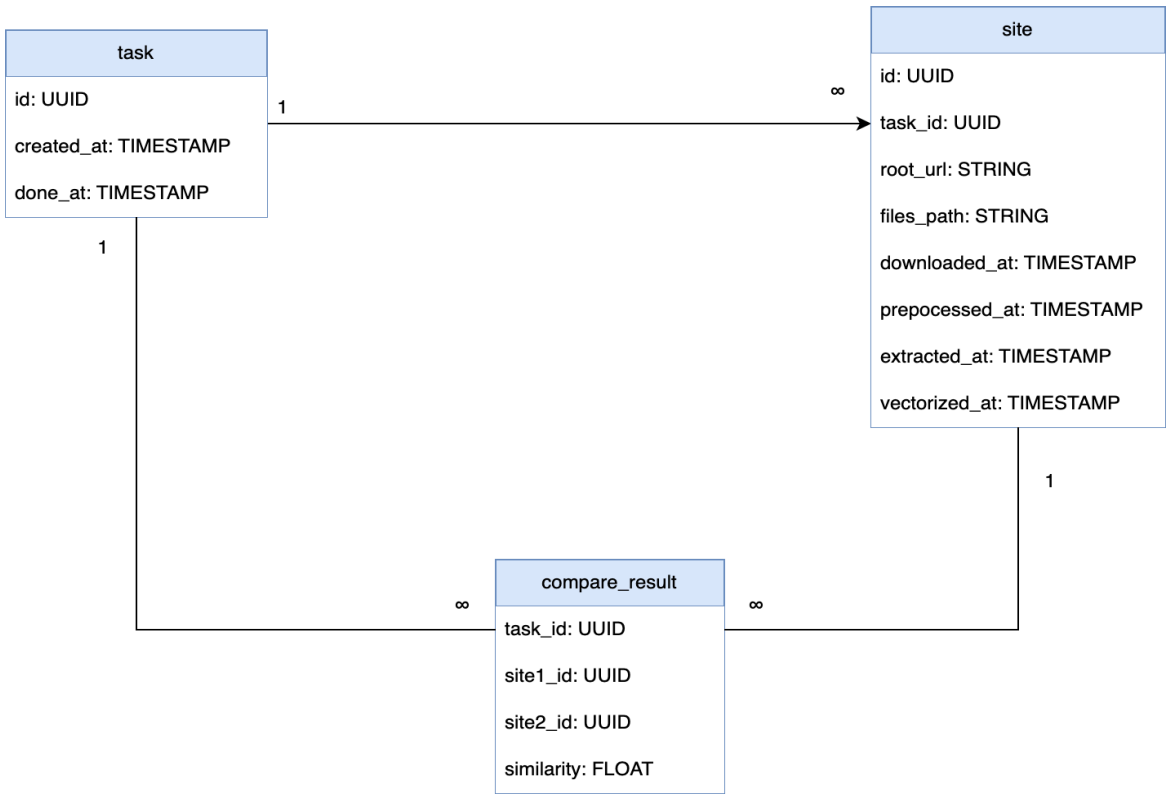


Рисунок 2.3 – Схема бази даних системи

Слід також зазначити, що всі взаємодії із БД будуть виконуватись за допомогою ORM SqlAlchemy. ORM SqlAlchemy - це сучасний інструмент для взаємодії між програмами на мові програмування Python та реляційними базами

даних. SqlAlchemy є абстракцією на операціями з БД. Найвагоміша перевага використання цієї абстракції - заміна реляційної (якщо це знадобиться) СУБД на іншу без необхідності зміни коди самої системи. У більшості випадків достатньо лише змінити налаштування SqlAlchemy при запуску.

2.4 Сервіс pipeline controller

Сервіс pipeline controller буде відповідати за управління етапами обробки даних та зберігати проміжні результати обробки. Для розробки каркасу сервісу буде використовуватись фреймворк FastStream.

Сервіс має відповідати наступним вимогам:

1. Валідація вхідних даних. У випадку невалідних даних результатом роботи сервісу має бути інформативне повідомлення про помилку.
2. Сервіс повинен обробляти всі можливі результати роботи інших сервісів і приймати рішення про подальшу обробку на основі цих результатів.
3. Сервіс повинен напряму взаємодіяти із внутрішніми ендпоінтами сервісу entrypoint. Це дозволить зберегти проміжні результати роботи всієї системи.

Формат вхідного повідомлення:

- settings: об’єкт формату:
  - task_id: ідентифікатор задачі, string у форматі UUID
  - site_id (optional): ідентифікатор сайту, string у форматі UUID
- data: об’єкт змінного формати (формат залежить від поточного кроку обробки).

Схема переходів між станами обробки задачі зображено на рисунку 2.4.

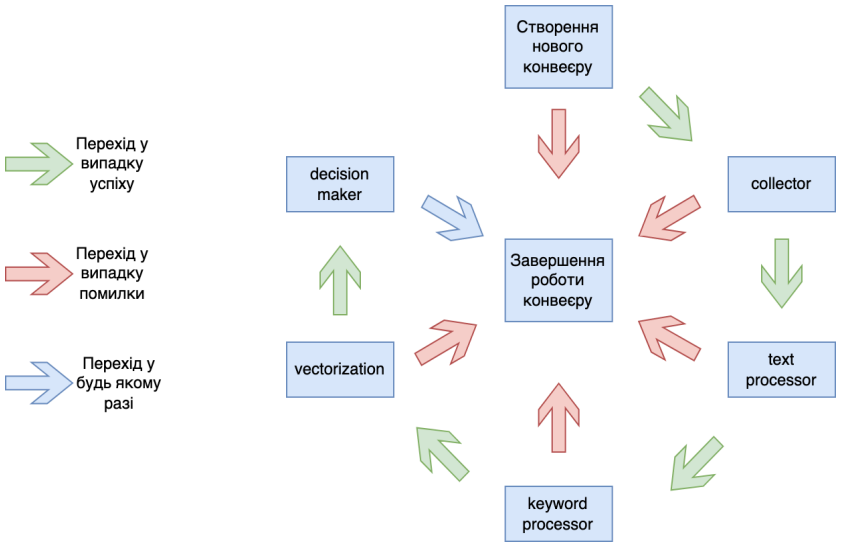


Рисунок 2.4 – Перехід між станами для задачі

Загальна схема роботи сервісу зображена на рисунку 2.5.

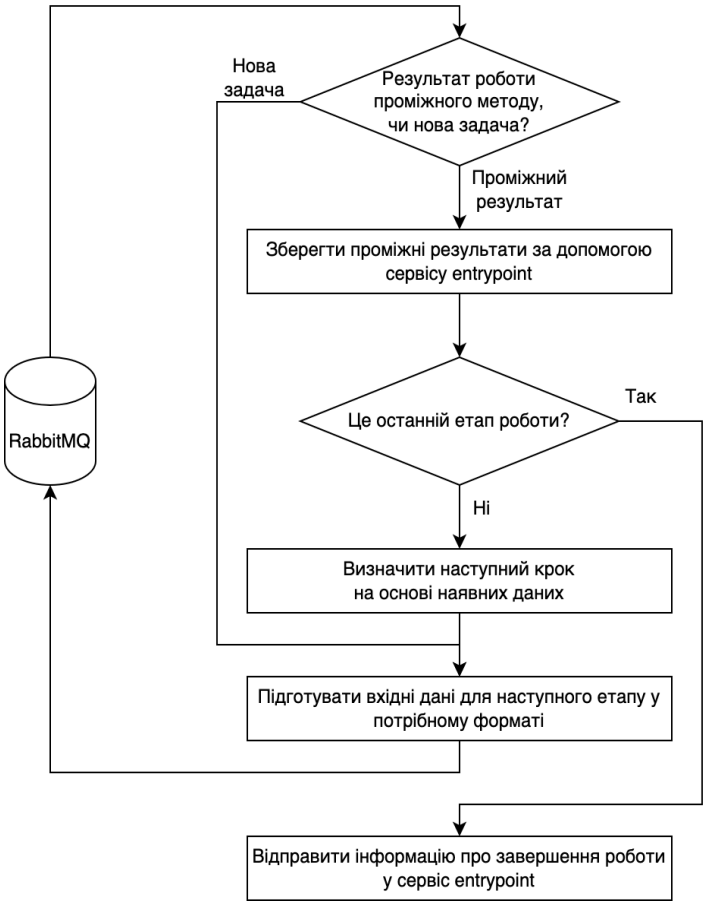


Рисунок 2.5 – Схема роботи сервісу pipeline controller

2.5 Сервіс collector

Сервіс collector буде відповідати за скачування з мережі Internet сторінок веб-сайтів. Результатом роботи сервісу є файли із завантаженим контентом сторінок і файл із мета інформацією про завантаження. Сервіс буде працювати з двома основними технологіями - FastStream та aiohttp.

FastStream - це фреймворк, який дозволяє створювати застосунки, які працюють за схемою “взяте повідомлення з черги -> обробити повідомлення -> створити повідомлення у іншій черзі з результатами роботи”. FastStream дозволяє швидко створювати застосунки не витрачаючи часу на обробку комунікацій між застосунком та брокером повідомлень. Також FastStream є абстракцією над брокером повідомлень. Тож у разі потреби заміни брокера повідомлень не потрібно переписувати внутрішню логіку роботи застосунку.

Вивчивши переваги та недоліки інструментів для скачування даних з мережі інтернет, мною було обрано бібліотеку aiohttp. Серед її переваг оптимізована робота з мережею і відсутність вбудованого менеджменту процесу завантаження даних. Це з однієї сторони додає роботи, але з іншої - дозволяє зробити систему оптимізованою саме під вимоги цього сервісу не перевантажуючи систему непотрібним функціоналом.

Сервіс має відповідати наступним вимогам:

1. Валідація вхідних даних. У випадку невалідних даних результатом роботи сервісу має бути інформативне повідомлення про помилку.
2. Сервіс повинен бути асинхронним і має обробляти кілька повідомлень з черги одночасно незалежно одне від одного.
3. Сервіс повинен скачувати певну кількість сторінок. Скачуватись повинні лише найважливіші сторінки.
4. Сервіс повинен зберігати завантажену інформацію на диску.

Загальна схема роботи сервісу зображена на рисунку 2.6.

Рисунок 2.6 – Схема роботи сервісу collector

Згідно з вимогою №3 система повинна скачати лише найважливіші сторінки з веб-сайту. То ж система ранжування посилань з веб-сторінок є критичним етапом для подальшої обробки і вирішення завдання дедуплікації. Ранжування сторінок повинно відбуватись за їх істотністю та релевантністю. Але єдина інформація про сторінки, що буде у нас на цьому етапі - це саме посилання, та текст, який відображається для цього посилання на початковій сторінці.

Після аналізу наявної інформації я зробив висновок, що найважливіші сторінки для можливості дедуплікації - це сторінки з інформацією про компанію. Зазвичай на ці сторінці більше за все унікальної інформації про те, чим займається компанія - власник веб - сайту.

А сторінки з юридичною інформацією навпаки сильно погіршують результати. Справа в тому, що сторінки з юридичною інформацію (на кшталт “policy”, “rules”, “terms”, та ін) містять дуже великий об’єм інформації. І ця інформація, зазвичай, дуже схожа від сайту до сайту. Але ця інформація сама по собі не дає ніякого уявлення про те, чим займається компанія. Тож подібна інформація лише додає “шуму” на подальших етапах. А це негативно відображається на результатах оцінки.

Тож для вирішення проблеми ранжування було проаналізовано відкриті дані про посилання, заголовки і типи сторінок. На основі цього аналізу побудовано 2 словники:

- слова, які ідентифікують сторінки з інформацією про компанію;
- слова, які ідентификують сторінки з юридичною інформацією

Перед зберіганням словників слова в них були оброблено алгоритмами стемінгу і токенізації. На основі цих даних і буде проходити ранжування посиланнь. Механізм ранжування зображено на рисунку 2.7.

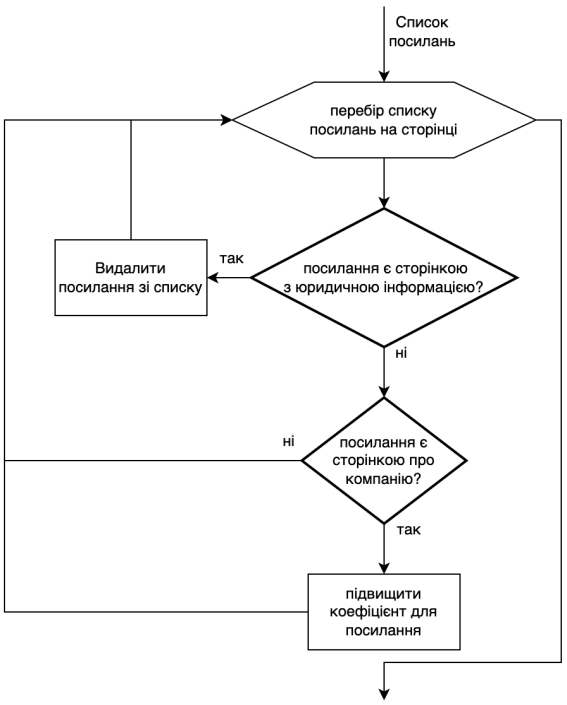


Рисунок 2.7 – Механізм ранжування посилань на внутрішні сторінки веб-сайту

Завантажені сторінки будуть зберігатись у файли. Сервіс повинен мати можливість зберігати файли як на жорсткий диск, так і у хмарні аналоги (буде реалізовано зберігання у Google Cloud Storage). Локальний диск зручно використовувати під час розробки і тестування, але для роботи масштабованої системи у реальних умовах більше підходять спеціально розроблені для цього хмарні аналоги.

2.6 Сервіс text processor

Сервіс text processor буде відповідати за обробку скачаних сторінок та екстракцію текстової інформації з цих сторінок. Результатом роботи сервісу є файли з очищеним, значущим текстом, поділеним на речення. Для розробки каркасу сервісу буде використовуватись фреймворк FastStream.

Сервіс має відповідати наступним вимогам:

1. Валідація вхідних даних. У випадку невалідних даних результатом роботи сервісу має бути інформативне повідомлення про помилку.
2. Сервіс повинен отримувати інформацію лише з блоків, які доступні реальним користувачам сайту. Іншими словами, якщо блок містить текстову інформацію, але його відображення на сторінці приховано, то це текст не повинен потрапити у результати роботи.
3. Сервіс повинен визначати мову, яка використовується на сторінці. Якщо на сторінках сайту використовується кілька мов, то потрібно визначити основну і працювати з текстом цією мовою
4. Сервіс повинен зберігати завантажену інформацію на диску.

Загальна схема роботи сервісу text processor зображена на рисунку 2.8.

Згідно з вимогою №2 сервіс повинен працювати лише з інформацією, яку бачить реальний користувач. Зробити блок з даними невидимим можна двома шляхами: за допомогою спеціальних HTML атрибутів безпосередньо всередині HTML розмітки сторінки та використовуючи CSS. Для того, щоб визначити невидимі елементи та прибрати.

Рисунок 2.8 – Схема роботи сервісу text processor

Першим етапом для задачі визначення невидимих блоків є розбір файлів з CSS стилями. Це дозволить отримати список CSS селекторів, для яких у файлах стилів сайту відключено відображення на сторінці.

Другим етапом є видалення певних тегів із коду HTML сторінки. Для пошуку у HTML розмітці блоків заздалегідь відомого формату, використання регулярних виразів є стандартним та потужним підходом. Для мови програмування Python стандартним засобом роботи з регулярними виразами є вбудована бібліотека “re”. Однак, у випадку великих обсягів даних (якими є HTML сторінки), традиційні методи можуть виявитися неефективними.

В цьому контексті, бібліотека Hyperscan для Python визначається як потужний інструмент, який надає альтернативний підхід до пошуку у тексті. Однією з основних переваг використання Hyperscan у порівнянні із звичайними регулярними виразами є його здатність працювати із великою кількістю шаблонів одночасно, що полегшує обробку широкого спектру вхідних даних.

Кожне завдання, яке вирішується за допомогою регулярних виразів є унікальним. Тож щоб оптимальний обрати підхід треба провести тестування вирішення задачі у реальних умовах.

Для порівняння швидкості роботи бібліотек “re” та “hyperscan” використовувалася задача із такими вимогами:

- вхідні дані - 40 завантажених сторінок одного сайту
- за допомогою регулярних виразів потрібно знайти блоки, для яких за допомогою HTML атрибутів відключенно відображення інформації користувачу
- оцінку часу виконання потрібно проводити без урахування підготовчих етапів, як то завантаження контенту файлів, компіляція регулярних виразів у шаблони, тощо.

Для порівняння було обрано бібліотеку pytest та плагін для цієї бібліотеки pytest-benchmark. Результати порівняння зображено на рисунку 2.9.

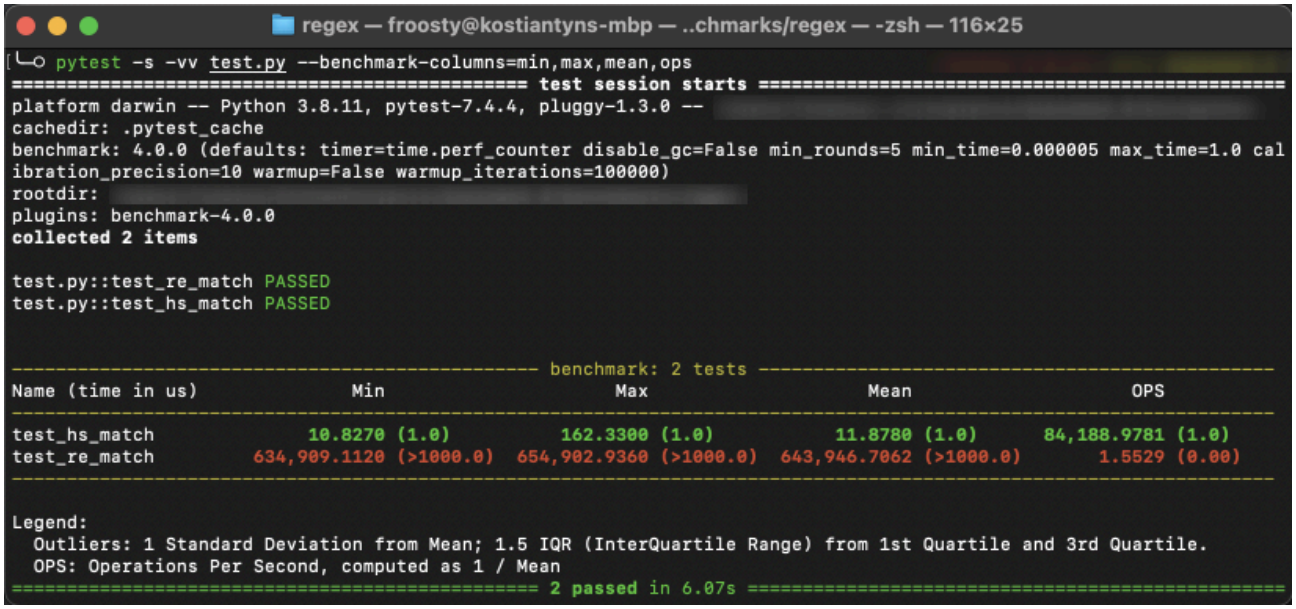


Рисунок 2.9 – Порівняння швидкості роботи бібліотек “re” та “hyperscan”

З результатів тестування видно, що традиційні методи роботи з регулярними виразама у Python є неефективними у випадку великого об’єму вхідних даних та

пошуку даних за кількома регулярними виразами. Бібліотека “hyperscan” працює більш ніж у 1000 разів швидше у умовах вирішення тестового завдання.

Щодо визначення мов, які використовуються на сторінках сайтів, то було вирішено використовувати бібліотеку Compact Language Detector 2 (CLD2). (CLD2) – це бібліотека, розроблена Google, призначена для визначення мови текстових фрагментів. Вона використовується для автоматичного визначення мови веб-сторінок, документів чи текстового контенту у великому обсязі даних [17]. Бібліотека CLD2 базується на алгоритмі, який використовує статистичні методи та нейромережеві техніки для аналізу частоти вживання слів та символів у тексті. Найбільшою перевагою роботи цієї бібліотеки є швидкість обробки та невибагливість до обчислювальних ресурсів.

2.7 Сервіс keyword processor

Сервіс keyword processor буде відповідати за обробку текстової інформації, отриманої із завантажених сторінок веб-сайтів та екстракцію ключових слів із цієї текстової інформації. Результатом роботи сервісу є json-файл зі списком ключових слів для сайту. Для розробки каркасу сервісу буде використовуватись фреймворк FastStream.

Сервіс має відповідати наступним вимогам:

1. Валідація вхідних даних. У випадку невалідних даних результатом роботи сервісу має бути інформативне повідомлення про помилку.
2. Метод аналізу тексту повинен враховувати синтаксичні та морфологічні особливості оброблюваних даних.
3. Сервіс повинен зберігати завантажену інформацію на диску.

Загальна схема роботи сервісу keyword processor зображена на рисунку 2.10.

Рисунок 2.10 – Схема роботи сервісу keyword processor

Для попередньої обробки тексту буде використовуватись програмний комплекс spaCy. Це бібліотека для обробки природної мови (NLP), яка надає інструменти для виконання різноманітних завдань у сфері обробки тексту. Розроблена як ефективний та високопродуктивний інструмент, spaCy стала популярною серед дослідників та розробників завдяки своїй швидкості та функціональності.

Ключові особливості spaCy:

1. Токенізація. SpaCy забезпечує потужний механізм токенізації, який дозволяє розділяти текст на окремі слова або токени. Це допомагає визначити кордони слів, що є важливим етапом для екстракції ключових слів.

2. Лематизація. SpaCy надає можливість лематизації - зведення слова до його базової форми. Це полегшує обробку слова в його загальній формі, що сприяє правильному аналізу тексту та підвищує точність визначення ключових слів.

3. Частотний аналіз. За допомогою spaCy можна визначити частоту вживання слів у тексті, що дає можливість відібрати слова, які зустрічаються найчастіше, як потенційні ключові терміни.

4. Визначення частин мови. SpaCy дозволяє визначити частини мови для кожного слова, що може бути корисним для відбору термінів певного типу, таких як іменники чи дієслова.

Основні принципи роботи spaCy зображено на рисунку 2.11 (рисунок з офіційного сайту бібліотеки https://spacy.io/).

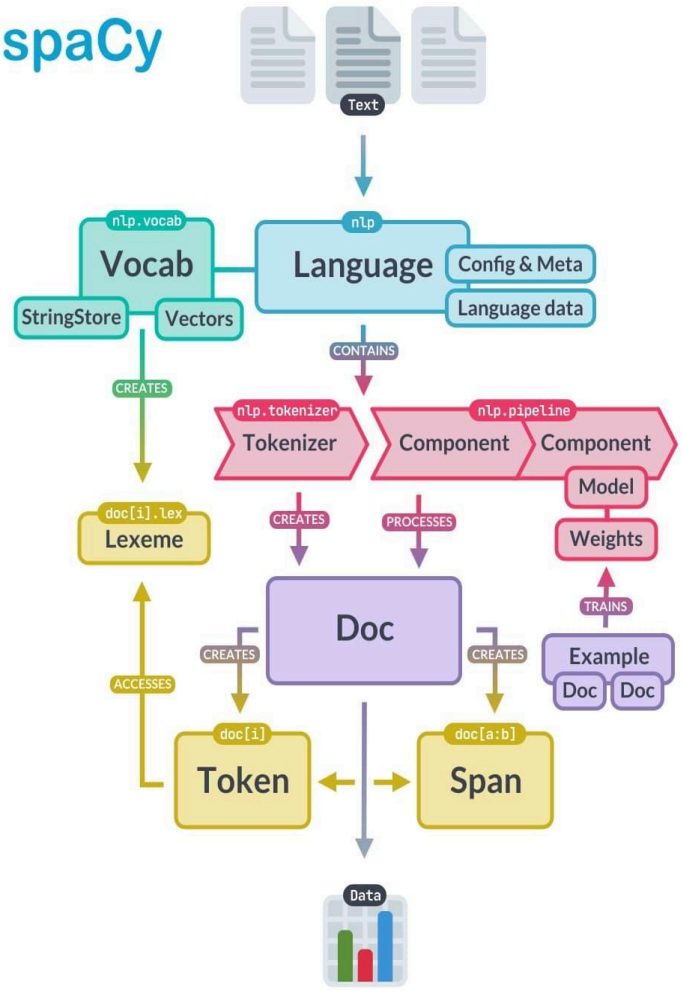


Рисунок 2.11 – Принципи роботи бібліотеки spaCy [16]

Загалом, результатом роботи spaCy є структурований об'єктний граф, який можна легко обробляти та аналізувати з програмного коду. Це дозволяє

розробникам та дослідникам ефективно взаємодіяти з текстовою інформацією та використовувати різноманітні функціональності бібліотеки для різних завдань обробки природної мови.

2.8 Сервіс vectorization processor

Сервіс vectorization processor буде відповідати за обробку ключових слів зі сторінок веб - сайту та перетворення цієї інформації з текстового формату у векторний. Результатом роботи сервісу є json-файл векторами, які характеризують ключові слова для певного веб - сайту. Для розробки каркасу сервісу буде використовуватись фреймворк FastStream.

Сервіс має відповідати наступним вимогам:

1. Валідація вхідних даних. У випадку невалідних даних результатом роботи сервісу має бути інформативне повідомлення про помилку.
2. Сервіс повинен зберігати завантажену інформацію на диску.

Загальна схема роботи сервісу зображена на рисунку 2.12.

Рисунок 2.12 – Схема роботи сервісу vectorization processor

На основі аналізу існуючих підходів до векторизації було обрано метод TF-IDF. Однією з ключових переваг tf-idf є те, що він враховує, як частоту термінів у конкретному документі, так і інверсійну частоту термінів в корпусі текстів. Це дозволяє відзначити не тільки важливі слова, які часто зустрічаються в конкретному документі, але і ті, які є унікальними для цього документа в порівнянні із загальним корпусом.

Такий підхід дозволяє забезпечити точні та інформативні вектори ключових слів, що робить tf-idf одним із найкращих виборів для подальшого порівняння і аналізу текстової інформації.

Показник Term Frequency вказує, наскільки часто конкретний термін з'являється у документі. Розраховується за формулою (2.1):

$$TF(t,\ d)\ =\ \frac{n_t}{\Sigma_k n_k}\ , \tag{2.1}$$

де n_t - кількість входжень слова *t* у документ, а $\Sigma_k\ n_k$ - загальна кількість слів у документі *t*.

Показник Inverse Document Frequency оцінює, наскільки рідкісним є термін у контексті усіх зібранні документів. Розраховується за формулою (2.2):

$$IDF(t,\ D)\ =\ log\frac{|D|}{\left|\{d_i \in D \mid t \in d_i\}\right|}\ , \tag{2.2}$$

де $|D|$ - загальна кількість документів, $\left|\{d_i\ \in\ D \mid t \in d_i\}\right|$ - кількість документів, у яких зустрічається *t*.

Після цього потрібно обчислити показник TF-IDF. Це добуток показників TF та IDF. Розраховується за формулою (2.3):

$$TF\ -\ IDF(t,\ d,\ D)\ =\ TF(t,\ d) \times IDF(t,\ D). \tag{2.3}$$

Для коректного результату роботи методу TF-IDF у розподіленій системі потрібно побудувати словник термінів. Словник повинен бути побудований на великому об’ємі “навчальних даних”. У подальшому це дозволить отримувати коректні значення IDF. “Навчальний датасет” буде зібрано наступним чином:

1. Як вхідні дані було використано лист веб-сайтів. Ці веб-сайти не мають прив'язки до якоїсь конкретної категорії. У цьому листі 200000 сайтів.
2. За допомогою сервісу collector були завантажені сторінки цих сайтів. Загальна кількість завантажених сторінок.
3. За допомогою сервісу text processor було виділено текстові дані зі скачаних сторінок.
4. За допомогою сервісу keyword processor було отримано список ключових слів.
5. Отриманий список пройшов фільтрацію. Із навчального датасету було виключено слова, які зустрічаються менш ніж на 300 сайтах (менш ніж 0.02% від загальної кількості сайтів). Таким чином було відкинуто велику кількість нерепрезентативних даних.

На основі зібраного датасету було навчено TfidfVectorizer із пакету “sklearn”. У результаті було побудовано словник із 48232 ключових слів, який у подальшому буде використовуватися для векторизації даних.

2.9 Сервіс decision maker

Сервіс decision maker буде відповідати за розрахунок коефіцієнту подібності між сайтами. Результатом роботи сервісу є список коефіцієнтів подібності. Для розробки каркасу сервісу буде використовуватись фреймворк FastStream.

Сервіс має відповідати наступним вимогам:

1. Валідація вхідних даних. У випадку невалідних даних результатом роботи сервісу має бути інформативне повідомлення про помилку.
2. Сервіс повинен зберігати завантажену інформацію на диску.

Загальна схема роботи сервісу зображена на рисунку 2.13.

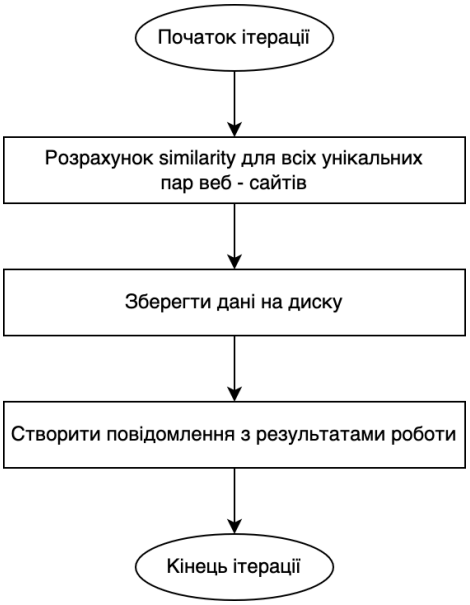


Рисунок 2.13 – Схема роботи сервісу decision maker

На основі аналізу існуючих підходів до розрахунку подібності векторів було обрано метод Cosine Similarity. Цей метод враховує не лише величину векторів, а й їхні напрямки. Ще однією перевагою метода косинусної схожості є його нечутливість до абсолютних розмірів векторів, що робить його придатним для порівняння різних об’ємів даних. Завдяки своїй ефективності та математичній обґрунтованості, метод Cosine Similarity є надійним засобом для розрахунку коефіцієнта подібності між векторизованими ключовими словами.

3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Оцінка продуктивності системи

Оцінка допомагає виявити швидкість та продуктивність методів обробки природної мови, що є критичним у випадках, коли потрібно обробляти великі обсяги текстових даних в реальному часі. Оцінка також виявляє обмеження та слабкі сторони конкретних методів, що надає можливість подальшого вдосконалення алгоритмів та розробки нових стратегій обробки тексту.

Для проведення оцінки продуктивності системи було створено 3 набори даних: із 100, 1000 та 10000 доменів.

Із кожним набором даних метод тестувався у різних конфігураціях:

1. Single script. Весь метод було реалізовано у вигляді одного скрипта.
2. Single async script. Теж саме, що у першому випадку, але завантаження сторінок сайтів було оптимізовано за допомогою асинхронного підходу до завантаження даних з мережі.
3. MS2. Мікросервісна реалізація (згідно опису у розділі 2). Було запущено 2 екземпляри для кожного мікросервісу (за виключенням сервісу desicion maker).
4. MS5. Мікросервісна реалізація (згідно опису у розділі 2). Було запущено 5 екземплярів для кожного мікросервісу (за виключенням сервісу desicion maker).
5. MS10. Мікросервісна реалізація (згідно опису у розділі 2). Було запущено 10 екземплярів для кожного мікросервісу (за виключенням сервісу desicion maker).

Для кожної пари конфігурація системи/набір даних вимірювався загальний час обробки. Результати експерименту зображено у таблиці 3.1

Таблиця 3.1 - Результати виконання експериментів у секундах

Варіант реалізації системи	100 веб-сайтів	1000 веб-сайтів	10000 веб-сайтів
Без мікросервісів	643,86 сек	6270,67 сек	59710,93 сек
Асинхронне завантаження	236,26 сек	2319,61 сек	23140,32 сек
Мікросервіси, 2 екземпляри	146,24 сек	1429,63 сек	14725,93 сек
Мікросервіси, 5 екземплярів	58,69 сек	603,36 сек	6298,01 сек
Мікросервіси, 10 екземпляри	30,20 сек	299,16 сек	3216,56 сек

Оскільки система оперує веб-сайтом, як одиницею вхідних даних, здійснити оцінку продуктивності системи можна визначивши середню кількість часу, яку система витрачає на обробку кожного окремого веб-сайту. Ця оцінка обчислюється за формулою (3.1):

$$\underline{T}(D) = \frac{T_D}{N_D}, \quad (3.1)$$

де T_D - це час, який витратила система на обробку даних D; N_D- кількість елементів у списку D, а $\underline{T}(D)$ - середній час, який витратила система на обробку одного елементу.

Діаграму із розрахунками $\underline{T}(D)$ для різних наборів даних та конфігурацій системи зображено на рисунку 3.1.

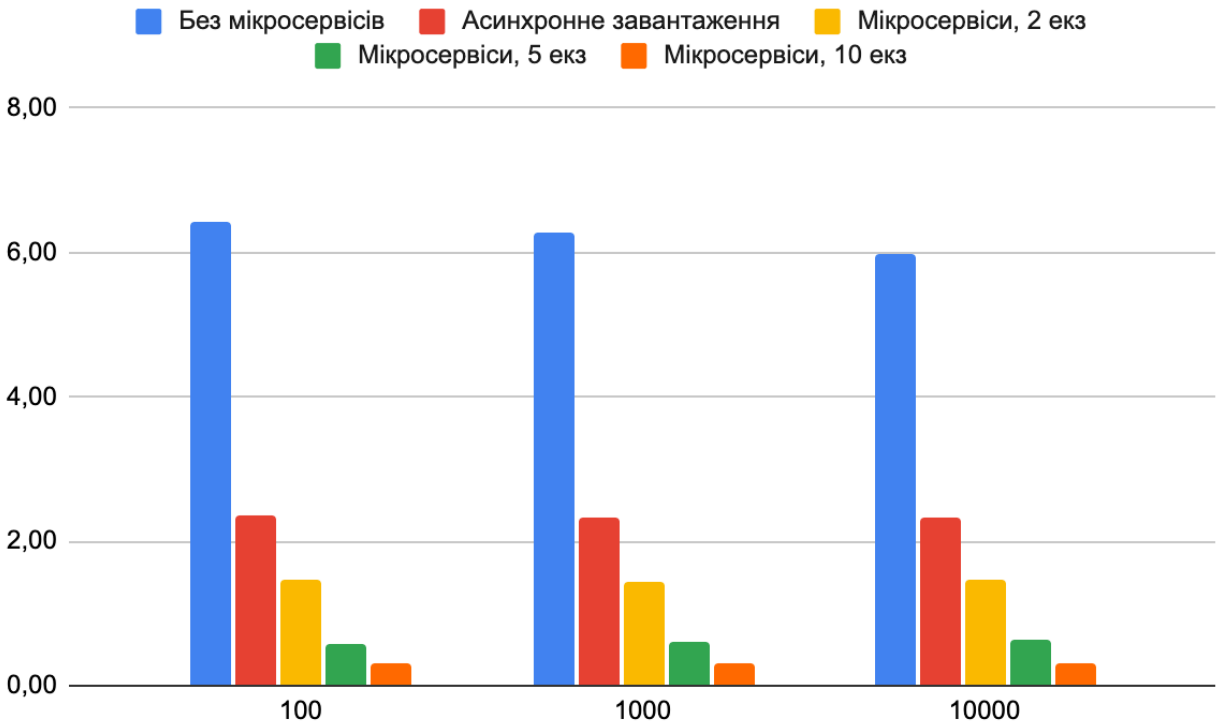


Рисунок 3.1 – Діаграма середнього часу обробки одного веб-сайту у залежності від кількості сайтів та конфігурації системи

На приведеній діаграмі спостерігається стійке та помітне покращення середнього часу обробки одного веб-сайту для конфігурацій системи, де вона поділена на мікросервіси та активно масштабується (порівняно з single-script підходом). Також ця тенденція не змінюється зі збільшенням кількості даних.

Результати наведених експерементів та вимірювань продуктивності однозначно свідчать на користь використання мікросервісної архітектури. Система, розбита на окремі компоненти, демонструє помітно кращу продуктивність та ефективність у порівнянні з її немікросервісним аналогом. Підвищення швидкодії та здатність ефективно масштабуватись стають ключовими перевагами використання мікросервісної архітектури в контексті вирішення завдань обробки великого обсягу даних.

3.2 Оцінка якості визначення дублікатів

Впровадження запропонованої методики дедуплікації веб-сайтів представляє собою ефективний підхід для виявлення дуже схожих сайтів на основі технологій обробки природної мови. Використання цього методу значно спрощує процес пошуку дублікатів, дозволяючи автоматизовано визначати ступінь подібності між двома веб-ресурсами.

Порівняно з ручним процесом виявлення дублікатів, запропонований метод надає можливість обробляти великі обсяги даних веб-сайтів за короткий час. Важливо зауважити, що цей підхід дозволяє виключити ефект впливу візуального враження на об’єктивність оцінки подібності веб-сайтів (у порівнянні з ручною обробкою).

Незважаючи на переваги, існує ймовірність помилок, пов'язаних із відсутністю обробки семантичних особливостей. Через те, що метод векторизації TF-IDF не враховує семантичну особливість, то результати можуть погіршитись у разі використання рідко використовуваних синонімів.

Оцінка результатів методів інформації є необхідною для визначення їхньої ефективності та застосовності. Перш за все, такий аналіз дозволяє оцінити точність та надійність методів у інтерпретації інформації, визначаючи їхню придатність для вирішення конкретних завдань.

Для проведення дослідження було створено список із 200 сайтів:

- було обрано 40 сайтів, які утворюють 23 пари дублікатів;
- інші 160 сайтів було обрано випадковим чином.

Розрахуємо загальну кількість порівнянь за формулою (3.2):

$$N = \frac{n \times (n-1)}{2}, \tag{3.2}$$

де n - кількість веб-сайтів, а N - загальна кількість унікальних пар, утворених вхідними доменами.

Тож загальна кількість порівнянь зроблених системою дорівнює 19900. Розподіл отриманих результатів порівнянь у вигляді діаграм зображено на рисунку 3.2.

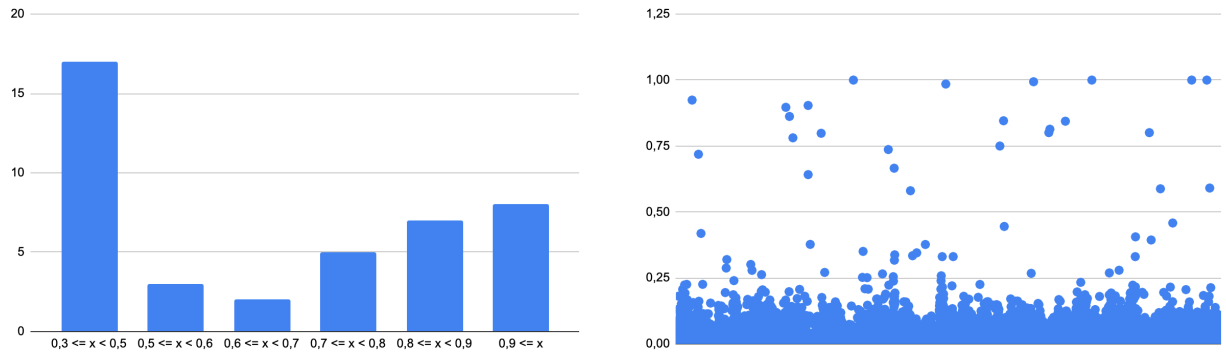


Рисунок 3.2 – Результати порівнянь 100 веб-сайтів

Приклад веб-сайтів, які було вірно позначено системою на рисунку 3.3. Як видно на рисунку 3.3 сайти насправді є сайтами однієї і тієї ж компанії. Але сайти мають різну адресу та наповнення контентом.

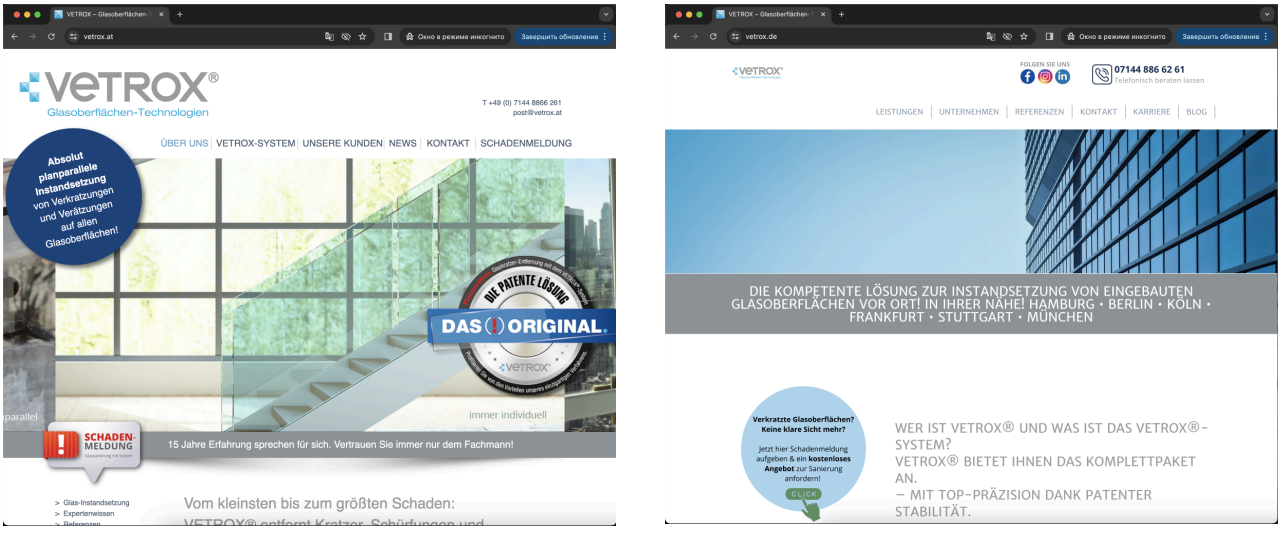


Рисунок 3.3 – Правильно визначена пара дублікатів сайтів

Приклад сайтів, які система хибно визначила, як дублікати зображено на рисунку 3.4.

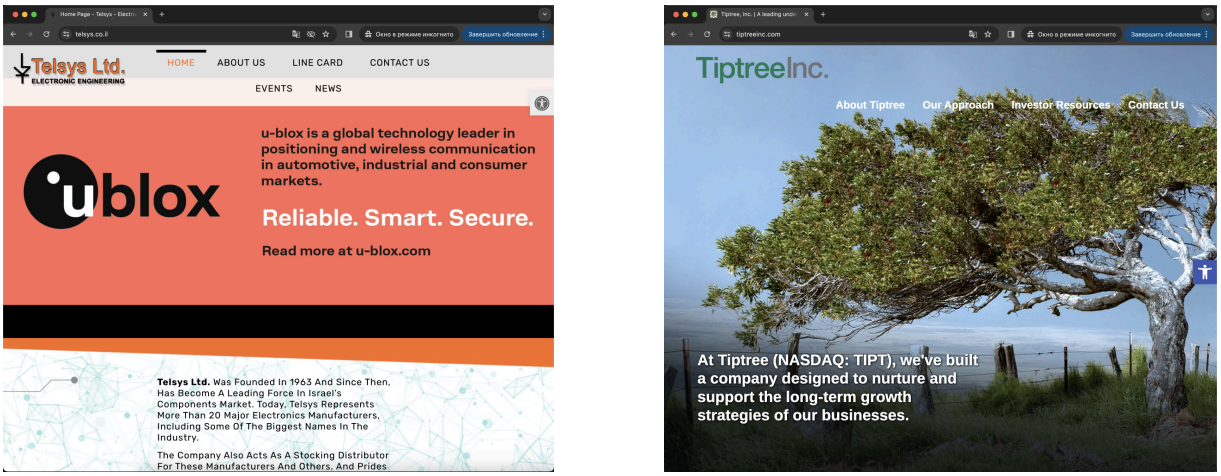


Рисунок 3.4 – Хибно визначена пара дублікатів сайтів

Для цієї задачі загалом, і для обраних даних зокрема, має місце нерівномірний розподіл у результатах роботи. Якщо треба порівняти сайти із вхідного списку за принципом кожен-до-кожного, то “негативних” порівнянь буде завжди у рази більше. По цій причині було обрано спосіб оцінювання результатів F_1.

F_1 Score – це ключовий показник, що використовується для вимірювання ефективності алгоритмів класифікації в ситуаціях, коли точність та повнота мають важливість. Здатність цієї метрики впоратися з незбалансованими даними робить її особливо корисною в різних областях, де ризики помилок класифікації можуть бути нерівномірно розподілені між класами.

Незважаючи на свої переваги, F_1 Score має свої обмеження. Вона є чутливою до вибору порогового значення, яке визначає, коли вважати вихід моделі позитивним або негативним. Це може призводити до неоднозначності у випадках, коли важко визначити оптимальне порогове значення.

F_1 Score визначається взаємозв'язком двох основних метрик - точності і повноти. Точність вимірює, яку частку позитивних випадків наш класифікатор

визначив правильно, тоді як повнота визначає, яку частку всіх позитивних випадків вдалося виявити. F_1 Score узагальнює ці метрики, шукаючи оптимальний баланс між ними.

Точність (precision) розраховується за формулою (3.3):

$$Precision = \frac{TP}{TP + FP}, \quad (3.3)$$

де TP - кількість правильно передбачених позитивних випадків, FP - кількість неправильно передбачених позитивних випадків.

Повнота (recall) розраховується за формулою (3.4):

$$Recall = \frac{TP}{TP + FN}, \quad (3.4)$$

де TP - кількість правильно передбачених позитивних випадків, FN - кількість неправильно передбачених негативних випадків.

Далі потрібно розрахувати метрик F_1 за формулою (3.5):

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}, \quad (3.5)$$

Оцінка результатів роботи буде проводитись для різних порогових значень. Це дозволить зрозуміти, як впливає на якість результатів вибір того, чи іншого порогового значення. Після вивчення розподілу результатів оцінки подібності веб-сайтів (рис. 3.2) було обрано наступні порогові значення для оцінки:

- 0.5;
- 0.6;
- 0.7;
- 0.8.

Результати розрахунку F_1 score для цих порогових значень представлено у таблиці 3.2.

Таблиця 3.2 - Результати розрахунку F1 score для різних порогових значень

Метрика оцінювання	Порогові значення			
	0.5	0.6	0.7	0.8
TP	20	18	17	14
FP	5	4	3	1
FN	3	5	6	9
Precision	0.8	0.81	0.85	0.93
Recall	0.87	0.78	0.74	0.61
F_1	0.83	0.8	0.79	0.74

З таблиці видно, що із зростанням обраного порогу значень спостерігається підвищення точності, але в той же час спостерігається зменшення повноти результатів. Загалом можна вважати прийнятними порогові значення 0.5 та 0.6. Проте варто зауважити, що користувач може адаптувати порогове значення відповідно до своїх вимог до характеристик результату. Адже система дозволяє змінювати порогове значення для прийняття рішення.

3.3 Розширення можливостей використання за рахунок API інтерфейсу

З метою оптимізації функціональності та розширення можливостей використання розробленої системи, було розроблено API інтерфейс.

У світі сучасних інформаційних технологій, HTTP API (Application Programming Interface) стає важливим інструментом для взаємодії програмних систем. Його використання визначається необхідністю забезпечення взаємодії різноманітних програм із забезпеченням ефективного обміну даними.

Однією з ключових переваг HTTP API є його простота використання та розуміння. Він базується на простих принципах взаємодії клієнта і сервера, що спрощує розробку та реалізацію в програмних проектах. Це робить його особливо

ефективним для розробників, які прагнуть мінімізувати складність інтеграції та взаємодії між різними програмними модулями.

Для забезпечення організації взаємодії користувача з системою, користувачу повинна бути надана докладна документація по використанню API. Документація до розробленого API зображена на рисунку 3.5.

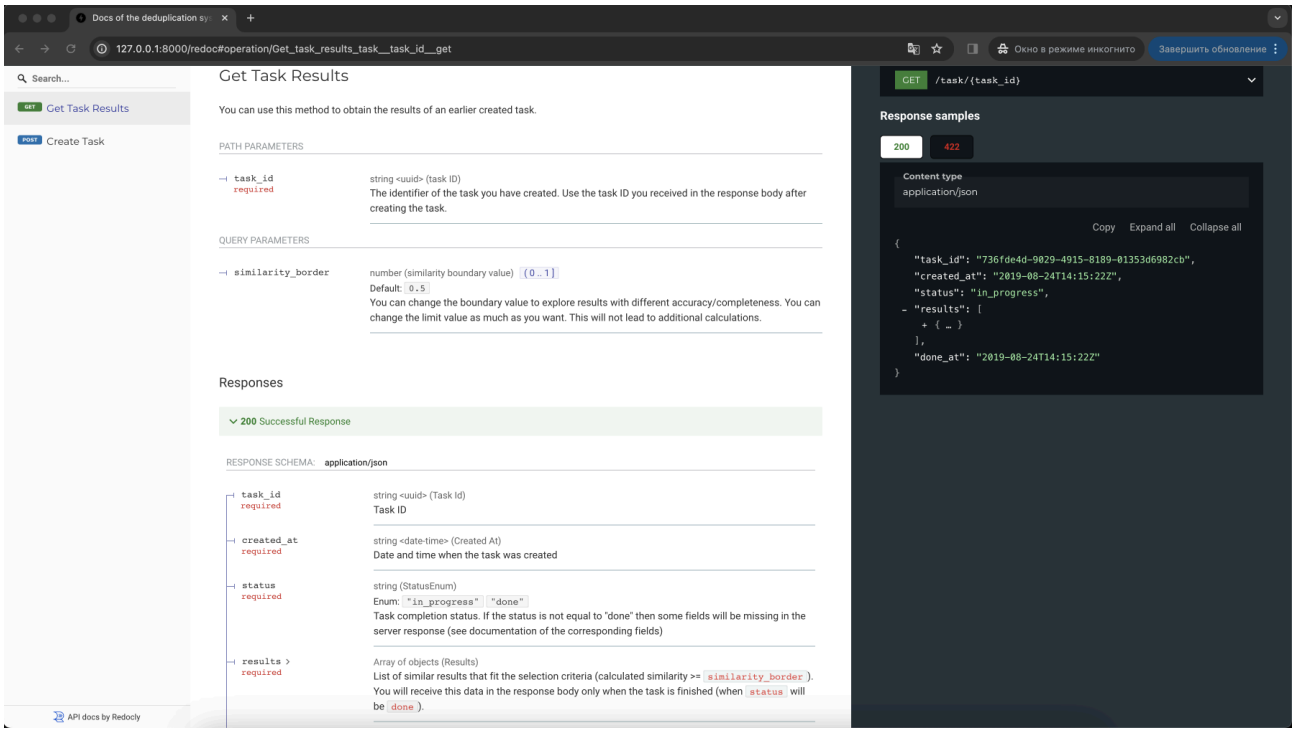


Рисунок 3.5 – Документація до розробленого API

Як видно на рисунку, користувач має доступ до такої інформації:

- список доступних методів;
- загальна інформація про метод;
- вхідні параметри для роботи методу з описом типів даних та додатковою інформацією;
- формат відповіді від серверу з вказанням структури та типів даних для кожного з полів;
- приклад відповіді на вдалі дії користувача, а також приклад відповіді з повідомленням про помилку.

Для початку роботи системи, потрібно створити завдання. Для цього потрібно відправити HTTP POST запит на адресу /task. У тілі запиту повинні бути вказані вхідні дані у форматі JSON. Приклад створення завдання зображено на рисунку 3.6.

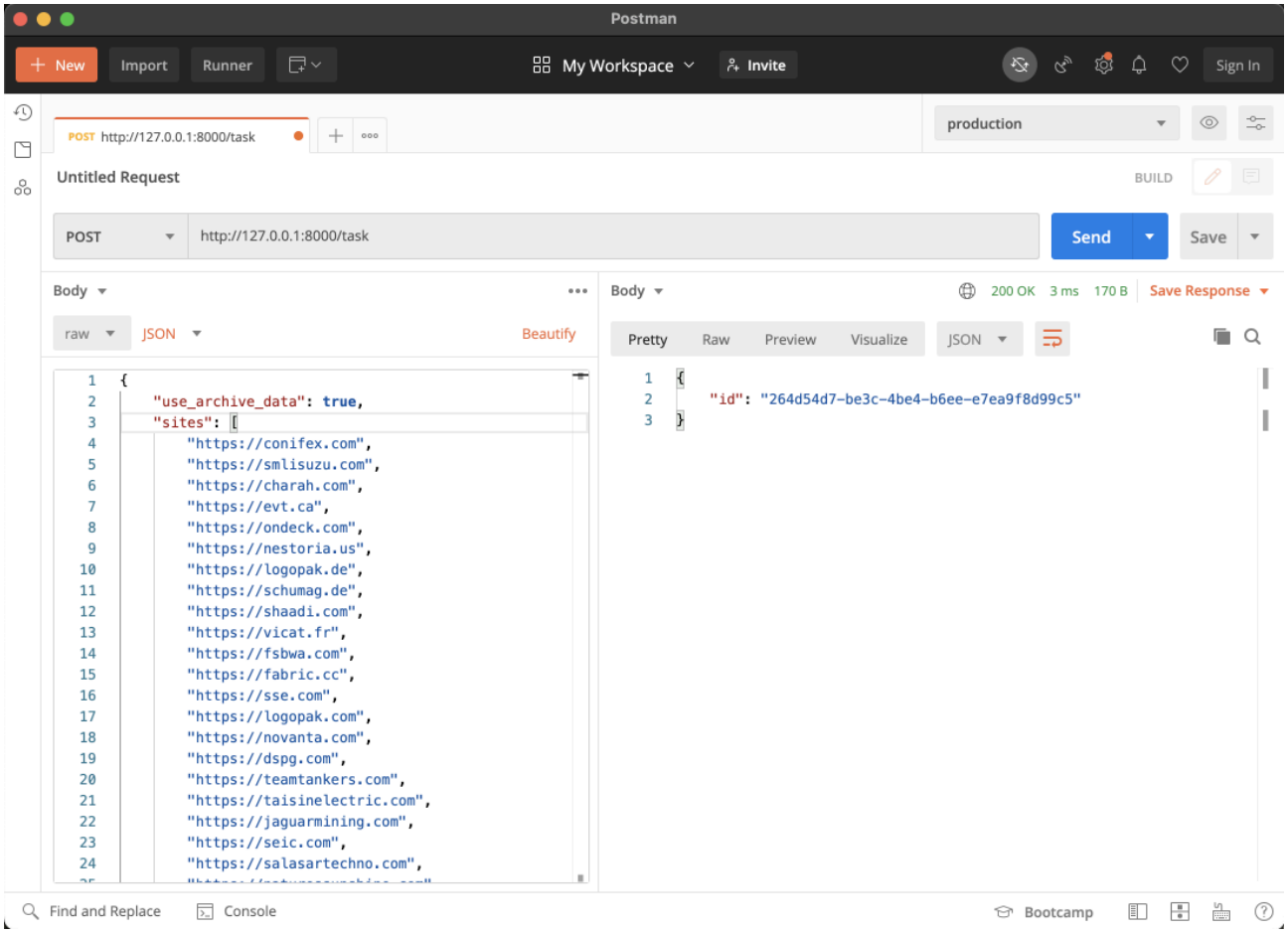


Рисунок 3.6 – Створення нового завдання

Після успішного створення завдання користувач має змогу перевірити статус обробки та результат обробки даних. Для отримання результатів виконання завдання користувач має відправити HTTP GET запит на адресу /task/{task_id_here}?similarity_border={similarity_border}. У посиланні необхідно підставити ідентифікатор цільового завдання та кордонне значення для обрання потенційних дублікатів. За замовченням кордонне значення дорівнює 5.

Якщо завдання все ще знаходиться у процесі виконання, то користувач отримає відповідь зображену на рисунку 3.7.

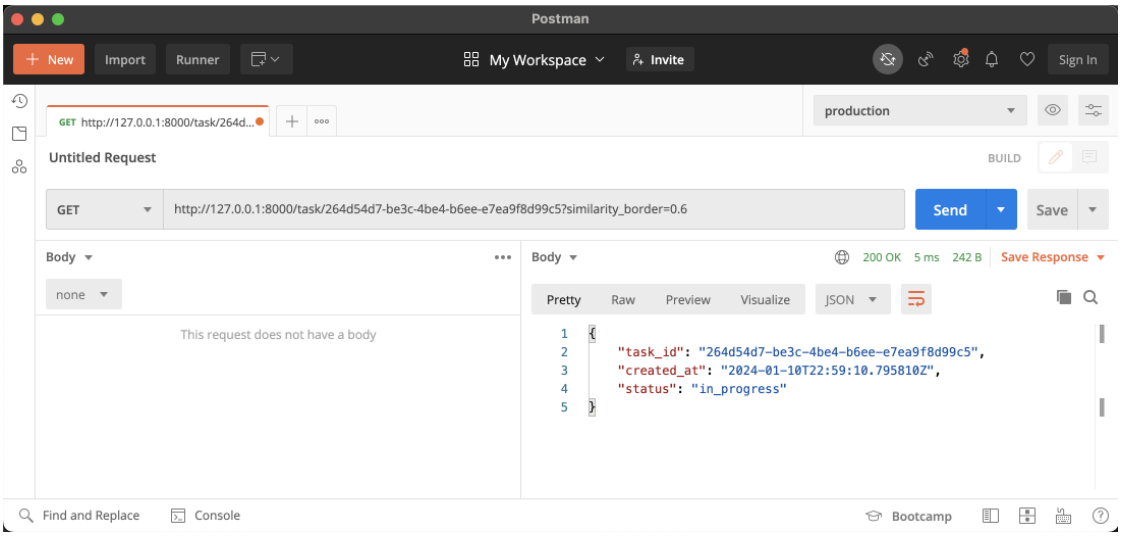


Рисунок 3.7 – Завдання знаходиться у процесі обробки

Також можлива ситуація, що користувач відправив невірний ідентифікатор завдання. У цьому випадку користувач отримає відповідь, зображену на рисунку 3.8.

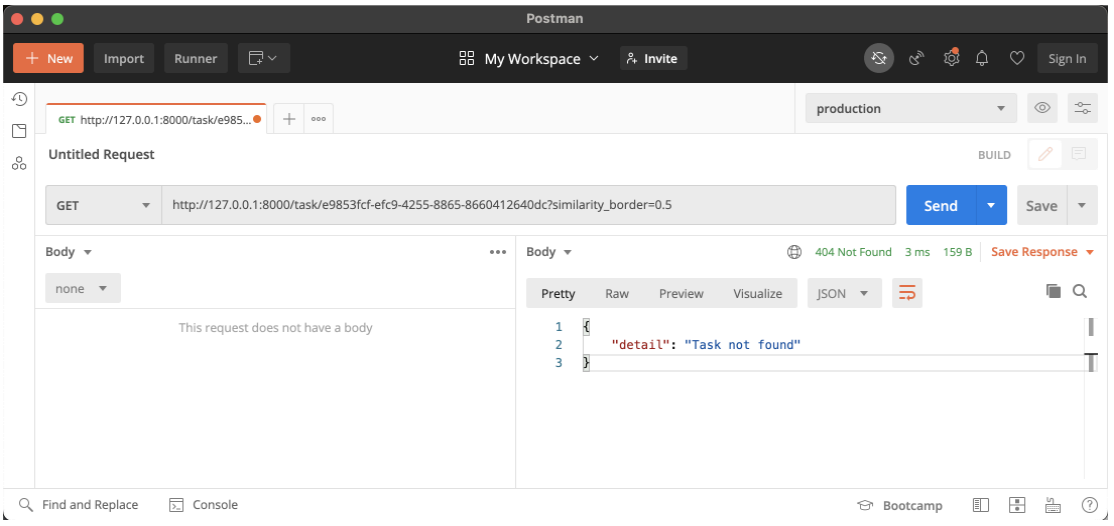


Рисунок 3.8 – Завдання з зазначеним ідентифікатором не знайдено

Коли завдання буде завершено, то користувач отримає у відповідь результати виконання завдання. Така відповідь зображена на рисунку 3.9.

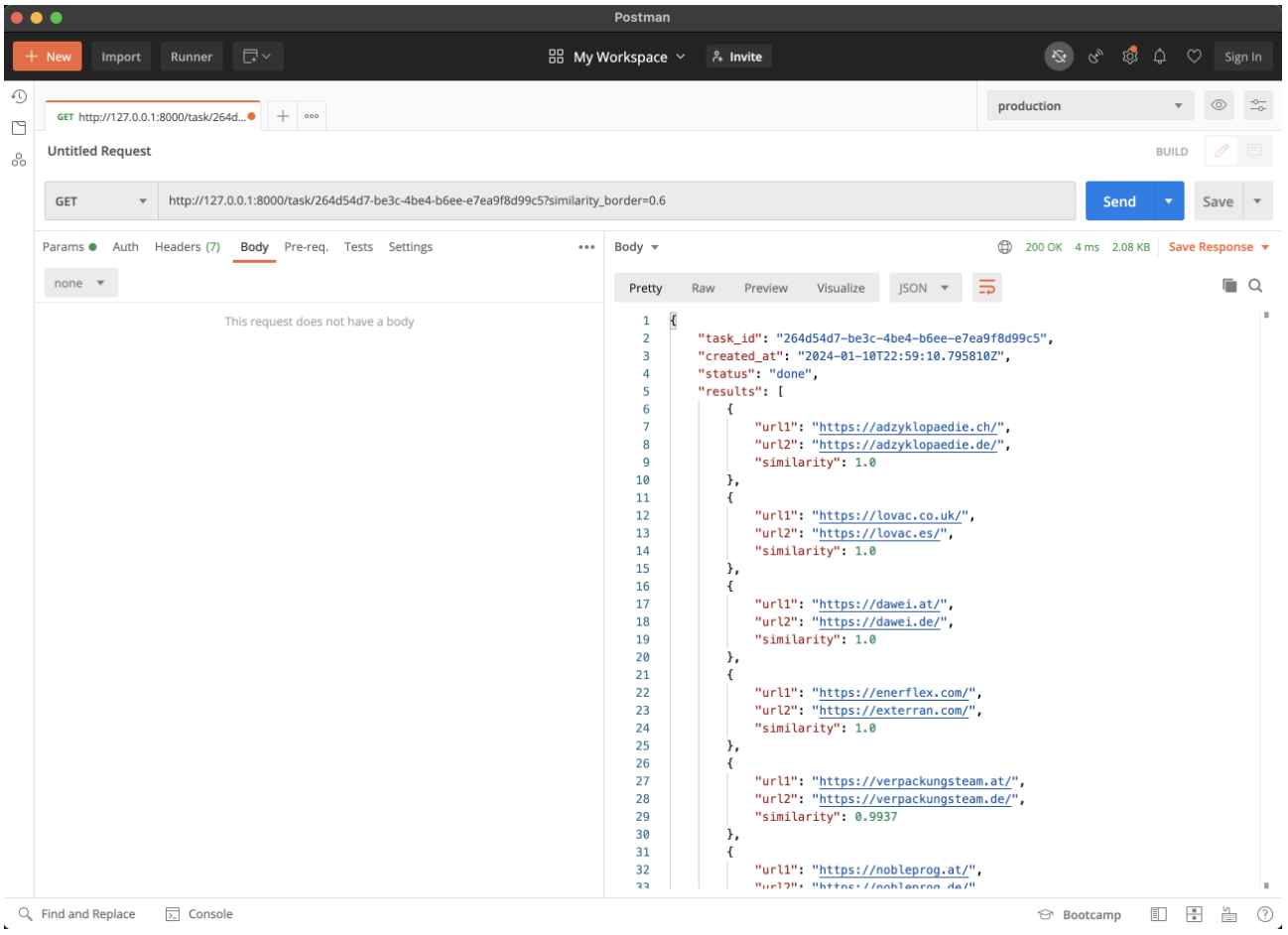


Рисунок 3.9 – Результати виконання завдання

У результаті користувач отримає пари визначених дублікатів за вказаним кордонним значенням. Слід відмітити, що користувач може відправляти запит про результати з різним пороговим значенням необмежену кількість разів. Завдяки тому, що результати порівняння зберігаються у БД, такі запити суттєво не збільшують обчислювальне навантаження на систему.

ВИСНОВКИ

В результаті виконання випускної роботи було виконано всі поставлені задачі.

1. Проведено дослідження існуючих технологій завантаження інформації з веб-сайтів. Це дозволило визначити їхні ключові переваги та недоліки. Ключовими викликами для задачі завантаження інформації з веб-сайтів є великий об’єм даних, різноманітність та нестабільність структури веб-сайтів.

2. Аналіз методів обробки текстової інформації (з веб-сайтів) підкреслив необхідність удосконалення методології порівняння для досягнення необхідної точності при вирішенні задачі дедуплікації. Особлива увага була приділена екстракції та векторизації ключових слів, що є одним із видів метаданих сайту, як методу для оцінки подібності веб-сайтів.

3. Розроблено методику автоматизованого оцінювання подібності веб-сайтів, яка включає етапи екстракції та векторизації ключових слів.

4. Виконано практичну реалізацію методики з використанням мікросервісної архітектури, що дозволяє підвищити продуктивність системи та забезпечує можливість горизонтального масштабування. Взаємодія користувачів із системою стала більш гнучкою та ефективною завдяки впровадженню API інтерфейсу.

5. Оцінка якості роботи системи з точки зору визначення дублікатів сайтів через розрахунок F-міри підтвердила високий рівень точності та стабільність. Одержане значення F-міри (від 0,74 до 0,83 при порогах відбору 0,5 та 0,8 відповідно) свідчить про успішність використання розробленої системи для дедуплікації веб-сайтів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Manku G. S. Detecting Near-Duplicates for Web Crawling [Електронний ресурс] / G. S. Manku, A. Jain, A. D. Sarma // Proceedings of the 16th international conference on World Wide Web. – ACM, 2007. – С. 141-150. – Режим доступу: http://www2007.org/papers/paper215.pdf

2. Broder A. Z. On the resemblance and containment of documents [Електронний ресурс] / A. Z. Broder // Proceedings. Compression and Complexity of SEQUENCES 1997 (SEQUENCES'97). – IEEE, 1997. – С. 21-29. – Режим доступу: https://ieeexplore.ieee.org/document/666633

3. Indyk P. Locality-sensitive hashing scheme based on p-stable distributions [Електронний ресурс] / P. Indyk, R. Motwani // Proceedings of the thirtieth annual ACM symposium on Theory of computing. – ACM, 1998. – С. 253-262. – Режим доступу: https://dl.acm.org/doi/10.1145/276698.276876

4. G. Salton, A. Wong, C. S. Yang. "A Vector Space Model for Automatic Indexing," Communications of the ACM, 1975.

5. K. Sparck Jones. "A Statistical Interpretation of Term Specificity and Its Application in Retrieval," Journal of Documentation, 1972.

6. C. D. Manning, P. Raghavan, H. Schütze. "Introduction to Information Retrieval," Cambridge University Press, 2008.

7. J. M. Tan. "Similarity Measures for Text Document Clustering," Proceedings of the 2005 SIAM International Conference on Data Mining.

8. M. A. Hasan et al. "Similarity Measures for Text Document Clustering," Asian Journal of Information Technology, 2012.

9. Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). "Latent Dirichlet Allocation."

10. Blei, D. M. (2012). "Latent Dirichlet Allocation for Topic Modeling."

11. P. Bojanowski et al. "Enriching Word Vectors with Subword Information." ArXiv, 2016 https://arxiv.org/abs/1607.04606

12. Eisenstein, J. (2019). "Introduction to natural language processing." MIT press.

13. A. Joulin et al. "Bag of Tricks for Efficient Text Classification." ArXiv, 2017. https://arxiv.org/abs/1607.01759.

14. Salton, G., Wong, A., & Yang, C. S. (1975). "A vector space model for automatic indexing." Communications of the ACM, 18(11), 613-620.

15. Manning, C. D., Raghavan, P., & Schütze, H. (2008). "Introduction to information retrieval." Cambridge University Press.

16. SpaCy. "Library Architecture" https://spacy.io/api

17. Millstein, Frank. Natural language processing with python: natural language processing using NLTK. Frank Millstein, 2020.

18. Sharp, Bernadette. (2015). Towards a Cognitive Natural Language Processing Perspective. 10.1007/978-3-319-08043-7_3.

19. Sarica, Serhad, and Jianxi Luo. "Stopwords in technical language processing." Plos one 16.8 (2021): e0254937.

20. Tabassum, Ayisha, and Rajendra R. Patil. "A survey on text pre-processing & feature extraction techniques in natural language processing." International Research Journal of Engineering and Technology (IRJET) 7.06 (2020): 4864-4867.

21. Eisenstein, J. (2019). Introduction to natural language processing. MIT press.

22. Walkowiak, Tomasz, Szymon Datko, and Henryk Maciejewski. "Bag-of-words, bag-of-topics and word-to-vec based subject classification of text documents in polish-a comparative study." Contemporary Complex Systems and Their Dependability: Proceedings of the Thirteenth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX, July 2-6, 2018, Brunów, Poland 13. Springer International Publishing, 2019.

23. Aliero, Abubakar Ahmad, et al. "Systematic Review on Text Normalization Techniques and its Approach to Non-Standard Words." International Journal of Computer Applications 975: 8887.

24. Jalilifard, Amir, et al. "Semantic sensitive TF-IDF to determine word relevance in documents." Advances in Computing and Network Communications: Proceedings of CoCoNet 2020, Volume 2. Singapore: Springer Singapore, 2021. 327-337.

25. Qader, Wisam A., Musa M. Ameen, and Bilal I. Ahmed. "An overview of bag of words; importance, implementation, applications, and challenges." 2019 international engineering conference (IEC). IEEE, 2019.

26. Kaur, Jashanjot, and Preetpal Kaur Buttar. "Stopwords removal and its algorithms based on different methods." International Journal of Advanced Research in Computer Science 9.5 (2018): 81-88.

27. Geetha, Dr & Gomathy, Dr & Yagn, Mr.D.Sri & Praneesh, Sai. (2023). The role of natural language processing. Interantional journal of scientific research in engineering and management. 07. 1-11. 10.55041/IJSREM27094.

28. Chowdhary, KR1442, and K. R. Chowdhary. "Natural language processing." Fundamentals of artificial intelligence (2020): 603-649.

29. Якименко, Дмитро & Катаєва, Євгенія. (2022). Методи та засоби інтелектуального аналізу текстових документів. Вісник Черкаського державного технологічного університету. 43-52. 10.24025/2306-4412.2.2022.259408.

30. Geetha, Dr & Gomathy, Dr & Ram, Mr.P.V. & N, Surya. (2023). Novel study on natural language processing. Interantional journal of scientific research in engineering and management. 07. 1-11. 10.55041/IJSREM27091.

31. Kim, Sang-Woon, and Joon-Min Gil. "Research paper classification systems based on TF-IDF and LDA schemes." Human-centric Computing and Information Sciences 9 (2019): 1-21.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО - КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Магістерська робота

«Оптимізація задачі дедуплікації сайтів на основі аналізу метаданих»

Виконав: студент групи КНДМ-62 Салюк Костянтин Володимирович

Керівник: к.т.н, доц., доцент кафедри ІПЗ Золотухіна Оксана Анатоліївна

Київ - 2024

МЕТА, ОБ'ЄКТ, ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: оптимізація задачі дедуплікації веб-сайтів за рахунок автоматизації визначення подібності веб-сайтів на основі аналізу метаданих.

Об'єкт дослідження: процес визначення подібності веб-сайтів.

Предмет дослідження: методи та засоби вилучення та обробки інформації з веб-сайтів.

ОГЛЯД ІСНУЮЧИХ ПІДХОДІВ

Агрегація даних з веб сайтів

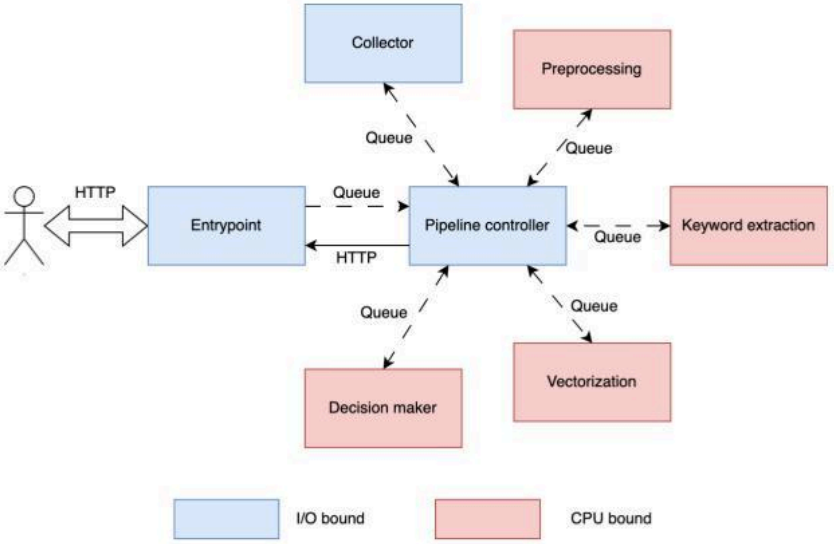
– Бібліотека requests
– Бібліотека aiohttp
– Фреймворк Scrapy
– Бібліотека BeautifulSoup

Обробка контенту та порівняння веб сайтів

– Метод Bag of Words (BoW)
– Doc2Vec
– Метод FastText
– Метод Latent Dirichlet Allocation (LDA)
– Метод хешування
– Метод TF - IDF
– Jaccard Similarity
– Cosine Similarity

ЕТАПИ ВИРІШЕННЯ ЗАДАЧІ ДЕДУПЛІКАЦІЇ ВЕБ-САЙТІВ

АРХІТЕКТУРА СИСТЕМИ



АЛГОРИТМ РОБОТИ МІКРОСЕРВІСУ COLLECTOR

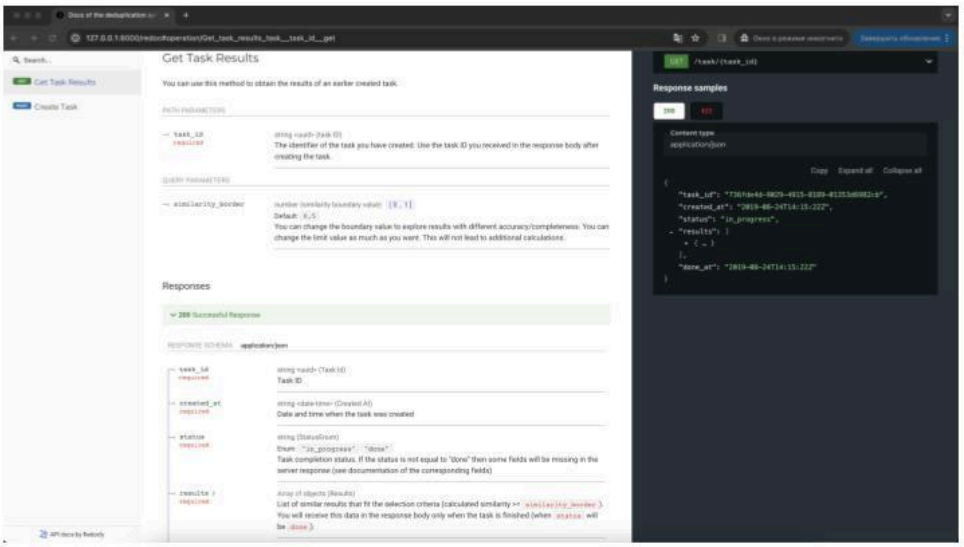
АЛГОРИТМ РОБОТИ МІКРОСЕРВІСУ TEXT PROCESSOR

АЛГОРИТМ РОБОТИ МІКРОСЕРВІСУ KEYWORD PROCESSOR

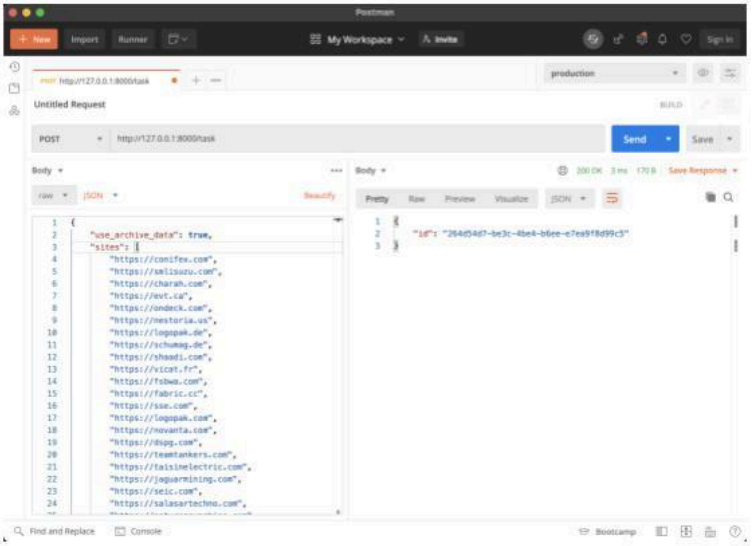
АЛГОРИТМ РОБОТИ МІКРОСЕРВІСУ VECTORIZATION PROCESSOR

АЛГОРИТМ РОБОТИ МІКРОСЕРВІСУ DECISION MAKER

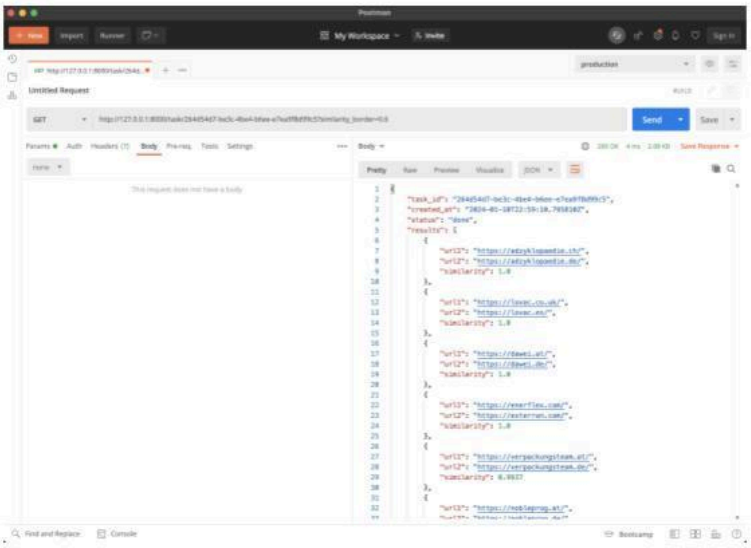
ДОКУМЕНТАЦІЯ API



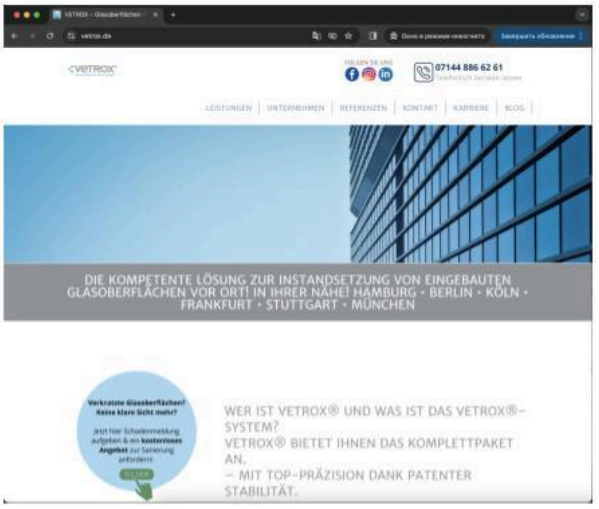
СТВОРЕННЯ ЗАДАЧІ



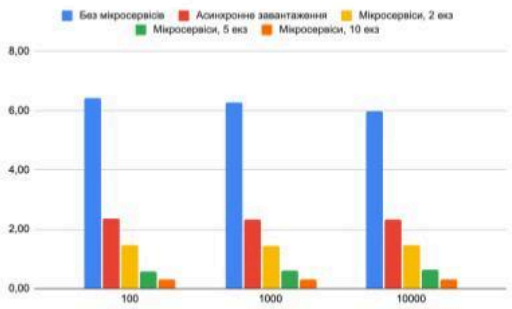
РЕЗУЛЬТАТИ РОБОТИ СИСТЕМИ



ПРИКЛАД ЗНАЙДЕНИХ ДУБЛІКАТІВ



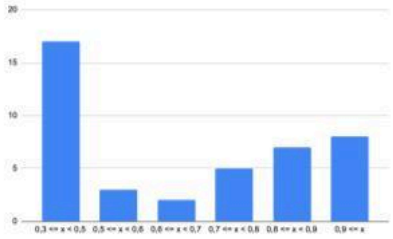
ОЦІНКА ПРОДУКТИВНОСТІ СИСТЕМИ



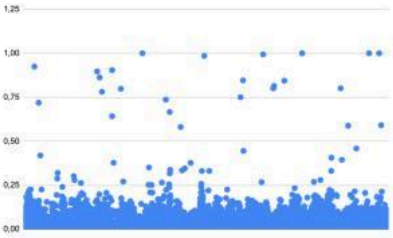
Середній час обробки одного веб-сайту

Варіант реалізації системи	100 веб-сайтів	1000 веб-сайтів	10000 веб-сайтів
Без мікросервісів	643,86 сек	6270,67 сек	59710,93 сек
Асинхронне завантаження	236,26 сек	2319,61 сек	23140,32 сек
Мікросервіси, 2 екземпляри	146,24 сек	1429,63 сек	14725,93 сек
Мікросервіси, 5 екземплярів	58,69 сек	603,36 сек	6298,01 сек
Мікросервіси, 10 екземпляри	30,20 сек	299,16 сек	3216,56 сек

ОЦІНКА ЯКОСТІ ВИЗНАЧЕННЯ ДУБЛІКАТІВ



Розподіл коефіцієнтів подібності



Коефіцієнти подібності 100 сайтів

Метрика оцінювання	Порогові значення			
	0.5	0.6	0.7	0.8
TP	20	18	17	14
FP	5	4	3	1
FN	3	5	6	9
Precision	**0.8**	**0.81**	**0.85**	**0.93**
Recall	**0.87**	**0.78**	**0.74**	**0.61**
F_1	**0.83**	**0.8**	**0.79**	**0.74**

ВИСНОВКИ

1.Проведено дослідження існуючих технологій завантаження інформації з веб-сайтів. Це дозволило визначити їхні ключові переваги та недоліки. Ключовими викликами для задачі завантаження інформації з веб-сайтів є великий об'єм даних, різноманітність та нестабільність структури веб-сайтів.

2.Аналіз методів обробки текстової інформації (з веб-сайтів) підкреслив необхідність удосконалення методології порівняння для досягнення необхідної точності при вирішенні задачі дедуплікації. Особлива увага була приділена екстракції та векторизації ключових слів, що є одним із видів метаданих сайту, як методу для оцінки подібності веб-сайтів.

3.Розроблено методику автоматизованого оцінювання подібності веб-сайтів, яка включає етапи екстракції та векторизації ключових слів.

4.Виконано практичну реалізацію методики з використанням мікросервісної архітектури, що дозволяє підвищити продуктивність системи та забезпечує можливість горизонтального масштабування. Взаємодія користувачів із системою стала більш гнучкою та ефективною завдяки впровадженню API інтерфейсу.

5.Оцінка якості роботи системи з точки зору визначення дублікатів сайтів через розрахунок F-міри підтвердила високий рівень точності та стабільність. Одержане значення F-міри (від 0,74 до 0,83 при порогах відбору 0,5 та 0,8 відповідно) свідчить про успішність використання розробленої системи для дедуплікації веб-сайтів.