

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.912

«До захисту допущено»

Завідувач кафедри

_____ Євгенія СУЛЕМА

« ____ » _____ 2024 р.

Магістерська дисертація

на здобуття ступеня магістра

освітньо-науковою програмою

**«Інженерія програмного забезпечення мультимедійних та
інформаційно-пошукових систем»**

зі спеціальності 121 Інженерія програмного забезпечення

**на тему: «Модифікований метод та програмне забезпечення
автоматизованого пошуку ключових слів у природномовних
текстових даних»**

Виконав:

студент II курсу, групи КП-21мн
Бухаленков Дмитро Олександрович _____

Керівник:

Доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Тетяна Миколаївна _____

Консультант з нормоконтролю:

Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:

старший викладач каф. ПМА ФПМ,
Мальчиков Володимир Вікторович _____

Засвідчую, що у цій магістерській
дисертації немає запозичень з праць
інших авторів без відповідних
посилань.

Студент _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення»

Освітньо-наукова програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___» _____ 2022 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Бухаленкову Дмитру Олександровичу

1. Тема дисертації «Модифікований метод та програмне забезпечення автоматизованого пошуку ключових слів у природномовних текстових даних», науковий керівник дисертації Заболотня Тетяна Миколаївна, к.т.н., доцент, затверджені наказом по університету від «27» березня 2024 р. № 1478-С.
2. Термін подання студентом дисертації «12» червня 2024 р.
3. Об'єкт дослідження: процес автоматизованої обробки природномовних текстових даних.
4. Предмет дослідження: методи, алгоритми та програмні засоби автоматизованого пошуку ключових слів у текстових даних.
5. Перелік завдань, які потрібно розробити:
 - провести збір даних для тестування та порівняння методів автоматизованого пошуку ключових слів у текстових даних;
 - провести аналіз існуючих методів автоматизованого пошуку ключових слів у текстових даних та виявити їх недоліки;
 - розробити модифікований метод автоматизованого пошуку ключових слів у природномовних текстових даних;
 - обрати критерії для вимірювання кількісних характеристик ефективності розробленого методу та порівняння з існуючими аналогами;
 - виконати програмну реалізацію розробленого методу;
 - виконати тестування розробленого програмного забезпечення;
 - провести аналіз отриманих результатів.
6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
 - схема етапів модифікованого методу для автоматизованого пошуку ключових слів у природномовних текстових даних (плакат)

- схема архітектури розробленого програмного забезпечення (плакат)
- структурна схема модуля пошуку пошуку ключових слів та термінів (плакат)
- UML діаграма сценаріїв використання розробленого програмного забезпечення (креслення)
- гістограма порівняння значень абсолютної точності пошуку ключових слів (плакат)
- гістограма порівняння значень повноти пошуку ключових слів за Жаккардом (плакат)

7. Орієнтовний перелік публікацій:

- Тези доповіді “Програмна реалізація модифікованого методу пошуку ключових слів”
- Публікація 2 статей у фахових журналах за спеціальністю 121 «Інженерія програмного забезпечення»

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент кафедри ПЗКС		

9. Дата видачі завдання «11» жовтня 2022 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ознайомлення з предметною галуззю	15.12.2022	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	02.03.2023	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	16.05.2023	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення;	24.10.2023	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження; підготовка матеріалів доповіді на конференції ПМК-2023	15.12.2023	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	20.02.2024	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу	16.04.2024	
8.	Оформлення текстової та графічної частини магістерської дисертації	06.05.2024	

Студент

Дмитро БУХАЛЕНКОВ

Науковий керівник

Тетяна ЗАБОЛОТНЯ

РЕФЕРАТ

Актуальність теми. Основний зміст будь-якого тексту можна описати одним або кількома словами з цього тексту, що відображають його суть. Такі слова називають ключовими словами. В більшості випадків їх кількість становить близько десяти. Іноді ключовими можуть вважати не тільки слова, а й цілі словосполучення і речення.

Незважаючи на просте визначення поняття ключового слова, процес пошуку таких слів є складним аналітичним завданням. Не існує ідеального способу визначити перелік ключових термінів для довільного тексту будь-якої тематики. Кожен текст має свою структуру, стиль викладення, стилістичні особливості написання.

В цілому, задача пошуку ключових слів виникає у багатьох сферах, пов'язаних з обробленням текстових даних. Так інформація про ключові слова використовується при інформаційному текстовому пошуку, класифікації, кластеризації даних. Тому важко переоцінити важливість якісного визначення ключових слів та словосполучень для ефективного розв'язання таких задач. При цьому важливою вимогою до методів визначення ключових слів є можливість їх автоматизації, адже обсяги даних, які проходять через сучасні електронні пристрої та програмні системи, неможливо ефективно обробляти вручну.

За багато років досліджень у галузі АОТ спеціалістами було запропоновано методи, різні за точністю, ефективністю та можливістю застосування. Однак досі триває пошук шляхів покращення точності методів для пошуку ключових слів та підвищення їх універсальності щодо застосування до різних типів текстів.

Таким чином, підвищення точності методів пошуку ключових слів та алгоритмів функціонування відповідних програмних засобів є актуальною задачею.

Об'єктом дослідження є процес автоматизованої обробки природномовних текстових даних.

Предметом дослідження є методи, алгоритми та програмні засоби автоматизованого пошуку ключових слів у текстових даних.

Мета роботи: підвищення ефективності пошуку ключових слів в тексті комп'ютерними системами автоматизованої обробки природномовних текстів за рахунок розробки та реалізації модифікованого гібридного методу автоматизованого пошуку ключових слів.

Методи дослідження. В роботі використовуються загальні методи обробки природномовних текстів, методи обчислення статистичних характеристик текстів, теорія програмування, теорія алгоритмів. Також у дослідженні було застосовано методи статистичного аналізу для експериментальної перевірки теоретичних тверджень, зроблених у дисертаційній роботі.

Наукова новизна. Запропоновано метод автоматизованого пошуку ключових слів, який враховує складні залежності між парами слів в тексті для визначення багатослівних виразів, що дозволяє знаходити ключові терміни, які складаються з кількох слів, та підвищує абсолютну точність пошуку ключових слів у межах від 6,2% до 13,6% в текстах обсягом 150-400 слів та у межах від 1,7% до 34,1% в текстах обсягом 1500-2500 слів у порівнянні з існуючими аналогами.

Практична значущість. На основі запропонованого модифікованого методу виконано програмну реалізацію у середовищі Python за допомогою платформи Python NLTK, та допоміжних пакетів AllenNLP та JiWER. Проведені випробування створеної модифікації гібридного методу пошуку ключових термінів показали ефективність її використання для пошуку ключових термінів в текстах за критерієм абсолютної точності пошуку у порівнянні з існуючими аналогами.

Розроблена програмна реалізація запропонованого методу може бути використана у вигляді модуля комплексної пошукової системи при індексуванні текстових ресурсів.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювалися на науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК 2023 (28-30 листопада 2023 р., м. Київ).

Стаття «Модифікований метод пошуку ключових слів та термінів у текстових даних» в фаховому журналі «Проблеми програмування № 1 (2024)».

Стаття «Дослідження ефективності модифікованого методу автоматизованого пошуку ключових слів у тексті» в фаховому журналі «Інформаційні технології та суспільство №1 (12) за 2024» (прийнято до друку, буде опубліковано 26 червня).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, зроблено оцінку стану проблеми, обґрунтовано актуальність напрямку досліджень.

У першому розділі розглянуто основні види методів автоматизованого пошуку ключових слів в текстах, наведено відомі методи та їх особливості застосування, недоліки та переваги. Розглянуто існуючі сервіси, що надають можливості пошуку ключових термінів.

У другому розділі запропоновано модифікований гібридний метод для пошуку ключових слів у природномовних текстових даних, що враховує складні залежності між парами слів в тексті для визначення багатослівних виразів, що, на відміну від оригінального методу, дозволяє знаходити ключові терміни, що складаються з кількох слів. Окремо визначено метрики, за якими доцільно вимірювати ефективність методу і порівнювати результати з існуючими аналогами.

У третьому розділі наведено опис технологій, що були використані для програмної реалізації методу, та обґрунтовано їх вибір. Описано архітектуру розробленого програмного забезпечення, особливості реалізації, та розглянуто приклад використання.

У четвертому розділі проведено аналіз ефективності розробленої програмної реалізації за критеріями абсолютної точності та повноти пошуку ключових слів в тексті за Жаккарм. Наведено обґрунтування отриманих результатів. Порівняння з існуючими аналогами показують, що запропонована модифікація має більшу абсолютну точність, але дещо меншу повноту за Жаккарм. Описано напрямки для подальшої роботи в рамках даного дослідження.

У висновках проаналізовано отримані результати дослідження.

У додатках наведено графічні матеріали до пояснювальної записки, скріншоти презентації та лістинги коду розробленого програмного забезпечення.

Робота виконана на 115 аркушах, містить 4 додатки та посилання на список використаних літературних джерел з 63 найменувань. У роботі наведено 27 рисунки, 3 таблиці, 17 лістингів.

Ключові слова: ключові слова, ключові терміни, оброблення текстових даних, Python, стенфордська класифікація.

ABSTRACT

Theme urgency. The main content of any text can be described with one or more words from this text that reflect its essence. Such words are called keywords. In most cases, their number is about ten. Sometimes not only words can be considered key, but also whole phrases and sentences.

Despite the simple definition of the concept of a keyword, the process of finding such words is a complex analytical task. There is no perfect way to define a list of key terms for any text on any topic. Each text has its own structure, presentation style, stylistic features of writing.

In general, the task of searching for keywords arises in many areas related to the processing of textual data. Thus, information about keywords is used in information text search, classification, clustering of data. Therefore, it is difficult to overestimate the importance of qualitative definition of keywords and phrases for effective solving of such problems. At the same time, an important requirement for methods of determining keywords is the possibility of their automation, because the volumes of data that pass through modern electronic devices and software systems cannot be efficiently processed manually.

Over many years of research in the field of ATP, specialists have proposed methods that differ in accuracy, efficiency, and applicability. However, the search for ways to improve the accuracy of methods for finding keywords and to increase their versatility in applying to different types of texts is still ongoing.

Thus, increasing the accuracy of keyword search methods and the algorithms of the corresponding software tools is an urgent task.

Object of research is a process of automated processing of natural language text data.

Subject of research is the methods, algorithms and software tools for automated search for keywords in text data.

Research objective is to increase the effectiveness of the search for keywords in the text by computer systems for automated processing of natural

language texts due to the development and implementation of a modified hybrid method of automated keyword search.

Research methods. The thesis uses general methods of processing natural language texts, methods of calculating statistical characteristics of texts, programming theory, algorithm theory. The research also used methods of statistical analysis to experimentally verify the theoretical statements made in the thesis.

Scientific novelty. A method of automated keyword search is proposed, which takes into account complex dependencies between pairs of words in the text to determine multi-word expressions, which allows finding key terms that consist of several words, and increases the absolute accuracy of keyword search in the range from 6.2% to 13.6% in texts with a volume of 150-400 words and in the range from 1.7% to 34.1% in texts with a volume of 1500-2500 words in comparison with existing solutions.

Practical value. On the basis of the proposed modified method, software implementation in the Python environment was developed using the Python NLTK platform and AllenNLP and JiWER auxiliary packages. Tests of the created modification of the hybrid method of searching for key terms showed the effectiveness of its use for searching for key terms in texts according to the criterion of absolute search accuracy in comparison with existing solutions.

The developed software implementation of the proposed method can be used as a module of a comprehensive search system for indexing text resources.

Approbation. The basic points and outcomes of the research have been presented and discussed at the 16th scientific conference for students and postgraduates «Applied mathematics and computing» PMK-2023 (November 28-30, 2023, Kyiv).

Structure and content of the thesis. The master thesis consists of the introduction, four chapters, conclusions, and appendixes.

The introduction provides a general description of the work, assesses the current state of the problem, substantiates the relevance of research.

In the first section the main types of methods of automated search for keywords in texts are considered, known methods and their application features, disadvantages and advantages are given. Existing services that provide the ability to search for key terms are considered.

In the second section a modified hybrid method for searching keywords in natural language text data is proposed, which takes into account complex dependencies between pairs of words in the text to determine multi-word expressions, which, unlike the original method, allows finding key terms consisting of several words. Separately defined metrics are used to measure the effectiveness of the method and compare the results with existing analogues.

The third section provides a description of the technologies that were used for software implementation of the method, and justifies their choice. The architecture of the developed software is described, implementation features, and an example of use is considered.

In the fourth chapter an analysis of the effectiveness of the developed software implementation was carried out according to the criteria of absolute accuracy and completeness of the search for keywords in the text according to Jaccard. The justification of the obtained results is given. Comparisons with existing counterparts show that the proposed modification has higher absolute accuracy, but slightly lower Jaccard completeness. Directions for further work within the framework of this research are described.

The results of the study are analyzed in the conclusions.

The appendices provide graphic materials for the explanatory note, screenshots of the presentation and code listing of the developed software.

The work is performed on 115 sheets, contains 4 appendices and links to the list of used literature sources from 63 titles. The paper presents 27 figures, 3 tables, 17 listings.

Keywords: keywords, key terms, text data processing, Python, Stanford classification.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ	4
ВСТУП	6
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ	8
1.1. Поняття ключового слова	8
1.2. Ключові слова і статистика	9
1.3. Аналіз підходів та методів автоматизованого пошуку ключових слів.....	12
1.4. Аналіз підходів до автоматизованого пошуку ключових слів в Інтернет-ресурсах	18
1.5. Огляд існуючих систем для автоматизованого пошуку ключових слів	24
1.6. Висновки до першого розділу	30
2. МОДИФІКОВАНИЙ ГІБРИДНИЙ МЕТОД АВТОМАТИЗОВАНОГО ПОШУКУ КЛЮЧОВИХ СЛІВ ТА ТЕРМІНІВ В ТЕКСТІ.....	32
2.1. Оригінальний гібридний метод автоматизованого пошуку ключових слів в тексті	32
2.2. Способи покращення ефективності оригінального методу	36
2.3. Проблема автоматизованого пошуку ключових виразів і термінів з кількох слів в контексті оригінального методу	40
2.4. Сучасні системи синтаксичних анотацій	42
2.5. Спосіб застосування інформації про багатослівні вирази в тексті для автоматизованого пошуку ключових виразів і термінів	46
2.6. Модифікований метод автоматизованого пошуку ключових слів та термінів	48
2.7. Вибір метрик для визначення кількісних характеристик ефективності модифікованого методу	51
2.8. Висновки до другого розділу.....	57
3. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ МОДИФІКОВАНОГО ГІБРИДНОГО МЕТОДУ АВТОМАТИЗОВАНОГО ПОШУКУ КЛЮЧОВИХ СЛІВ ТА ТЕРМІНІВ В ТЕКСТІ	58
3.1. Обґрунтування вибору засобів реалізації програмного забезпечення.....	58
3.2. Архітектура розробленого програмного забезпечення.....	67
3.3. Особливості реалізації програмного забезпечення	70

3.4. Приклад використання розробленого програмного забезпечення ...	77
3.5. Висновки до третього розділу	81
4. АНАЛІЗ ЕФЕКТИВНОСТІ ЗАСТОСУВАННЯ МОДИФІКОВАНОГО МЕТОДУ	83
4.1. Критерії оцінювання та аналіз ефективності запропонованого методу	83
4.2. Обґрунтування отриманих результатів	89
4.3. Напрямки подальшої роботи	102
4.4. Висновки до четвертого розділу	104
ВИСНОВКИ.....	106
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ.....	108
ДОДАТКИ.....	115

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

API – Application Programming Interface – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення.

CLI – Command Line Interface – тип інтерфейсу для взаємодії користувача і комп'ютера, де команди подаються через введення текстових інструкцій.

NLP – Natural Language Processing – загальний напрям інформатики, штучного інтелекту та математичної лінгвістики. Він вивчає проблеми комп'ютерного аналізу та синтезу природної мови.

N-грама – в обчислювальній лінгвістиці та теорії ймовірностей: послідовність з n елементів із певної вибірки тексту або мовлення. Елементами можуть бути фонemi, склади, букви, слова залежно від застосування. N-грами зазвичай отримуються з текстового або мовного корпусу.

POS Tagging – part-of-speech tagging – також називається граматичним позначенням або розбором на частини мови, це процес позначення слова в тексті (корпусі) належним до певної частини мови, заснований як на його визначенні, так і на його контексті – тобто, на його зв'язку з суміжними і спорідненими словами у фразі, реченні, або абзаці.

REST – Representational State Transfer – підхід до архітектури мережеских протоколів, які надають доступ до інформаційних ресурсів. В основі REST закладено принципи функціонування Всесвітньої павутини і, зокрема, можливості HTTP.

SEO – Search Engine Optimization – процес коригування HTML-коду, текстового наповнення (контенту), структури сайту, контроль зовнішніх чинників для відповідності вимогам алгоритму пошукових систем, з метою підняття позиції сайту в результатах пошуку в цих системах за певними

запитами користувачів. Чим вище позиція сайту в результатах пошуку, тим більша ймовірність, що відвідувач перейде на нього з пошукових систем, оскільки люди зазвичай йдуть за першими посиланнями.

UML – Unified Modeling Language – уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення.

Корпус текстів – підібрана й оброблена за певними правилами сукупність текстів, які використовують як базу для дослідження мови. Їх застосовують для статистичного аналізу і перевірки статистичних гіпотез, підтвердження лінгвістичних правил цієї мови.

Лема – початкова, словникова форма слова. В українській мові для іменників і прикметників це форма називного відмінка однини, для дієслів та дієслівних форм – форма інфінітиву. Базове поняття в області корпусної лінгвістики та автоматичної обробки природної мови. Процес автоматичного приведення слів тексту до лем називається лематизація.

Парсер – Синтаксичний Аналізатор – це програма або частина програми, яка виконує синтаксичний аналіз. В інформатиці це процес аналізу вхідної послідовності символів, з метою розбору граматичної структури згідно із заданою формальною граматикою.

Семантика – розділ мовознавства, пов'язаний з лексикологією; вивчає значення слів і їхніх складових частин, словосполук і фразеологізмів.

Словоформа – це граматична форма того самого слова, яка виявляє лексичну тотожність із співвідносними словами.

Тезаурус – Синонімічний Словник – одномовний словник, який надає приклади синонімів для конкретного слова.

ВСТУП

Текстові дані зустрічаються майже в усіх сферах життєдіяльності людини. Книги, журнали, вебсайти, соціальні мережі – основним типом вмісту в цих джерелах знань є текст. З того моменту, як комп'ютери ставали все більш поширеними, людство змогло зібрати в електронному вигляді величезну купу текстової інформації. Однак великий обсяг даних потребує належного впорядкування, аби забезпечити зручний та швидкий доступ саме до тієї інформації, яка потрібна шукачеві.

Саме для цього фахівцями були розроблені спеціальні інформаційно-пошукові системи. Вони допомагають відшукати потрібну інформацію в бібліотеках, каталогах та базах знань величезних обсягів. Як відомо, у текстове поле пошуку зазвичай потрібно вписувати пошуковий запит, що стосується вмісту документа, який користувач хоче знайти. Але пошукові системи не лише приймають пошукові запити від користувачів, вони повинні правильним чином додати документи до своєї бази знань і зробити доступ до них зручним. Тому пошукові системи при обробці наявних ресурсів проводять так звану індексацію: спеціальні боти сканують ресурс і намагаються визначити тематику ресурсу та пов'язані з ним ключові слова.

Цей етап пошуку ключових слів є дуже важливим – якість знайдених ключових слів на етапі індексації безпосередньо впливатиме на релевантність результатів пошуку, що будуть видані користувачу пошукової системи. Отже, в інтересах власників пошукових систем і користувачів цих систем аби пошук ключових слів для текстових ресурсів був якнайбільш якісним.

Звичайно, такі величезні обсяги текстової інформації, які проходять в через комп'ютерні системи щодня, неможливо обробити людині вручну. Для цього застосовують спеціальні методи автоматизованого пошуку ключових слів. Однак, на жаль, не існує якогось універсального алгоритму визначення ключових слів людиною, який би підійшов для довільного

тексту будь-якої тематики або об'єму. Це ускладнює створення таких інструментів для комп'ютерних систем, адже комп'ютери не можуть суб'єктивно аналізувати текст, як людина, їм потрібен чіткий набір правил та команд.

За багато років спеціалістами було створено та розроблено багато різних підходів та методів автоматизованого пошуку ключових слів. Найстаріші з них ґрунтуються на законах статистики, таких як частота повторюваності слова в тексті, довжина слова тощо. Більш сучасні – використовують спеціальні словники текстових даних для отримання списку ключових слів з тексту. Ці методи мають як сильні так і слабкі сторони. З появою сучасних комп'ютерів з'явилася можливість застосувати потужності процесорів для поєднання переваг різних підходів, що спонукало спеціалістів на створення гібридних методів. Дослідження в цьому напрямку тривають і досі.

Дослідження в рамках даної магістерської дисертації зосереджено на розробленні модифікації сучасного гібридного методу пошуку ключових слів в природномовних текстових даних, який що враховує складні залежності між парами слів в тексті для визначення багатослівних виразів, що, на відміну від оригінального методу, дозволяє знаходити ключові терміни, що складаються з кількох слів.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1. Поняття ключового слова

Основний зміст тексту можна описати одним або кількома певними словами з цього тексту, що відображають його суть [1]. Такі слова називають ключовими словами.

Ключове слово – це слово в тексті, що разом з іншими ключовими словами може дати читачу загальне уявлення про тематику та зміст тексту [2]. В більшості випадків кількість таких слів становить близько десяти. Іноді ключовими можуть вважати не тільки слова, а й цілі словосполучення і речення [3].

Висока частота повторюваності у тексті, стилістична нейтральність, здатність відображати основний зміст тексту та семантична зв'язаність з іншими такими словами можуть вказувати на те, що слово є ключовим. Однак пошук та вибір ключових слів з тексту є досить складною та нетривіальною задачею.

Досі не існує точно визначеного алгоритму визначення ключових слів людиною, що надає складності реалізації програмних засобів автоматизованого пошуку та визначення ключових слів для комп'ютерних систем. Потрібно вибирати слова, які найбільш точно передають тематику тексту. В той же час, бажано не обирати ключовими слова, які не є загальновживаними [4].

Часто у підходах до визначення ключових слів виділяють етап попереднього оброблення тексту. Під час нього можуть видаляти неінформативні слова – так звані стоп-слова, що скоріше за всього не бувають ключовими, а тільки додають вербального шуму. До таких можна віднести сполучники, прийменники, вигуки, та інші [5].

Під час автоматизованого пошуку ключових слів варто враховувати, що ключові слова можуть мати синоніми у різних реченнях тексту. В такому випадку потрібно вважати їх не різними словами, а одним. Слід також

звернути увагу і на займенники. Іноді перед прийняттям рішення щодо того, наскільки слово відображає суть тексту і чи є воно ключовим, їх замінюють на відповідні іменники.

Одним з підходів до автоматизованого пошуку ключових слів або словосполучень також можна вважати пошук за допомогою N-грам. Вони являють собою послідовність слів, що не розділені стоп-словами та знаками пунктуації [6]. Для додаткового аналізу можна визначати зв'язки між словами в N-грамах, їх типи та кількість. Якщо було побудовано синтаксичну розмітку, є можливість отримати ще більше інформації, керуючись якою визначити основний зміст документа [7].

Вибір ознаки чи метрики, за якою буде проводитися зважування значимості слова, відрізняється в різних підходах. За обраними ознаками для слів обчислюється вага інформативності, проводиться ранжування і встановлюється деякий мінімальний поріг. Після чого слова-кандидати, що мають найбільшу вагу, відбираються як ключові [8]. Варто зазначити, що вибір ознак, таких як частота, типи та кількість зв'язків, частина мови, до якої належить слово та інших, що будуть визначати вагу слів-кандидатів, прямо впливає на результат процесу автоматизованого пошуку ключових слів.

1.2. Ключові слова і статистика

Оскільки немає якогось “золотого правила”, за яким можна точно визначати ключові слова в текстах, було розроблено багато методів та підходів щодо визначення ключових слів. Відомі методики умовно можна розділити на дві категорії: статистичні та експертно-лінгвістичні.

Найстаріші та найвідоміші методи відносяться до статистичних. Якщо аналізувати текстові документи, в усіх можна помітити певні закономірності. Деякі слова зустрічаються часто і в документах зовсім різноманітної тематики. Інші зустрічаються рідко, але несуть в собі вже більше інформації. Отже, одразу може виникнути думка про те, що частота

вживання слова вказує на сильний зв'язок слова із основним змістом або тематикою тексту.

З огляду на це, Джордж Ципф, американський лінгвіст та економіст, вивчаючи статистичні закономірності в природних мовах, у 1949 році виявив кілька закономірностей, які отримав емпірично [9]. Він вважав, що довші слова трапляються у текстах рідше, ніж короткі. На основі свого твердження, він вивів два закони.

Перший закон Ципфа звучить так: якщо виміряти частоту входжень кожного слова в тексті, згрупувати слова з однаковою частотою, і ці частоти розмістити в порядку спадання та пронумерувати, то тоді ймовірність зустріти в тексті деяке наперед вибране число дорівнює відношенню кількості повторів цього слова до загальної кількості всіх слів в тексті:

$$p = \frac{n_i}{|N|}, \quad (1.1)$$

де n_i – кількість входжень слова i в текст; $|N|$ – загальна кількість слів в тексті.

Порядковий номер частоти називається рангом частоти. Таким чином, слова, що найчастіше зустрічаються, мають ранг 1, ті, що за частотою на другому місці – 2 і далі за зменшенням.

Ципф помітив, що добуток частотності слова на його ранг дорівнює деякому постійному значенню:

$$n_i \cdot r_i = C, \quad (1.2)$$

де r_i – ранг слова; C – константа.

Поширеність слова в тексті, в залежності від його частотності, змінюється гіперболічно, тож слово, що знаходиться на другому місці за кількістю входжень, зустрічається в два рази рідше ніж найперше, третє слово – в три рази рідше, ніж перше, і так далі. Константа з (1.2) є майже однаковою для всіх текстів спільної мовної групи. Наприклад, для текстів англійською мовою це значення близьке до 0,1.

Другий закон встановлює відповідність між частотами входження слів в текст та кількістю таких слів з однаковою частотою. Якщо по осі абсцис відкласти кількість входжень слова, а по осі ординат – кількість слів, що відповідають цій частоті, то графік кривої буде мати схожий вигляд для всіх текстів даної мови. В різних мовах графіки подібні, може відрізнятись лише коефіцієнт гіперболи.

Закон Ципфа працює не лише для природних мов, а й прослідковується в інших явищах реального світу: розподіл доходів людей, кількості жителів міст. Пізніше, коли з'явилася можливість перевірити закономірності Ципфа комп'ютерним шляхом, вони були підтверджені і дещо уточнені іншими науковцями [9].

Подальші дослідження показали, що під час застосування закону Ципфа в лінгвістиці, крива графіку може вказати найбільш значущі слова з тексту. Вони знаходяться в середній частині графіка, адже, дійсно, слова, що трапляються в тексті занадто часто, скоріше за все, є неінформативними, або стоп-словами, а які занадто рідко – не несуть достатньо інформації, аби вважатися опорними. Від вибору діапазону, що буде накладатися на центр кривої, залежить результат обраних ключових слів. Для зменшення ширини цього діапазону можливо використовувати ряд підходів. Наприклад, перед обробленням виключати з подальшого розгляду стоп-слова.

Для підвищення точності автоматизованого пошуку ключових слів існують і інші способи. Наприклад, можна аналізувати слова не тільки в межах даного текстового документу, а й всієї колекції. Слова, які часто трапляються серед інших документів, схильні мати менше зв'язку зі змістом безпосередньо вибраного для аналізу документа. І навпаки, якщо слово – майже ексклюзивне до поточного тексту, скоріше за все воно може бути ключовим [10].

Метрика TF-IDF дозволяє врахувати особливості використання слова в конкретному документі і в колекції документів в цілому. Відношення частоти входження слова в документ до загальної кількості слів виражається

терміном TF (term frequency – частота входження слова) і обчислюється наступним чином:

$$TF = \frac{n_i}{\sum_k n_k}, \quad (1.3)$$

де n_i – кількість входжень слова в текстовому документі; $\sum_k n_k$ – загальна кількість слів в текстовому документі.

Друга половина метрики, IDF (inverse document frequency – обернена частота документа), позначає обернену частоту слова в колекції документів і обчислюється за наступною формулою:

$$IDF = \log \frac{|D|}{|d_i \supset t_i|}, \quad (1.4)$$

де $|D|$ – обсяг колекції текстових документів; $|d_i \supset t_i|$ – число текстових документів, в яких зустрічається слово t_i .

Для обчислення значення метрики необхідно перемножити отримані TF та IDF частини. Як видно, TF буде збільшувати результуюче значення, а IDF частинка, що завжди менша за 1, буде зменшувати. Це пояснюється тим, що вага слова для обраного документа буде зменшуватися, якщо воно буде траплятися занадто часто в усіх інших документах з колекції, а отже, матиме меншу значущість саме для обраного тексту [11].

Звісно, ця метрика може давати неточні результати, адже вона спирається на частоту використання слова, але не враховує частини мови, зв'язки між словами. Таким чином, до “ключових” можуть потрапляти стоп-слова, слова з неінформативних частин мови, власних назв, імен.

1.3. Аналіз підходів та методів автоматизованого пошуку ключових слів

Відомі такі підходи та методи автоматизованого пошуку ключових слів в тексті:

- зважування за прямим *Q-параметром* та непрямим *F-параметром*;

- визначення *семантичного поля*;
- автоматизовані системи *TERMINO* та *LEXTER*;
- метод *k-factor*;
- метод *C-value*;
- метод *k-factor* з “*вікном*”;
- метод *зважування іменникових словосполучень* на основі локальної частоти іменників;
- підхід з використанням словників;
- метод *морфологічних шаблонів*.

Розглянемо дані підходи нижче більш детально.

У будь-якому текстовому документі, написаному природною мовою, можна виділити дві частини: тематично нейтральну і тематично марковану. Тематична нейтральна частина буде виражатися законами мови, тоді як тематично маркована буде більшою мірою залежати від змісту і тематики тексту та стилю автора. Виявлення обох частин тексту допомагає у розумінні змісту та особливостей стилю написання, що притаманний даному тексту. Таким чином, якщо в тексті певні слова вживаються досить часто і в різноманітних формах, це буде свідчити про їх безпосередній зв'язок із тематикою цього тексту.

Для пошуку тематично маркованої та нейтральної лексики використовують системне зважування за прямим *Q-параметром* та непрямим *F-параметром*. *Q-параметр* представляє частоту вживання слова, а *F-параметр* – довжину.

Пошук тематично маркованої лексики відбувається за допомогою обчислення частоти використання словоформ, або кількості цих словоформ. Такий підхід базується на припущенні, що чим частіше слово зустрічається або чим більше у нього словоформ в тексті, тим більше воно відображає тематику та зміст останнього. Однак, варто зауважити, що лексично несамотійні слова можуть зустрічатися дуже часто і тим не менш бути лише граматичними засобами мови [12].

Спеціалістами було виявлено залежності між частотою вживання слова та його довжиною, а саме те, що слова, які вживаються часто, є короткими, а які рідше – довгими. Але така закономірність вимагає від слова бути часто вживаним протягом усього тексту. Виходячи з цього, можна зробити висновок, що часте вживання в тексті короткого слова підлягає стандартній поведінці коротких слів в тексті, а якщо довге слово буде нетипово часто повторюваним, то це може вказувати на його відношення до змісту і тематики тексту.

Окремої інформації про частоту і довжину все ж недостатньо для пошуку тематично маркованої лексики, тож необхідно якимось чином порівнювати ці дані між собою. Для цього застосовують зважування словоформ за прямим та непрямым параметром.

Прямий параметр, або Q-вага, характеризує функціонування слова в тексті. Для того, аби обчислити Q-вагу, необхідно розмістити словоформи тексту за спаданням їх частоти, а потім пронумерувати починаючи з 1. Цей номер є рангом слова, причому слова з однаковою частотою отримують однаковий ранг. Таким чином, слова з максимальною частотою будуть мати ранг 1, слова, що мають меншу частоту – 2 і так далі.

В свою чергу, непрямий F-параметр характеризує функціонування слова на тривалому відрізку часу. Для його обчислення слова розташовують за спаданням їх довжин у звуках. Далі аналогічно кожному слову надається ранг від 1 і за збільшенням на 1. Слова однакової довжини у звуках матимуть однаковий ранг.

Далі для визначення тематично маркованої лексики для кожного слова обчислюється різниця Q-ваги та F-ваги. Якщо ця різниця позитивна – слово відносять до тематично маркованої лексики, а у випадку негативної різниці – до тематично нейтральної. Варто зазначити, що значення різниці завжди будуть в межах від -1 до $+1$. Слова, що отримали позитивний результат, можна вважати словами-кандидатами у ключові слова [13].

Іншим підходом до автоматизованого пошуку ключових слів є визначення семантичного поля. Для цього необхідно для кожної словоформи в тексті знайти її абсолютну та відносну частоту. Відносна частота обчислюється як частка від ділення абсолютної частоти на кількість словоформ. Далі з тексту вибираємо ті речення, які містять дане слово. Далі в вибраних реченнях також обчислюємо абсолютну і відносну частоти. Для отримання індексу значимості слова, треба поділити відносну частоту в обраних реченнях на відносну частоту в вихідному цілому тексті. Якщо в результаті обчислень індекс більше одиниці, то словоформу відносять до семантичного поля обраного слова [14].

В 2004 році французький науковець Матьє Роше запропонував системи для автоматизованого пошуку ключових слів французькою мовою: *TERMINO* та *LEXTER* [15]. Обидві системи використовують морфологічну та синтаксичну інформацію слів з тексту.

Спочатку токенізатор намагається зрозуміти, до якої частини мови належить слово. Далі на основі цієї інформації виконується процес лематизації, і для кожного слова визначається нормальна форма. На цьому етапі алгоритми двох систем починають відрізнятися. В системі *TERMINO* застосовується лексично-синтаксичний аналізатор для опису речень, після чого за допомогою шаблонів шукаються ключові фрази. *LEXTER* використовує морфологічну інформацію і шукає максимальні ланцюжки, що містять терміни.

Ще одним методом можна назвати метод *k-factor*, запропонований Марко Бароні для системи *BootCaT*, розробленої для автоматичного формування тематичного корпусу з використанням пошукових систем.

Відповідно до даного методу, спочатку надається декілька вихідних термінів, використовуючи які, система робить пошук і знаходить документи за темою. В цих документах система знаходить вже нові однослівні терміни, використовуючи порівняння частот між сформованим корпусом та “стандартним корпусом”. Нові терміни знову використовуються для

здійснення запитів. Корпус, що отримується в результаті, разом із списком знайдених однослівних термінів використовується для ітеративного пошуку багатослівних термінів. Таким чином, якщо коротший термін-кандидат має не набагато більшу частоту, ніж довший термін-кандидат, до якого він повністю входить, то перевага при відборі надається більш довгому терміну [16].

У вищенаведеному та подібних йому методах на результат впливає наступна закономірність: частота входження довгих термінів зазвичай нижча, ніж у коротких. Для компенсації цієї особливості був розроблений метод *C-value*. Значення термінологічності за методом *C-value* обчислюється для терміну наступним чином:

$$C\text{-value}(a) = \begin{cases} \log_2 |a| \cdot freq(a), & a \text{ не вкладений} \\ \log_2 |a| \cdot freq(a) - \frac{1}{P(T_a)} \cdot \sum_{b \in T_a} freq(b), & a \text{ вкладений} \end{cases} \quad (1.5)$$

де a – слово-кандидат у ключові слова; $|a|$ – кількість слів в словосполученні; $freq(a)$ – частотність a ; T_a – словосполучення, що містять a ; $P(T_a)$ – кількість словосполучень, де є a .

З формули видно, що зі збільшенням довжини та частоти входження терміну-кандидата збільшується і його вага. Однак, вона зменшується при збільшенні числа словосполучень, до яких входить слово-кандидат [17].

Є інший метод, схожий на *k-factor* та *C-value* методи, але він бере до уваги не лише суміжні слова, що стоять одне за одним, а додатково вводить поняття “вікна”, в якому будуть аналізуватися слова і прийматимуться рішення щодо включення їх до вкладених термінів.

Цей метод є ітеративним. На кожній ітерації для кожного елемента (на першій ітерації це одне слово, а на наступних – це сукупності слів) з кандидатів запам’ятовуються його безпосередні сусіди та сусіди у “вікні”, після чого обчислюється частотність для пар. Якщо пара елементів є безпосередніми сусідами у більше ніж половині випадків знаходження в тому ж самому “вікні”, то пара вважається терміном, або частиною терміну.

Пара склеюється в єдиний елемент і додається до загального списку елементів, для розрахунку наступних ітерацій. Метод був спроможний віднайти в текстах досить довгі терміни до 9 слів [18].

В цілому можна сказати, що більшість термінів є іменними групами, хоча й не завжди. Інший запропонований метод, що отримав назву *“Метод зважування іменникових словосполучень на основі локальної частоти іменників”*, за допомогою синтаксичного аналізатора спочатку знаходить іменні групи, які далі розглядаються як терміни-кандидати. Із знайдених елементів виключаються ті, що мають коми та сполучники, такі як “або” і “та”, а головні слова в іменних групах призводяться до словникової форми. Варто зауважити, що це єдиний метод із розглянутих, яким можна отримати терміни із прийменниками всередині [19].

Класичним методом автоматизованого оброблення текстових даних для подальшої класифікації, вилучення даних і пошуку ключових слів є підхід, з використанням словників, або так званих тезаурусів деякої тематики. Але в такого методу є ряд недоліків. По-перше, не можна обробляти тексти довільної предметної області, не маючи відповідного тезаурусу. По-друге, для текстів з вузькою тематикою може просто не існувати достатньо деталізованого словника для отримання адекватних результатів. В таких випадках методи, які не потребують підготовки тезаурусів за тематикою тексту, виграють в універсальності, хоча можуть давати не настільки точні результати, як словникові за найліпших для них обставин [20].

В мовах, де слова часто стоять в непрямих відмінках, наприклад, українській, виникає складність при автоматизованому пошуку ключових слів або складних термінів, адже слова на будь-якому місці в терміні можуть стояти не в називних відмінках. Для вирішення таких проблем можна застосовувати метод морфологічних шаблонів.

Спершу відбувається попереднє фільтрування стоп-слів, після чого кандидати на ключові терміни відбираються шляхом аналізу за

морфологічними шаблонами, що враховують форми слів. Найтипівішими шаблонами є пари: дієприкметник з іменником, прикметник з іменником, іменник з іменником в орудному відмінку та іменник з іменником у родовому [21].

Коли кандидати у ключові словосполучення було відібрано, виникає необхідність обрати найвагоміші з точки зору тематики тексту. Що цікаво, не дивлячись на те, що зазвичай для цього застосовують закон Ципфа, він не працює у випадку ранжування ключових словосполучень. Тому для словосполучень використовують наступну метрику:

$$KeyPhrase(i) = \frac{\max(Frequency)}{Frequency(i)} < \frac{word_num}{3}, \quad (1.6)$$

де $\max(Frequency)$ – максимальна частота терма, що зустрічається найчастіше та всіх його форм в корпусі; $Frequency(i)$ – частота поточного терма; $word_num$ – приблизна кількість відібраних термів.

Методи, що ґрунтуються на законах Ципфа, погано працюють для текстів невеликої довжини, бо в них частоти всіх слів доволі низькі та однакові між собою. В такому випадку для покращення якості результатів пошуку ключових слів необхідно застосовувати дані, що менше залежать від довжини тексту і частот входження слів.

1.4. Аналіз підходів до автоматизованого пошуку ключових слів в Інтернет-ресурсах

Задача автоматизованого пошуку ключових слів неодмінно стоїть перед розробниками пошукових систем, адже зазвичай при використанні таких систем користувачі подають на вхід якусь ключову фразу і бажають отримати список документів та ресурсів, що за тематикою збігаються з основною ідеєю пошукового запиту.

Сама задача автоматизованого пошуку ключових слів є підзадачею більш ширшого завдання – генерації набору ключових слів або термінів для документа, причому таких, що необов'язково входять до даного документу.

Дійсно, якщо взяти дві фрази з однаковим змістом, що відповідає тексту, але одна з них буде більшою мірою цілком входити до тексту, а інша – ні, то можливий такий випадок, що за пошуковим запитом, сформованим на основі другої фрази, не буде знайдено документ, а це не є бажаним результатом [22].

Можна навести такі підходи до автоматизованого пошуку ключових слів в Інтернет-ресурсах:

- метрика *TF-IDF*;
- алгоритм *TF-ICF*;
- підхід з використанням *статичного корпусу зі статей Вікіпедії*, налаштований під пости Twitter;
- *гібридний метод*.

Далі детальніше опишемо кожен з них.

Під час процесу автоматизованого пошуку ключових слів або термінів в тексті є важливим вибір лише найважливіших термінів як ключових [23]. Для зважування термінів існує досить велика кількість підходів, які поділяють на ті, що потребують навчання, і ті, що не потребують. Перша група спирається на попередній аналіз та оброблення корпусу текстів, що означає, що для аналізу важливості слова в деякому тексті необхідно проаналізувати всі тексти з колекції. В другому випадку для пошуку використовують зібрані словникові дані, що є наближеними моделями існуючих слів обраної мови. Для підвищення точності результатів процесу пошуку ключових слів над удосконаленням систем на базі обох груп постійно працюють фахівці [24].

Метрика *TF-IDF* та подібні за принципом до неї для розрахунку ваг слів-кандидатів потребують оброблення всієї колекції документів. Тоді якщо колекція регулярно доповнюється новими документами, метрики кожного разу потребують повного перерахунку всіх документів, що, звісно, не є оптимальним, наприклад для платформ блогів.

Як альтернативу, було запропоновано алгоритм *TF-ICF*, що враховує доповнюваність колекцій документів. Вага кожного терміну обчислюється за наступною формулою:

$$w_{ij} = \log(1 + f_{ij}) \cdot \log\left(\frac{N + 1}{n_j + 1}\right), \quad (1.7)$$

де w_{ij} – вага терміну j в документі i ; n_j – кількість документів в корпусі, в яких термін j зустрічається один і більше разів; N – загальна кількість документів в корпусі; f_{ij} – кількість входжень терміну j в документ i .

Таким чином, аналізуючи достатній статичний корпус текстів досягається складність алгоритму $O(n)$, тоді як для *TF-IDF* складність становить $O(n^2)$ [25].

На основі *TF-ICF* було запропоновано підхід, який використовує статті Вікіпедії в якості статичного корпусу [26]. Було проаналізовано статті, що мають вручну вказаний список ключових термінів. Обчислення ймовірності обрання терміну ключовим в подальшому здійснюється за наступною формулою:

$$P(W) \approx \frac{\text{кількість}(D_{\text{ключове}})}{\text{кількість}(D_W)}, \quad (1.8)$$

де W – термін; $D_{\text{ключове}}$ – документ, де термін W був обраний ключовим; D_W – документ, де зустрічається термін W .

Формулу можна пояснити так: чим частіше термін був обраний ключовим в документах, в яких він з'являється, тим ймовірніше він буде обраний ключовим в наступних документах. Значення оцінки може бути від 1 до 0: чим ближче воно до одиниці, тим вищою буде значимість терміну.

Особливість цього підходу полягає в тому, що оцінка доволі точно може відображати справжню інформативність термінів, адже ключові слова для статей Вікіпедії були відібрані вручну, що додає впевненості в адекватності результату.

Водночас для отримання оцінки необхідна достатня кількість даних, тож слова, що трапляються надто рідко не можуть бути нормально оцінені

цією метрикою. Пропонується розглядати лише терміни, що трапляються в статтях Вікіпедії п'ять разів та більше і розраховувати вагу терміну за формулою:

$$w_i = TF_i K_i, \quad (1.9)$$

де i – порядковий номер терміну; TF_i – кількість входжень терміну в документ; K_i – інформативність терміну отримана на основі розрахунків по Вікіпедії.

Частинка TF у формулі відображає відношення частоти терміну до загальної кількості термінів в текстовому документі, що дозволяє оцінити важливість терміну t_i серед інших кандидатів.

Одним з моментів, який потрібно враховувати щодо такого підходу, є існування кількох синонімів для одного й того ж самого поняття у Вікіпедії. Наприклад, термін “PC” одночасно може означати “Personal computer”, “Prime Cost”, “Pacific Coast” та багато інших понять. Автори методу пропонують усереднити значення інформативності для синонімів одного поняття задля уникнення подальших проблем.

Розробниками було протестовано метод на повідомленнях платформи мікроблогів Twitter. Даний сервіс є відомим своїми обмеженнями на довжину одного запису в 140 символів, а також великим обсягом нових записів, що з'являються щохвилини. Додатково до описаних кроків було проведено попереднє оброблення текстів повідомлень з видаленням специфічних скорочень, які не несуть тематичного змісту, таких як “RT” та @-посилань, а також окремо відібрано хештеги, які фактично є вручну заданими ключовими словами для повідомлення.

Невелика довжина повідомлень вимагає підвищення повноти результатів пошуку, адже частота вживання слів прямує до одиничного входження в межах одного допису. Було вирішено аналізувати N-грами, тобто неперервні послідовності слів. Можна порахувати, що з k слів можливо отримати Q_k N-грам, і це число буде дорівнювати:

$$Q_k = \sum_{i=1}^k i, \quad (1.10)$$

Після знаходження N-грам зі списку потенційних ключових термінів видаляються стоп-слова та неінформативні кандидати, наприклад, слово, яке зустрічається само по собі, буде видалено зі списку. Список для стоп-слів пропонується брати із системи *SMART* [27]. Далі розраховується вага кожного терміну за (1.10), де значення інформативності береться з бази даних.

Пошук ключових термінів зводиться до аналізу деякої кількості текстових документів (або дописів) і обчислення порогового значення ваги для кожного з них. Терміни, що матимуть вагу більше порогової будуть вважатися ключовими [28]. Першим пороговим значенням пропонується брати середнє арифметичне всіх ваг знайдених термінів. Далі проводяться дії для уточнення цього значення.

Не варто забувати і про терміни, отримані на етапі попереднього оброблення з хештегів. Ці терміни автор допису вказав власноручно, тож від них очікується більше адекватності щодо перебування в списку кандидатів у ключові терміни. Це означає, що можна спиратися на вагу хештегів при обранні граничного значення, адже воно повинно не перевищувати мінімальну вагу серед хештегів.

В даному підході вага кожного хештегу, якщо той був знайдений у базі даних, порівнюється із обчисленим пороговим значенням і знижує його, якщо виявляється нижчою. Варто зауважити, що часто користувачі не вказують хештеги в дописах, але за їх наявності цей етап здатний значно покращити результати.

В запропонованому методі всі дописи обробляються в хронологічному порядку. від тих, що були написані раніше, до недавніх. Враховуючи особливості платформи, такий підхід є логічним, адже користувач може в наступних записах повертатися до попередніх тем. Система запам'ятовує знайдені в попередніх дописах ключові терміни і в

наступних повідомленнях вони будуть одразу безумовно знайдені, що знизить порогове значення ваги для даного запису [29].

В кінці алгоритму при остаточному виборі ключових термінів, елементи списку обробляються в порядку збільшення ваг. Так, якщо вага кандидата менша порогового значення, але він обирається ключовим, бо присутній в списку раніше, порогове значення прирівнюється його вазі і таким чином наступні терміни можна автоматично зарахувати як ключові.

Одним із найновіших методів автоматизованого пошуку ключових слів в тексті є запропонований в 2021 році українським спеціалістом О. Яхимовичем так званий *“гібридний метод”* [30]. Він використовує інструменти сучасних програмних синтаксичних аналізаторів для оброблення текстів і отримання необхідних даних для подальшого зважування слів-кандидатів у ключові слова. Наведемо основні етапи методу:

1. Синтаксичний аналіз тексту і отримання даних про зв'язки між парами слів і частини мови, до яких належать слова тексту.
2. Фільтрування пар слів, зв'язки між якими належать до переліку неінформативних.
3. Заміна займенників в парах слів відповідними іменниками.
4. Відсіювання слів, які при синтаксичному аналізі було віднесено до неінформативних частин мови.
5. Фільтрування стоп-слів.
6. Визначення кількості зв'язків для кожного слова з пари.
7. Прийняття перших n слів з найбільшою кількістю зв'язків як ключових (де n – бажана кількість шуканих ключових слів).

Розробники методу запевняють, що запропонований гібридний метод має приріст повноти у межах від 8,1% до 12,7% за метрикою Жаккара, та від 9,1% до 14,3% абсолютної точності пошуку ключових слів у порівнянні з існуючими аналогами. Отже, метод можна вважати одним із найбільш точних серед сучасних розробок.

1.5. Огляд існуючих систем для автоматизованого пошуку ключових слів

Однією з найбільш відомих систем для автоматизованого пошуку семантичних метаданих в текстах природною мовою є вебсервіс *OpenCalais* [31]. Наразі розвитком і підтримкою сервісу займається мегакорпорація Thomson Reuters, що спеціалізується на консалтингових послугах у сфері мас медіа.

В системі семантичні метадані представлені як іменовані сутності. Додатково з ними зберігаються пов'язані події та факти. Ці іменовані сутності розглядаються як ключові слова, терміни та словосполучення, отримані із оригінального тексту.

Для роботи система *OpenCalais* використовує методи оброблення природних мов, машинне навчання та інші алгоритми. Для пошуку семантичних метаданих сервіс застосовує зібрані онтології різноманітних предметних областей і тематик у форматі Resource Description Framework.

На основі вихідного тексту будуються графометрична та морфологічна розмітки, після чого розмічені словосполучення ідентифікуються за допомогою моделі машинного навчання, що була підготовлена для пошуку іменованих сутностей. Між знайденими сутностями далі шукаються семантичні відносини. Результатом даних обчислень є набір RDF-трійок, сформований із сутностей та відношень між ними.

Система *OpenCalais* доступна всім бажаючим у вигляді безкоштовного вебдодатку. Додатково передбачається використання сервісу за допомогою REST API (необхідна реєстрація) з відповідями у XML форматі.

Сервіс *MonkeyLearn* надає різноманітні платні послуги для роботи з текстовими даними: аналіз відгуків у Інтернет-магазинах, вимірювання

задоволеності клієнтів, визначення іменованих сутностей та фактів з тексту, недоліків у бізнес-процесах, пошук ключових слів в текстах та інші.

На сайті сервісу є сторінка з безкоштовною демонстрацією інструменту автоматизованого пошуку ключових слів в тексті [32]. Додатково, окрім пошуку ключових слів, сервіс також класифікує їх на кілька категорій: аспекти, кваліфікатори, точки зору.

Сервіс не розкриває всі подробиці про те, як саме реалізовано пошук ключових слів, адже це їхня комерційна таємниця, але з короткого опису на сайті можна припустити, що використовуються технології машинного навчання та статистичні методи.

WordCount є онлайн сервісом, що дозволяє отримати дані статистики по введеному тексту: визначити кількість слів, символів, приблизний час для прочитання чи проговорювання даного тексту [33]. Додатково також можна отримати короткий підсумок тексту, виправити граматичні помилки, стилізувати текст.

Окремо виділимо інструмент для автоматизованого пошуку ключових слів з тексту. Він реалізований за допомогою аналізу N-грам. Після введення тексту у вікні результатів користувачу відображаються 3 категорії ключових термінів, що складаються з 1, 2 чи 3 слів відповідно.

Сервіс *Komprehend* дозволяє шукати ключові слова та вирази в текстах не тільки через вебінтерфейс, а й за допомогою API запитів [34]. Додатково надає інструменти для аналізу тональності, намірів, визначення сутностей з тексту.

Автоматизований пошук ключових слів реалізовано за допомогою статистичного аналізу за зважування слів в тексті.

Безкоштовний інструмент для автоматизованого пошуку ключових слів *TerMine* був розроблений спеціалістами британського Національного Центру Дослідження Тексту [35]. Він використовує метод C-value для складених термінів та іншу розробку цього дослідницького центру для

пошуку акронімів AcroMine. Варто зазначити, що система придатна лише для оброблення текстів англійською мовою.

Система *Maui Indexer* автоматично знаходить елементи тексту, що характеризують його тематику, а саме: виділяє ключові слова, ключові словосполучення, терміни [36]. Результати аналізу використовують для класифікації текстів. Технологія працює на основі методів оброблення текстів природних мов та машинного навчання.

Роботу системи можна розділити на два основних етапи. На першому етапі виконується початкова побудова та навчання моделі. На другому підготовлена модель використовується для обробки та класифікації текстів.

Розробники протестували систему на текстах з тематиками комп'ютерних наук, сільського господарства, медицини, фізики, біології, біоінформатики, а також на дописах з різноманітних блогів Інтернету.

В стандартній комплектації система підтримує оброблення текстів англійською, французькою та іспанською, але є можливість розширити список підтримуваних мов.

Maui розповсюджується за ліцензією GNU General Public License Version 3 і є вільним кросплатформним програмним забезпеченням.

Інший інструмент під назвою *ContentAnalyzer* використовується для аналізу змісту вебсторінок, формування списків ключових слів та словосполучень, а також для генерації автореферату за текстом [37].

Для роботи система обчислює такі характеристики тексту:

- частота входження слова або словосполучення в документі;
- відношення частоти до загальної кількості слів в документі;
- вага слова, що обчислюється за допомогою частоти та вагових коефіцієнтів;
- відношення ваги слова до кількості слів в текстовому документі.

Програмне забезпечення *ContentAnalyzer* є безкоштовним та працює на пристроях з операційними системами сімейства Windows.

Ще один інструмент, що аналізує вміст вебсайтів та підказує як підвищити шанси видачі його ресурсу в результатах пошуку – *Terminology Management* від компанії *Acrolinx* [38].

Розробники вважають, що кожен бренд повинен мати свій набір ключових термінів, фраз, висловів, які характеризують його діяльність та виділяють з-поміж конкурентів. Такі слова і фрази мають використовувати на сторінках сайту бренду, в офіційних поштових розсилках. Це допоможе бренду набутися самобутності і підвищити позиції в результатах пошукових систем.

Технологія ґрунтується на використанні баз даних, що зберігають терміни, фрази, вирази для бренду. Інструмент має інтеграцію з багатьма офісними додатками, такими як Microsoft Word, Google Docs, Microsoft Outlook. Під час написання текстів або листів інструмент підказує які слова краще використати для бренду.

Для використання інструменту необхідно придбати абонемент на користування.

TreeTagger – це популярний інструмент для створення розмітки тексту [39]. Був створений Гельмутом Шмідтом з Інституту Комп'ютерної Лінгвістики Університету Штутгарту, що в Німеччині. Підтримує оброблення текстів дуже багатьох мов, таких як англійська, німецька, французька, італійська, грецька, китайська, словацька, польська і так далі. При наявності корпусу текстів достатньої якості можна натренувати і на поки що відсутні в підтримці мови.

В основі інструмент використовує алгоритми, засновані на бінарних деревах рішень. Ці дерева будуються рекурсивно на спеціальних тренувальних наборах триграм. На кожному кроці рекурсії алгоритм ділить набори триграм на дві підмножини з максимальною виразністю (самобутністю). Гілка з найбільшою інформативністю вибирається як продовження дерева, після чого відбувається наступний крок рекурсії. В

результаті отримується набір найбільш виразних та інформативних термінів.

TreeTagger є дуже потужним тегувальним засобом, який багато більш багатофункціональних і сучасних систем використовують в своїх основах. Інструмент є безкоштовним для використання і доступний на всіх основних операційних системах.

Term Extractor від компанії FiveFilters – ще одна система для автоматизованого пошуку ключових термінів в текстах або на вебресурсах [40]. Розробники описують процес пошуку ключових слів наступним чином:

- спочатку використовуються лінгвістичні програмні засоби типу Parts-Of-Speech (POS), тобто такі, що будують синтаксичну розмітку тексту;
- даний інструмент вважає за ключові слова лише іменники, тож з тексту формується список іменників-кандидатів у ключові слова;
- за допомогою статистичних підходів в результаті отримується невеликий але досить точний список ключових термінів.

Автори зазначають, що результати пошуку умовно близькі до таких інструментів як TreeTagger та Yahoo Keyword Extraction [41].

FiveFilters використовує систему в інших своїх інструментах. Так, за допомогою сервісу PDF Newspaper можна автоматично формувати газету з новинами з Web за заданими користувачем термінами, а Feed Creator формує списки новин у форматі RSS та JSON.

Term Extractor є безкоштовним сервісом, роботу якого можна випробувати у вебдодатку termextract.fivefilters.org. Інструмент написаний мовою PHP і його вихідний код є відкритим.

Ще один інструмент, що вміє шукати ключові терміни – *MultiTrans TermBase* від MultiCorpora [42]. Він підтримує декілька мов і призначений в основному для використання фахівцями з перекладу.

За допомогою MultiTrans TermBase можна шукати синоніми однослівних або складених (до 5 слів) термінів в корпусах текстів. Система будує бази даних термінологій для подальшого перевикористання для забезпечення узгодженості перекладу термінів у межах ресурсу.

Під час оброблення масивів текстів, система автоматично відбирає всі унікальні слова та фрази. Підтримка різноманітних мов досягається статистичними алгоритмами, що підлаштовуються до будь-якої природної мови. Користувач може вручну складати списки слів на виключення для зменшення шуму в результатах пошуку.

Після аналізу одного і того ж набору текстів різними мовами в дію вступають більш потужні алгоритми, що аналізують тексти і визначають відповідні переклади одних і тих самих слів та термінів на різних мовах. В результаті система демонструє список можливих перекладів для кожного виразу для подальшого ручного прийняття або відхилення користувачем.

За словами розробників, в реальних робочих випадках система аналізує більше 60000 термінів з корпусів текстів обсягом 6 мільйонів слів лише за кілька годин.

SDL MultiTerm Extract – система автоматичного вилучення ключових термінів, що дозволяє пришвидшити процес створення термінологічних словників [43].

Основні можливості системи:

- пошук та вилучення термінології з текстів однієї або декількох мов;
- застосовуються два потужних методи вилучення термінології: статистичний метод *SDL MultiTerm Extract* та лінгвістичний *SDL PhraseFinder*;
- відображення контексту для кожного терміну-кандидата;
- інструментарій для фільтрації термінів-кандидатів для відсіювання небажаних термінів;
- списки слів для виключення;

- створення словника на основі двох паралельних текстів різними мовами.

За словами розробників, система підтримує багато мов і різних форматів файлів – TMX, TMW, TXT, RTF, .doc та інші.

SynchroTerm від Terminotix – потужний інструмент для вилучення ключових термінів і створення термінологічних баз даних для текстів двох різних мов [44].

Для визначення одних і тих самих термінів на різних мовах використовуються статистичні алгоритми. Система підтримує налаштування різноманітних параметрів відбору термінів, таких як мінімальна та максимальна кількість слів в терміні, кількість входжень терміну, змістовність.

SynchroTerm підтримує близько 30 мов і 8 форматів файлів експорту. Програмне забезпечення розповсюджується за моделлю SaaS і є платним.

1.6. Висновки до першого розділу

У даному розділі розглянуто проблему автоматизованого пошуку ключових слів в текстових документах, наведено основні відомі підходи до задачі автоматизованого пошуку ключових слів та існуючі системи.

Ключові слова використовуються для упорядкування великої кількості текстових даних, класифікації, групування. Зокрема, точність автоматичного знаходження ключових слів індексаторами пошукових систем безпосередньо впливає на якість видачі результатів пошуку.

Умовно всі методи можна розділити на дві основні категорії: статистичні та експертно-лінгвістичні. Статистичні методи ґрунтуються на обчисленні частотності входження слова, його довжини та інших подібних значеннях. Такі методи є доволі мово-незалежними та швидкими, але не завжди дають адекватні результати в ситуаціях, коли статистика не відображає реального положення справ, наприклад, в текстах невеликих розмірів. Експертно-лінгвістичні методи використовують наперед зібрані

словники, або тезауруси, і можуть давати більш адекватний результат в певних випадках. Водночас вони ресурсомісткі на етапах збору даних для словників та не є мово-незалежними і вимагають існування словника для вузьких тематик.

Етапом, що прослідковується в усіх підходах до автоматизованого пошуку, є обчислення ваг кандидата у ключові слова або терміни, який відображає значимість кандидата у порівнянні з іншими в межах даного тексту. Значення ваг розраховується за однією або кількома ознаками, що обумовлено конкретним підходом або методом.

Перспективним вважається використання більш універсальних методів, для уникнення обмежень щодо тематики чи обсягу оброблюваних текстів, та сучасних інструментів для підвищення ефективності і точності інструментів автоматизованого пошуку ключових слів.

2. МОДИФІКОВАНИЙ ГІБРИДНИЙ МЕТОД АВТОМАТИЗОВАНОГО ПОШУКУ КЛЮЧОВИХ СЛІВ ТА ТЕРМІНІВ В ТЕКСТІ

2.1. Оригінальний гібридний метод автоматизованого пошуку ключових слів в тексті

За основу модифікації було вирішено взяти вищезгаданий гібридний метод пошуку ключових слів в англомовних текстах, що був запропонований українським фахівцем О.В. Яхимовичем [30]. Метод використовує інструменти сучасних програмних синтаксичних аналізаторів для оброблення текстів і отримання необхідних даних для подальшого зважування слів-кандидатів у ключові слова.

Наведемо основні етапи методу:

1. Синтаксичний аналіз тексту і отримання даних про зв'язки між парами слів і частини мови, до яких належать слова тексту.
2. Фільтрування пар слів, зв'язки між якими належать до переліку неінформативних.
3. Заміна займенників в парах слів відповідними іменниками.
4. Відсіювання слів, які при синтаксичному аналізі було віднесено до неінформативних частин мови.
5. Фільтрування стоп-слів.
6. Визначення кількості зв'язків для кожного слова з пари.
7. Прийняття перших n слів з найбільшою кількістю зв'язків як ключових (де n – бажана кількість шуканих ключових слів).

Прокоментуємо метод більш детально. Розробники методу виділяють дві основні частини методу: у першій виокремлюються ключові слова на основі кількості зв'язків між лексичними одиницями тексту; у другій відбувається фільтрація вербального шуму. Варто зазначити, що метод реалізовано для текстів англійською, але в теорії, за умови існування

якісних моделей для інших мов, метод можна застосовувати для більшості природних мов.

При побудові розмітки тексту для отримання пар лем із типами залежностей між ними на оброблення подається одразу не весь текст, а кожне його речення окремо. Це зроблено для того, аби програмний пакет міг легше обробити кожне речення в ізоляції. Однак для заміни займенників в тексті на відповідні іменники на аналіз кореференційних зв'язків подається одразу весь текст, адже частіше всього деяка сутність буде згадана в наступних реченнях саме за допомогою відповідних займенників.

На рис. 2.1 представлено схему алгоритму отримання даних про зв'язки між парами слів і частини мови, до яких належать слова тексту. На рис. 2.2, в свою чергу, наведено послідовність кроків процесу фільтрації вербального шуму для покращення точності результатів.

Для отримання пар слів використовується *стенфордська класифікація* [45] зв'язків між лексичними одиницями речень тексту. Розробниками методу шляхом багаточисленних випробувань було визначено 7 типів зв'язків, що не несуть суттєвого змістовного навантаження і не відіграють важливої ролі в контексті пошуку ключових слів. Це зв'язки: CC, DET, EXPL, FIXED, PUNCT, REF, ROOT.

Заміна займенників на відповідні іменники відбувається за допомогою аналізу кореференційних зв'язків між словами в тексті.

Для фільтрації слів, що відносяться до неінформативних частин мови, автори гібридного методу використовують *класифікацію Пенна* [46] і виділяють 21 тег з цієї класифікації: CC, CD, DT, EX, IN, LS, MD, PDT, POS, PRP, PRP\$, RP, SYM, TO, UN, WDT, WP, WP\$, WRB, -LRB-, -RRB-.

Для відсіювання стоп-слів розробники методу використовували стандартні списки стоп-слів для англійської мови.

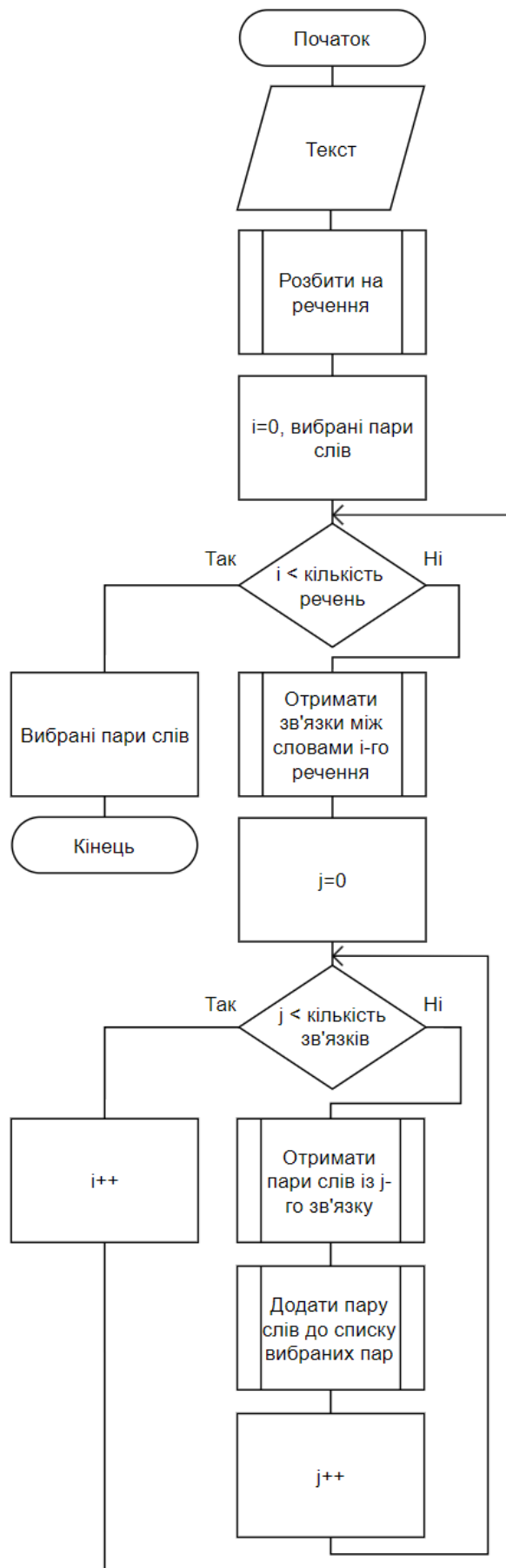


Рис. 2.1. Схема алгоритму отримання даних про зв'язки між парами слів і частини мови, до яких належать слова тексту



Рис. 2.2. Схема послідовності кроків процесу фільтрації вербального шуму

Якщо встановлена необхідна кількість слів і виникає колізія (тобто, потрібно обрати лише частину із слів з однаковою кількістю зв'язків), метод надає можливість вирішити цю проблему у два етапи:

- 1) спочатку для всіх спірних слів з однаковою кількістю зв'язків будується новий “рейтинг” з кількості зв'язків кожного слова зі словами, що вже були визначені як ключові на попередніх етапах;
- 2) якщо після попереднього етапу все ще залишаються слова з однаковою кількістю зв'язків, то ключовими обираються спочатку іменники, потім дієслова, потім інші частини мови.

Розробники методу запевняють, що запропонований гібридний метод має приріст повноти у межах від 8,1% до 12,7% за *метрикою Жаккара*, та від 9,1% до 14,3% *абсолютної точності* пошуку ключових слів в англomовних текстах обсягом 140-1400 у порівнянні з існуючими аналогами. Отже, метод можна вважати одним із найбільш точних серед сучасних розробок, і його модифікація для отримання ще більш точних результатів вбачається перспективною.

2.2. Способи покращення ефективності оригінального методу

Головною особливістю оригінального методу можна назвати визначення кількості синтаксичних зв'язків між словами і відсіювання пар слів з неінформативними типами зв'язків та слів, що належать до неінформативних частин мови. Таким чином, на якість результатів пошуку ключових слів, насамперед, впливає вміст списків неінформативних типів зв'язків та частин мови, що були визначені авторами заздалегідь. Авторами даного дослідження було вирішено удосконалити метод, аби збільшити точність результатів пошуку ключових слів. Було зроблено припущення, що кращих результатів можна досягти, змінивши списки типів зв'язків, що вважаються неінформативними, і тегів частин мови, що не є інформативними для задачі пошуку ключових слів.

Для підтвердження або спростування гіпотези була проведена низка випробувань із зміненими списками типів зв'язків та тегів частин мови. Випробування необхідно було проводити на текстах, для яких вже було

визначено деякий перелік ключових слів авторами тексту. Для цього найкраще підходять тексти статей і їх анотацій у наукових журналах, адже там майже завжди автором додатково вказаний список ключових слів та термінів для кожної статті. Такий список можна вважати еталонним і порівнювати з ним отримані списки ключових слів в результаті роботи автоматизованого методу.

Авторами даного дослідження було випробувано такі зміни в списку неінформативних POS (частини мови):

- виключити зі списку неінформативних частин мови тег *CD*, або *cardinal number* (кардинальне число). Було висунуте припущення, що важливі для змісту ключові слова або фрази можуть містити конкретні числа, наприклад у виразі “Order 767”. Якщо є якийсь загальновідомий історичний наказ під номером 767, включення такого числа до списку ключових слів покращить результати пошуку і ймовірність знаходження даного ресурсу, де описано цей наказ. За оригінальним методом числівник 767 було б повністю виключено зі списку потенційних ключових слів, що, можливо б погіршило якість інформаційного пошуку;
- виключити зі списку неінформативних частин мови тег *RP*, або *particle* (частка). В деяких специфічних поняттях чи термінах може міститися слово-частка. Наприклад, в реченні “Located right on the airfield, guests can watch other planes take off and land.” “off” є частиною “take off”, тобто злітати. Якщо відфільтрувати цю частку зі списку ключових слів, то пошуковий запит, що містить “take off” не буде чітко підходити до набору ключових слів;

Додатково було зроблено спробу покращити точність методу шляхом наступних змін в списку неінформативних типів зв’язків:

- включити до списку неінформативних типів зв'язків тип *ccomp*, або *clausal complement* (комплементна клаузальна конструкція). Частіше всього такий зв'язок трапляється між дієсловом або прикметником і додатком. Зв'язок є досить специфічним і часто не несе достатнього інформаційного змісту, щоб додавати ваги словам-кандидатам у ключові слова. Приклади речень, де трапляються зв'язки такого типу, наведено на рис. 2.3.

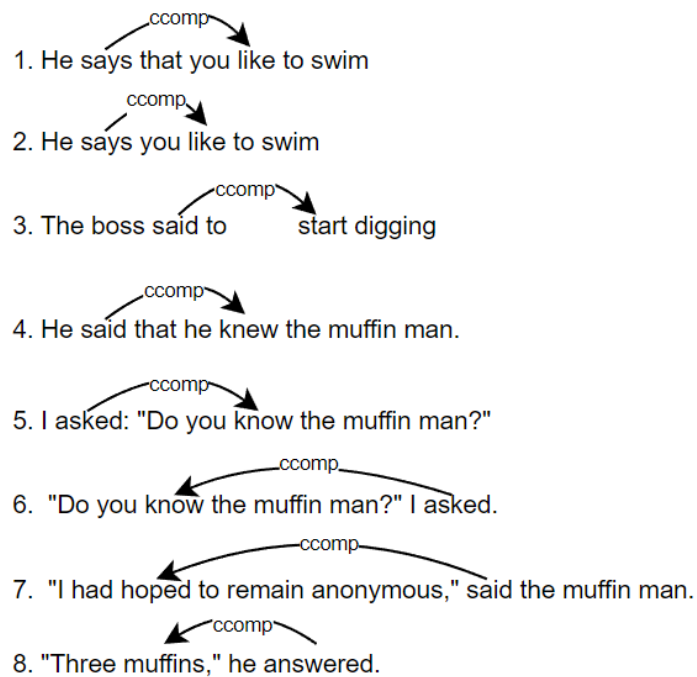


Рис. 2.3. Приклади речень з типом зв'язку *ccomp*

Після перегляду прикладів речень може здатися, що зв'язок досить специфічний і не несе достатнього інформаційного змісту, щоб додавати ваги словам-кандидатам у ключові слова. Отже, варто перевірити, чи вплине внесення зв'язків цього типу до переліку неінформативних на точність результатів роботи методу.

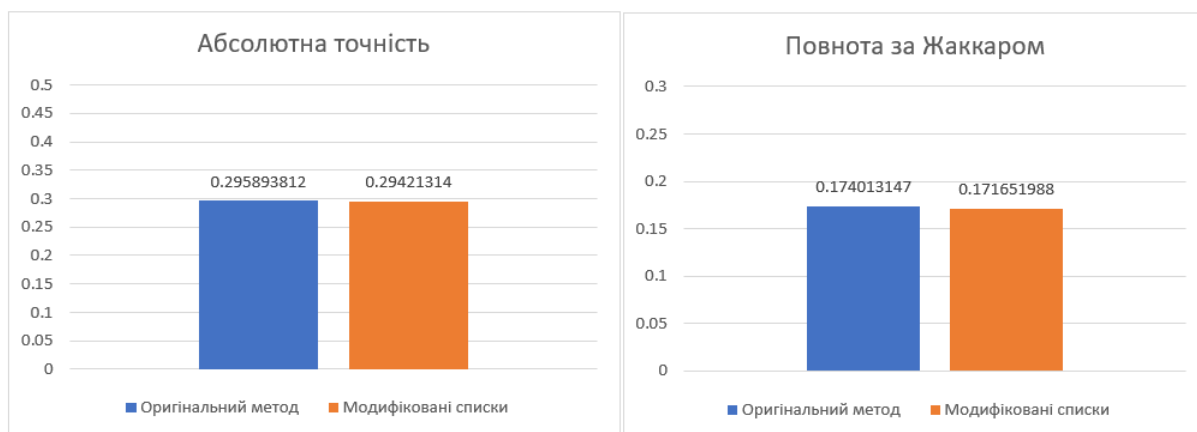


Рис. 2.4. Результати тестування методу з модифікованими
“неінформативними” списками

Були проведені випробування із запропонованими модифікаціями списків. Результати тестування на чисельних текстах тез статей з наукових журналів показали, що списки неінформативних частин мов та типів зв’язків, виділені авторами оригінального методу, є ефективними. Майже в усіх випадках результати не змінювалися, спостерігалися лише невеликі відхилення в ту чи іншу сторону на одне слово. Середні значення абсолютної точності та повноти за Жаккардом виявилися майже однаковими (рис. 2.4). Отже, перша гіпотеза була спростована.

Для формування нової гіпотези було вирішено поглянути на проблему з іншої сторони. З огляду на те, що головним досягненням розробників оригінального методу було вказане підвищення точності (за метриками повноти Жаккара та абсолютної точності) у порівнянні з аналогами, можна вважати, що ними була підібрана така послідовність кроків алгоритму методу, що надає найкращі результати для знаходження однослівних ключових слів. Однак, здатність методу шукати лише однослівні ключові слова виглядає суттєвим недоліком. Потрібно дослідити, чи можна розширити алгоритм і надати йому можливість знаходити ключові словосполучення.

2.3. Проблема автоматизованого пошуку ключових виразів і термінів з кількох слів в контексті оригінального методу

Приведемо як приклад результати аналізу тексту, який згадується авторами в оригінальній роботі. Для тексту тез статті “Variability management in software product lines using adaptive object and reflection” [47] авторами було задано наступні ключові слова та терміни: Software product line, Variability, Adaptive object model, Reflection, Brazilian Satellites Launcher. Результати роботи оригінального методу наведені авторами в наступній таблиці (табл. 2.1).

Таблиця 2.1

Результати роботи оригінального методу для тексту “Variability management in software product lines using adaptive object and reflection”

Слова задані автором тексту	Гібридний метод
software	+
Product	+
Line	+
variability	+
adaptive	—
Object	—
Model	+
reflection	+
Brazilian	—
Satellites	—
Launcher	—

Стає очевидним один недолік методу: він шукає лише ключові слова, але не ключові терміни, що можуть складатися з кількох слів. Для

зображення результату автором оригінального методу також було зроблено “спрощення” – ключові терміни було розбито на окремі слова, з якими було виконано порівняння.

Основна концепція методу – ґрунтування на складних зв’язках між двома лемами – не надає можливості шукати складені ключові терміни чи вирази. Це є досить суттєвим недоліком, адже ключові вирази відіграють важливу роль у пошукових системах. Пошуковий запит, що складається з кількох слів і хоча б частково відповідає визначеним ключовим термінам, буде мати вищий пріоритет при оцінюванні релевантності результатів пошуку, аніж просто пошук по окремим ключовим словам.

Наприклад, пошукова система Google надає різноманітні інструменти для SEO-спеціалістів, за допомогою яких можна оптимізовувати свої вебсайти для видачі в результатах пошуку або налаштовувати показ реклами системою Google AdSense. Звісно, в інтересах рекламодавців показувати рекламу цільовій аудиторії, аби зацікавити відвідувачів онлайн ресурсів у своїх товарах чи послугах [48].

SEO система Google є досить складною і складається з багатьох елементів, але одним із найосновніших є система ключових слів (keywords). Для максимального охоплення Google рекомендує використовувати як broad matching, тобто співпадіння за більш загальними словами, так і більш сфокусовані специфічні ключові слова. Варто зазначити, що під ключовими словами (keywords) Google має на увазі як однослівні ключові слова, так і багатослівні терміни чи вирази, але застосовує їх по-різному. Так, більш загальні співпадіння забезпечуються саме однослівними ключовими словами, а більш специфічні – виразами з кількох слів. При чому, ключові вирази матимуть найбільший пріоритет при пошуку співпадінь із запитом користувача.

Як бачимо, для правильного використання можливостей пошукових систем не достатньо лише однослівних ключових слів чи ключових виразів окремо, необхідно підтримувати актуальні списки обох груп одночасно.

Отже, було вирішено модифікувати метод таким чином, аби можна було отримувати в результаті ключові терміни, що складаються з кількох слів.

2.4. Сучасні системи синтаксичних анотацій

Інформацію про багатослівні вирази в тексті можна отримати, аналізуючи зв'язки між словами, які були отримані під час аналізу тексту синтаксичним аналізатором.

Таблиця 2.2

Таблиця типів синтаксичних зв'язків стенфордської системи Universal

Dependencies

	Nominals	Clauses	Modifier words	Function words
Core arguments	nsubj obj iobj	csbj ccomp xcomp		
Non-core dependents	obl vocative expl dislocated	advcl	advmod discourse	aux cop mark
Nominal dependents	nmod appos nummod	acl	amod	det clf case
Coordination	MWE	Loose	Special	Other
conj cc	fixed flat compound	list parataxis	orphan goeswith reparandum	punct room det

Для класифікації типів зв'язків між словами у тексті частіше всього використовується деяка відома і більш-менш загальноприйнята система

анотацій. Однією з найбільш розповсюджених систем анотацій граматики для різноманітних природних мов є стенфордська класифікація, яка ще називається *Universal Dependencies* [49]. Вона використовується в таких популярних програмних лінгвістичних пакетах, як Python NLTK, DKPro Core. Система налічує 37 визначених типів синтаксичних зв'язків (табл. 2.2).

Нагадаємо ще раз помічений недолік оригінального методу: він здатен шукати лише ключові слова. Якщо буде знайдено одне ключове слово, яке насправді є частиною більш складного ключового виразу, є велика ймовірність, що інші слова з цього виразу не будуть визначені в ролі ключових. Тим паче, якщо і будуть, то результат, що складається з окремих ключових слів, ніяк не індикує, що кілька ключових слів складають один ключовий вираз чи термін.

Можна припустити, що ключові вирази, чи терміни у більшості випадків є деякими багатослівними виразами. Якщо уважно придивитись до таблиці зв'язків системи *Universal Dependencies*, можна побачити секцію *MWE (Multiword Expressions)*, або ж багатослівні вирази. Отже, використання інформації про знайдені багатослівні вирази в тексті може бути корисним в контексті задачі автоматизованого пошуку ключових виразів чи термінів, що складаються з кількох слів.

Згідно з таблицею синтаксичних зв'язків, секція MWE включає в себе три типи зв'язків: *fixed*, *flat* та *compound*. Коротко опишемо кожен з них.

Fixed – використовується для позначення спеціальних службових конструкцій, фіксованих виразів, зворотів. Слова, що мають зв'язок *fixed*, не мають зв'язків інших типів з іншими словами. Приклади таких конструкцій в англійській мові: *as well as*, *because of*, *rather than*. На рис. 2.5 наведено приклади речень, де визначено зв'язки типу *fixed* [50].

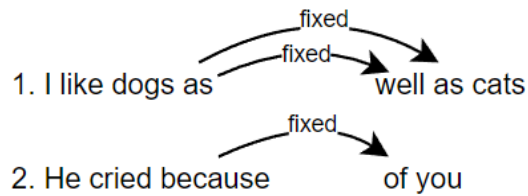


Рис. 2.5. Приклади речень зі зв'язком типу fixed

Flat – цей тип зв'язку використовується для екзоцентричних виразів, тобто таких, де немає ведучого або головного слова. До таких належать імена і дати. На рис. 2.6 наведено приклади визначення зв'язків для імен.

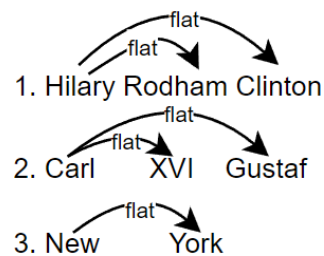


Рис. 2.6. Приклади визначення зв'язку flat для імен

Зв'язок *flat* також застосовується до виразів, де згадується титул або звання особи разом з ім'ям (рис. 2.7).

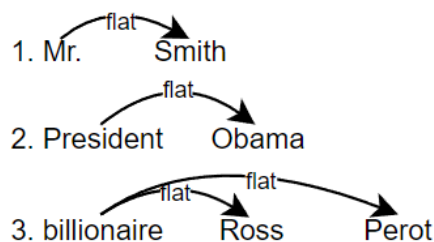


Рис. 2.7. Зв'язки flat для виразів з титулами або званнями

Для складених числових виразів також застосовується зв'язок *flat* [51]. На рис. 2.8 наведено приклад виразів з датами або складними числівниками, де визначено цей тип зв'язку.

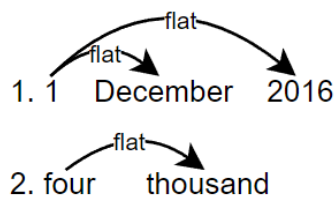


Рис. 2.8. Вирази з датами та складними числівниками, де визначено тип зв'язку flat

Compound – цей тип зв'язку визначено в виразах ендоцентричного типу, де, на відміну від екзоцентричних, є головне або ведуче слово. Такі вирази є сполученнями з кількох частин мови: іменникові сполучення, дієслівні, прикметникові, їх комбінації та іноді серійні дієслівні конструкції. Більшою мірою виражені [52]:

- складними іменниками. Це можуть бути *Іменник + Іменник* (bus stop, fire-flies, football), *Прикметник + Іменник* (full moon, blackboard, software), *Дієслово + Іменник* (breakfast, washing machine, swimming pool), *Іменник + Дієслово* (sunrise, haircut) та інші сполучення (USB cell phone chargers);
- серійними дієслівними конструкціями. Синтаксична конструкція, де представлена послідовність двох або більше дієслів, які функціонують як один предикат та описують одну подію. В сучасній англійській мові майже не зустрічаються, але збереглися деякі вирази: *let's go eat, come live with me*.
- фразовими дієсловами. Комбінація дієслова і прийменника, або дієслова і прислівника, або одночасно дієслова і прийменника з прислівником, яка є окремим членом речення і утворює окрему семантичну одиницю. Найчастіше складаються із власне смислового дієслова та одного або декількох прийменників (рідше прислівників): *keep on, pass out, look up, give up, put off, come across*.

Декілька прикладів, наведених в довіднику Universal Dependencies, зображено на рис. 2.9.

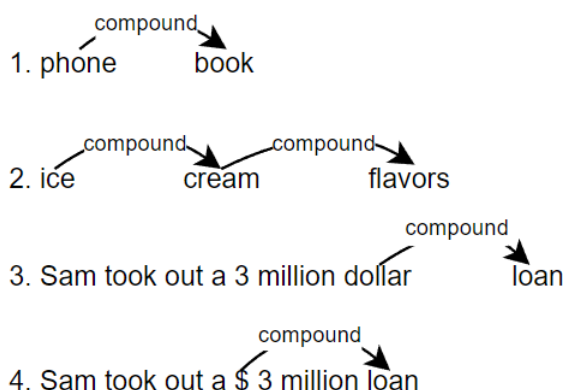


Рис. 2.9. Приклади виразів, де визначено зв'язок типу *compound*

З перелічених трьох типів зв'язків *fixed* представляє найменший інтерес з точки зору вирішення задачі пошуку ключових термінів. *Fixed* позначає спеціальні звороти і усталені сполучення слів, що утворилися історично або іншим шляхом, такі як *instead of*, *rather than* та інші. Вони слугують скоріше пов'язуючими функціональними частинами речень, але смислового навантаження самі по собі не несуть.

Таким чином, з огляду на вищезазначене, для подальшого дослідження було обрано можливість використання знайдених зв'язків типу *flat* та *compound* для отримання списку ключових словосполучень з текстів.

2.5. Спосіб застосування інформації про багатослівні вирази в тексті для автоматизованого пошуку ключових виразів і термінів

Покажемо, як можуть виглядати знайдені вирази з типами зв'язків *flat* та *compound* на прикладі короткого тексту з двох речень “Born in Honolulu, Hawaii, Obama is a graduate of Columbia University and Harvard Law School, where he was president of the Harvard Law Review. He was a community

organizer in Chicago before earning his law degree.” Для цього використаємо один з популярних програмних лінгвістичних пакетів Python NLTK.

Таким чином було знайдено наступні вирази: ['Honolulu', 'Hawaii'], ['Harvard', 'Law', 'Review'], ['Columbia', 'University'], ['Harvard', 'Law', 'School'], ['community', 'organizer'], ['law', 'degree']. Як бачимо, знайдені словосполучення дійсно в деякій мірі відображають зміст тексту, і теоретично підходять в якості ключових слів.

Даний приклад було наведено для дуже малого за обсягом тексту, тож практично кожне знайдене словосполучення або багатослівний вираз можна назвати ключовим. Тепер спробуємо знайти всі такі вирази в тексті більшого розміру, наприклад в анотації до статті “Variability management in software product lines using adaptive object and reflection” з наукового журналу [47]. Отже, було знайдено наступні вирази: ['Line', 'approach'], ['Software', 'Product', 'Line'], ['and', 'increase'], ['to', 'the'], ['because', 'it'], ['reuse', ',', 'which'], ['product', 'line'], ['software', 'product'], ['variability', 'management'], ['product', 'line'], ['software', 'product'], ['use', 'case', 'diagram'], ['variability', 'mechanism'], ['object', 'model'], ['software', 'system'], ['space', 'vehicle'], ['Satellites', 'Launcher'].

Більша частина знайдених виразів виглядає досить адекватною для прийняття за ключові терміни, але є кілька проблем. По-перше, серед знайдених виразів трапляються ті, що не несуть жодного смислового навантаження, такі як ['to', 'the'], ['because', 'it'], ['reuse', ',', 'which']. Такий “шум” можна пояснити недосконалістю реалізації парсера та особливостями структури оброблюваного тексту. По-друге, навіть якщо відсіяти вищезгаданий шум, залишається дуже багато кандидатів у ключові слова. Коли таких виразів занадто багато, вони втрачають свою важливість, адже це просто перелік понять, згаданих в тексті.

Очевидно, що вибір з тексту всіх виразів з типами зв’язків *flat* та *compound* без якої-небудь фільтрації не є повноцінним рішенням задачі пошуку ключових виразів.

Як запевняють автори оригінального методу, його використання дозволяє досить точно знаходити окремі ключові слова [30]. Звідси, можна використати визначений список окремих ключових слів для фільтрування отриманих багатослівних виразів, адже логічно припустити, що якщо текст містить деякі ключові словосполучення або терміни з кількох слів, окремі слова з них ймовірно будуть знайдені алгоритмом оригінального методу.

Таким чином, можна “відфільтрувати” всі знайдені багатослівні вирази, отримані з пошуку в тексті зв’язків flat та compound, використовуючи отримані раніше поодинокі ключові слова. Тобто, якщо в деякому знайденому з тексту виразі, що мав між словами зв’язки flat чи compound, міститься хоча б одне слово з набору ключових слів, такий вираз з великою ймовірністю буде ключовим для даного тексту. Проведені випробування на текстах тез статей наукових журналів підтверджують ефективність даного припущення за критеріями абсолютної точності та повноти пошуку ключових слів, про що йтиметься далі в цій роботі.

2.6. Модифікований метод автоматизованого пошуку ключових слів та термінів

З урахуванням вищенаведеного, пропонується модифікація оригінального гібридного методу з додатковими кроками для отримання списку ключових виразів та термінів до тексту. А саме:

- на етапі номер 1) оригінального методу збирається інформація про всі зв’язки типу flat і compound, з чого отримується набір всіх багатослівних виразів з такими зв’язками в тексті;
- після етапу номер 7) оригінального методу, враховуючи отримані результати, відбувається фільтрація отриманих на етапі 1) багатослівних виразів наступним чином: якщо ключове слово міститься у виразі, він потрапляє до списку ключових, інакше – відсіюється;

- в результаті отримується список ключових слів і список ключових виразів з двох або більше слів.

Список ключових виразів можна буде використовувати при індексації ресурсів в пошукових системах, надаючи їм більший пріоритет збігу із пошуковим запитом користувача, адже ключові терміни більш точно відображають зміст тексту, аніж поодинокі ключові слова.

Відомо, що синтаксичні аналізатори не ідеальні, і вже було продемонстровано, що деякі вирази, отримані в результаті аналізу, не завжди є адекватними, як наприклад вираз ['reuse', ',', 'which'] із вищенаведеного прикладу. Тому для відсіювання такого вербального шуму пропонується виключати вирази, де одним із слів є кома, крапка з комою. Водночас з цим, не можна просто виключати крапки, адже часто вони виступають як скорочення якогось терміну, наприклад вираз-ім'я з типом flat “Robert Downey Jr.” є досить адекватним виразом для того, аби відображати зміст тексту і бути ключовим, але при відсіюванні виразів з крапкою його буде помилково втрачено. Схожа ситуація з дефісами та скісними рисками, вони можуть бути частинами нормальних багатослівних виразів чи термінів, отже, за ними не можна фільтрувати.

При обробленні тексту синтаксичні аналізатори покладаються на те, що поданий текст набрано без помилок за всіма граматичними та синтаксичними правилами. Якщо в тексті будуть траплятися нетипові символи або знаки пунктуації в неправильних місцях, результати аналізу будуть містити похибки і не відображати дійсний стан речей. Необхідно контролювати правильність набору тексту перед подачею його на оброблення до аналізатора.

Варто зазначити, що при порівнянні слів в процесі відсіювання, необхідно приводити слова до початкової форми, щоб уникнути випадків, коли вираз буде відсіяний лише через те, що те саме слово стоїть не в тій формі. В англійській мові немає такого різновиду форм слова, як наприклад

в українській, де є різні відмінки, але множина слова все ж може створити проблеми, тож необхідно проводити лематизацію на етапі порівняння слів.



Рис. 2.10. Схема етапів модифікованого методу для автоматизованого пошуку ключових виразів та термінів

Отже, в загальному вигляді запропонований авторами даного дослідження модифікований гібридний метод є таким:

1. Синтаксичний аналіз тексту і отримання даних про зв'язки між парами слів і частини мови, до яких належать слова тексту.
2. *Отримання з тексту набору всіх виразів з типами зв'язків flat та compound.*
3. Фільтрування пар слів, зв'язки між якими належать до переліку неінформативних.
4. Заміна займенників в парах слів відповідними іменниками.
5. Відсіювання слів, які при синтаксичному аналізі було віднесено до неінформативних частин мови.
6. Фільтрування стоп-слів.
7. Визначення кількості зв'язків для кожного слова з пари.
8. Прийняття перших n слів з найбільшою кількістю зв'язків як ключові (де n – бажана кількість шуканих ключових слів).
9. *Фільтрація отриманих багатослівних виразів за допомогою попередньо отриманих ключових слів.*

На рис. 2.10 наведено загальну схему запропонованого модифікованого методу.

2.7. Вибір метрик для визначення кількісних характеристик ефективності модифікованого методу

Модифікований метод для пошуку ключових слів та виразів застосовує статистичні методи і можливості сучасних програмних лінгвістичних пакетів для аналізу тексту, що робить його доволі швидким. Враховуючи постановку задачі, а саме модифікацію методу з метою уможливлення отримання в результаті аналізу не тільки ключових слів, а й багатослівних ключових термінів та виразів, що може давати більш точні результати, ніж існуючі аналоги, час виконання нового модифікованого методу має менш вирішальне значення, аніж якість і точність отриманих

результатів. Отже, потрібно обрати метрики, що покажуть переваги використання модифікованого методу замість існуючих аналогів за саме знайденими ключовими словами та виразами.

Як вже було сказано вище, складно правильно оцінити точність знайдених ключових слів чи виразів для деякого довільно взятого тексту, адже хоча у ключових слів є деякі визначені характеристики, це досить суб'єктивна оцінка і різні спеціалісти можуть сперечатися щодо правильності визначення того чи іншого слова як ключового. Тому для досягнення максимально можливої об'єктивності було вирішено використовувати для випробування тексти із наперед визначеними набором ключових слів. Серед таких текстів є наукові статті, до яких при публікації в журналі автор статті сам підбирає набір ключових слів чи понять.

Для обґрунтування доцільності використання автори оригінального методу обрали дві метрики: *абсолютну точність* та *повноту* за Жаккаром, пояснюючи це тим, що з огляду на постановку задачі, час знаходження ключових слів для вибраного тексту не має такого вирішального значення, як якість знайдених слів. Окрім того, більшість існуючих інструментів для пошуку ключових слів є онлайн сервісами, обчислювальні потужності яких можуть складатися із мережі серверів. В свою чергу розроблене програмне забезпечення з реалізації запропонованого методу можливо протестувати лише на одному комп'ютері, тож порівняння часу виконання з хмарними серверами не є доречним.

Коротко опишемо суть цих метрик.

Абсолютна точність визначається як відношення кількості правильно знайдених ключових слів за допомогою використання програмної реалізації методу до кількості ключових слів, визначених автором тексту.

Наприклад, якщо взяти множину еталонних ключових слів до деякого тексту як A , а множину ключових слів, що було знайдено програмою як B ,

тоді абсолютну точність a пошуку ключових слів можна обчислити за формулою 2.1:

$$a = \frac{n(A \cap B)}{n(A)}, \quad (2.1)$$

де $n(A \cap B)$ – кількість правильно знайдених ключових слів; $n(A)$ – кількість еталонних ключових слів.

В свою чергу *повнота за Жаккаром* визначається як відношення кількості правильно знайдених ключових слів до загальної кількості еталонних ключових слів і знайдених ключових слів мінус кількість правильно знайдених ключових слів. Повнота за Жаккаром J обчислюється за формулою 2.2:

$$J = \frac{n(A \cap B)}{n(A) + n(B) - n(A \cap B)} = \frac{n(A \cap B)}{n(A \cup B)}, \quad (2.2)$$

де $n(B)$ – кількість програмно знайдених ключових слів; $n(A \cup B)$ – кількість елементів об'єднання обох множин.

Ці дві метрики достатньо легко використати для порівняння двох множин окремих слів, адже тоді можна точно визначити, чи входить слово до переліку визначених автором, чи ні. Але з виразами і ключовими термінами з кількох слів це складніше застосувати. Буде не зовсім справедливо порівнювати вирази один до одного, адже таким чином втратиться велика кількість правильно знайдених ключових виразів, які лише трохи відрізняються.

Наведемо приклад такої проблеми. Стаття з наукового журналу під назвою “Control of Reynolds number in a high speed wind tunnel” [53] має наступний перелік ключових слів, заданий авторами: Blowdown wind tunnel, Pressure control, Mach number control, Reynolds number control. Після застосування до тексту тез даної статті модифікованого методу, знайдено наступні ключові слова з одного слова: 'tunnel', 'relation', 'base', 'test', 'pressure', 'chamber', 'model', 'response', 'define', 'number', 'controller'. Також знайдено наступні ключові терміни та вирази: 'wind tunnel', 'Reynolds number

test', 'test data', 'stagnation pressure', 'control model', 'model simulation', 'system response', 'Reynolds number', 'PI controller'. Постає питання, як оцінити точність отриманих результатів.

Автори оригінального методу значно спрощують задачу: при порівнянні ключові терміни розділюються на кілька однослівних ключових слів: Blowdown wind tunnel стане 3 різними ключовими словами “Blowdown”, “wind” і “tunnel”, і тоді досить просто співставити слова і однозначно визначити чи було знайдене алгоритмом деяке слово як ключове. Але, як вже було згадано вище, однослівні ключові терміни дають лише часткову точність при інформаційному пошуку, для більш точних і специфічних співставлень потрібні багатослівні ключові терміни та вирази.

Не зрозуміло, чи вважати знайдений алгоритмом ключовий термін “wind tunnel” неправильним, коли автором було визначено “blowdown wind tunnel”. З одного боку в результатах алгоритму не вистачає слова “blowdown” для повного співпадиння, з іншого ж боку, 2 з 3 слів повністю співпадають. Аналогічна ситуація із “еталонним” ключовим терміном “reynolds number control” і знайденими за допомогою алгоритму термінами “reynolds number” та “reynolds number test”.

Отже, необхідно знайти метод для порівняння багатослівних виразів з неповним співпадинням. Можна звернути увагу на сфери, де стикаються з подібною задачею оцінки точності співпадиння словесних виразів. Такими можна назвати розроблення і тестування систем автоматичного розпізнавання мовлення або систем машинного перекладу. В обох випадках в результаті роботи систем на виході отримується деякий текст або ж набір словесних виразів чи фраз, які треба порівняти з “еталонним” набором для оцінки якості і точності роботи системи чи методу.

Однією із найвідоміших метрик для оцінки роботи систем розпізнавання мовлення та машинного перекладу є *Word Error Rate (WER)*, або ж Частота Помилкових Слів [54]. Метрика базується на понятті відстані Левенштейна, але працює на рівні слів, а не символів. Визначається як

відношення кількості замінів слів, видалення слів, або додавання слів до довідкового варіанту, щоб привести його до вигляду отриманого автоматичною системою, до кількості слів в довідковому значенні.

Значення метрики *WER* може бути обчислене за наступною формулою:

$$WER = \frac{S + D + I}{N}, \quad (2.3)$$

де *S* – кількість замінів; *D* – кількість видалень; *I* – кількість вставлень; *N* – кількість слів в “еталонному”, або довідковому варіанті.

Наведемо приклад визначення *WER* для деякого отриманого результату деякої автоматичної системи до довідкового значення. Нехай довідковим варіантом є вираз “blowdown wind tunnel”, а отриманий системою варіант – “wind of tunnel”. Для того, аби привести довідковий варіант до автоматично отриманого, нам необхідно провести наступні дії:

blowdown wind tunnel (Deletion)

-> wind tunnel

-> wind of tunnel (Insertion)

Отже необхідно виконати 2 дії (одне видалення і одне вставлення). Кількість слів в довідковому варіанті – 3. Значення *WER* буде дорівнювати:

$$\frac{2}{3} = 66,6\%, \quad (2.4)$$

Метрика має і недоліки: наприклад, вона не індикує, в яких саме аспектах протестована система може бути налаштована помилково, і куди розробнику необхідно звернути увагу. Також усі маніпуляції (видалення, заміна, вставлення) оцінюються з однаковою вагою. Іншими спеціалістами були запропоновані видозмінені варіанти метрики з неоднаковою вагою дій, наприклад, де заміна оцінюється в 1, а вставлення та видалення в 0,5. Вибір варіанту метрики залежить від конкретно взятої системи чи проблематики задачі.

Існує також і обернена до *WER* метрика – *WAcc*, або ж *Word Accuracy*, яка обчислюється як різниця одиниці і значення *WER*:

$$WAcc = 1 - WER, \quad (2.5)$$

Фактично це та сама метрика що і WER, але вимірюється не помилка, а точність.

Для нашої задачі потрібно виміряти схожість, тож для зручності оберемо метрику WAcc. В кінцевому випадку, якщо застосовувати абсолютну точність та повноту за Жаккаром, необхідно якимось чином вирішувати чи вважати отриманий вираз вірно знайденим, чи ні. Отже, необхідно встановити якесь порогове значення метрики, за якого буде зараховуватися збіг термінів. Емпіричним шляхом було вирішено встановити за порогове значення 66,66%. Тобто, якщо при обчисленні значення WAcc для деякого виразу у порівнянні з усіма еталонними значеннями отримуємо хоча б при одному порівнянні 66,66 або більше, то вважаємо, що вираз входить до переліку довідкових, а отже, був визначений правильно.

За такого порогового значення при порівнянні виразу “wind tunnel” із довідковим “blowdown wind tunnel” отримується 66,66%, а це означає збіг виразів. Очевидно, що при порівнянні виразів, які точно співпадають, наприклад “reynolds number test” із переліку знайдених і “reynolds number test” із переліку довідкових, отримується 100%, що є очікуваним.

Таким чином, запропонована модифікація метрик з використанням WAcc дозволяє застосовувати *абсолютну точність і повноту за Жаккаром* для оцінювання точності знайдених багатослівних ключових термінів та виразів.

Варто зазначити, що в такому випадку, отримані значення метрик в середньому будуть значно меншими, аніж при порівнянні способом, запропонованим авторами оригінального методу. Наприклад, якщо раніше ключовий термін reynolds number test розбивався на 3 різних ключових слова: “reynolds”, “number”, “test”, і тоді знайдене окремо ключове слово “reynolds” вважалося успішно знайденим, то тепер при порівнянні “reynolds” з “reynolds number test” отримуємо за метрикою WAcc значення

33,3%, що є нижчим за порогове, а отже, не буде вважатися успішно знайденим. Але це не вплине на результати порівняння запропонованого методу з існуючими аналогами, адже оцінювання буде проводитись за модифікованою метрикою, і отримані значення можна буде оцінити відносно одне одного.

2.8. Висновки до другого розділу

У даному розділі розглянуто запропонований модифікований гібридний метод для автоматизованого пошуку ключових слів та термінів, що складаються з кількох слів.

Описано, які підходи опробовано для покращення кількісних показників точності автоматизованого пошуку ключових слів (зміни в списках неінформативних частин мови та типів зв'язків), і які результати отримано. Описано, чому вирішено покращити метод шляхом додавання можливості пошуку ключових термінів та виразів з кількох слів, обґрунтована важливість ключових термінів та виразів у контексті використання їх в текстовому пошуку. Описано, яким чином використовується інформація про наявні вирази із зв'язками типами flat та compound, отримана за допомогою аналізу, для пошуку ключових термінів з кількох слів в текстах.

Окремо запропоновано модифікацію метрик абсолютної точності автоматизованого пошуку ключових слів та повноти за Жаккаром з використанням додаткової метрики WAcc для отримання кількісного значення точності пошуку ключових термінів. Модифіковані метрики будуть використані для порівняння запропонованого модифікованого методу з існуючими аналогами.

3. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ МОДИФІКОВАНОГО ГІБРИДНОГО МЕТОДУ АВТОМАТИЗОВАНОГО ПОШУКУ КЛЮЧОВИХ СЛІВ ТА ТЕРМІНІВ В ТЕКСТІ

3.1. Обґрунтування вибору засобів реалізації програмного забезпечення

Для демонстрації роботи запропонованого модифікованого методу автоматизованого пошуку ключових слів та термінів в тексті було створено програмну реалізацію у вигляді консольного додатку із можливістю запуску з параметрами командного рядка. Для розробки використовувалися такі технології:

1. В якості мови програмування використано *Python*.
2. Як основний лінгвістичний програмний пакет для більшості операцій над текстами було використано *NLTK*.
3. В якості допоміжного лінгвістичного програмного пакету для деяких деталей реалізації було використано бібліотеку *AllenNLP*, що написана на основі *spracy*.
4. Для обчислення метрик текстових даних використано бібліотеку *JiWER*.

3.1.1. Тип реалізації програмного забезпечення

Рішення про розроблення програмного забезпечення у вигляді консольного додатку з можливостями керування з *CLI* було прийнято з кількох міркувань.

Зазвичай алгоритми пошуку ключових слів використовуються у вигляді частини комплексної пошукової системи. Коротко розглянемо, як працює типова сучасна пошукова система.

Сучасна пошукова система майже одночасно має підтримувати виконання таких процесів:

- *автоматичний Інтернет-серфінг* – так звані вебкраулери, або пошукові роботи, мають перебирати сторінки з мережі Інтернет, обробляти на них вміст і вносити зібрану інформацію у базу даних пошукової системи. Цей процес можна назвати найпростішим з трьох, адже він досить прямолінійний;
- *індексація* – процес створення зв'язків між словами та іншими важливими для пошуку інформації токенами з вебсторінок із їх доменними іменами. Для ефективного пошуку інформації необхідно створити індекси. Для цього проводиться аналіз мови ресурсів, форматів даних, виокремлюються секції документів, збираються мета теги, ключові слова. Ключові слова та фрази є одними із видів даних, що використовується для створення індексів. Правильність відібраних даних для індексації впливає на якість та релевантність результатів пошуку. Для зменшення навантаження на сервери пошукових систем використовують кешування результатів. Зазвичай конкретні інструменти та алгоритми індексації та кешування пошукові системи тримають в секреті, адже це є їх комерційною таємницею;
- *пошук* – запитом користувача в пошуковій системі частіше всього виступають кілька ключових слів або якесь коротке речення. Пошукова система використовує різноманітні алгоритми, такі як текстовий пошук та додаткові параметри пошуку, для отримання релевантних результатів. Після зважування та фільтрації результатів (якщо дозволяється фільтрування за датою або іншими полями), результати видаються користувачу списком від найбільш до найменш релевантних [55].

Якщо проаналізувати, які саме інструменти потрібні для справжньої та сучасної пошукової системи, наприклад Google, то стане очевидним, що

задача автоматизованого пошуку ключових слів в текстах є лише однією із підзадач великої задачі індексації ресурсів.

Отже, як окрема програма із зручним та привабливим користувацьким інтерфейсом, програмне забезпечення для автоматизованого пошуку ключових слів та термінів в текстах не має великої практичної цінності. В цьому випадку консольний додаток, що є майже чистим модулем коду, який можна за необхідності легко інтегрувати як один з модулів великої системи з комплексною архітектурою, є оптимальним вибором.

В контексті проведення наукового дослідження, де швидкість розроблення надає переваги у вигляді швидкого тестування і отримання реальних результатів випробувань для оцінки успішності припущень, консольний додаток підходить якнайбільше, адже не потребує довгого розроблення і його легко виправляти і змінювати.

3.1.2. Мова програмування Python

Python – це високорівнева мова програмування загального призначення, пріоритетом якої за дизайном є читабельність коду. Це динамічно типізована мова, яка підтримує кілька парадигм програмування (структурна, об'єктно-орієнтована, функціональна) і має велику кількість корисних модулів в стандартній бібліотеці. Проста і швидка в опануванні, мова Python стала однією із найпопулярніших мов програмування, і знайшла широке застосування в сферах оброблення великих даних, написання рутинних сценаріїв для автоматизації дій з операційною системою, оброблення текстових даних, створення серверних частин додатків, машинному навчанні, автоматизованому тестуванні тощо.

Охарактеризувати мову програмування Python можна наступним чином:

- *Мова загального призначення.* Python не прив'язаний до якоїсь однієї сфери і відчуває себе добре у різноманітних технологічних доменах: від веброзробки, аналізу даних

машинного навчання та DevOps до автоматизованого тестування, прототипування ПЗ та інших.

- *Інтерпретованість.* Вихідний код програм, написаних на Python, компілюється в перехідний формат, що називається байткодом. Цей байткод запускається на віртуальній машині Python (PVM), яка імітує справжнє комп'ютерне обладнання. Віртуальна машина, в свою чергу, перекладає інструкції байткоду одна за одною в машинний код під час роботи програми. Інтерпретованість має свої переваги (крос-платформеність, автоматичне керування пам'яттю та ін.) і недоліки (низька ефективність виконання), про що йтиметься далі.
- *Об'єктно-орієнтованість.* Python підтримує об'єктно-орієнтоване програмування, що застосовує поняття класів і сутностей. ООП дозволяє краще структурувати код, переводячи моделі з об'єктів реального світу у площину програмування. Варто зазначити, що Python також підтримує і функціональну парадигму програмування.
- *Високорівневість.* Мова Python є високорівневою мовою програмування, це означає, що код більше схожий на команди, які зрозуміє людина, аніж команди для маніпуляції пам'яттю в комп'ютері.
- *Динамічна типізація.* Для оголошення змінних в Python не потрібно вручну вказувати тип, він буде автоматично визначений, виходячи зі значення змінної. Більше того, в одну й ту саму змінну можна перезаписувати значеннями різних типів безліч разів під час виконання коду. Це називається динамічною типізацією. Частина спеціалістів вважає, що динамічна типізація призводить до хаосу в коді і в майбутньому створить проблеми розробникам, якщо не буде точно визначено типи

змінних, інші ж вважають навпаки, статичну типізацію чинником проблем та незручностей.

- *Портативність.* Програми, написані на Python, можна запустити на будь-якій платформі. Це називається WORA (Write Once Run Anywhere), тобто потрібно написати код лише один раз, і можна запускати його будь-де, завдяки віртуальній машині Python. Це значно полегшує роботу, коли потрібно портувати розроблене програмне забезпечення на інші пристрої або платформи, у порівнянні з низькорівневими мовами типу C, C++, де часто необхідно значно переписати код для запуску на інших операційних системах.
- *Простота у вивченні.* Python є доволі простою у вивченні мовою програмування, і новачки легко і швидко її опановують. Цьому сприяє і синтаксис мови, який доволі схожий на звичайні команди дій, що написані англійською.
- *Висока масштабованість.* Python може витримувати велике навантаження на сервіси, що написані цією мовою. Такі відомі та популярні сервіси як Pinterest, Facebook, Instagram написані на Python. Вони щодня отримують мільйони запитів від користувачів по всьому світу і успішно справляються з навантаженням.
- *Велика кількість бібліотек.* Завдяки великій спільноті розробників, можна знайти готовий пакет для вирішення будь-якої задачі, чи то створення простого REST API сервера, проведення аналізу над великою кількістю даних, фінансові або математичні розрахунки, оброблення текстових даних тощо.

Під час вибору мови програмування для програмної реалізації запропонованого модифікованого методу в даній роботі велику роль відігравало існування програмних пакетів для оброблення текстових даних, адже оброблення тексту є досить нетривіальною задачею, особливо коли

необхідно не просто перебирати символи, а використовувати словникові дані, будувати семантичні зв'язки між словами, отримувати та використовувати інформацію про значення, форми слів. Python має дуже потужні бібліотеки типу NLTK або spaCy, а ресурси з інструкціями використання цих пакетів легко знайти, тому було вирішено обрати Python для реалізації програмної частини через простоту, потужність, швидкість та доступність його інструментарію.

3.1.3. Лінгвістичний програмний пакет Python NLTK

Python Natural Language Toolkit (або ж просто NLTK) – відомий і популярний комплекс бібліотек і програмних пакетів для оброблення природних мов, що написаний на мові Python [56]. Платформа надає прості у використанні інтерфейси для більш ніж 50 корпусів і лексичних ресурсів, таких як WordNet, а також набір бібліотек для оброблення тексту для класифікації, токенизації, стемінгу, тегування, синтаксичного аналізу та семантичного міркування, оболонки для індустріальних бібліотек NLP.

Python NLTK надає наступні інструменти для оброблення текстів природних мов та вирішення поширених задач комп'ютерної лінгвістики:

- *Токенізатор*. Процес перетворення послідовності символів на розпізнані групи, або лексеми, для отримання ідентифікованих послідовностей, називається токенизацією. В простих випадках поняття лексема і токен є одним і тим самим, але може бути і випадок більш складного аналізу, коли в токен закладаються різні атрибути, такі як ідентифікатор, оператор, частина мови тощо. Зазвичай в лінгвістичних програмних пакетах токенизатор окремо потрібен для розділення тексту на речення, а речення на слова. Також є частиною в більш складних інструментах, таких як розпізнавачі частин мови або аналізатори зв'язків між словами в реченнях.

- *Лематизатор*. В лінгвістиці лематизація – це процес приведення деякого слова до його основної форми. Особливо актуально в мовах, де є відмінки, наприклад в українській. Форма слова, яку зазвичай знаходимо в словнику, і є лемою. Часто застосовується, коли необхідно отримати статистику використання слова у тексті, але немає потреби вважати кожную форму слова окремими словами.
- *Аналізатор частин мови*. Розмічування частин мови (POS tagging) – це процес визначення частини мови для слова в реченні або тексті. що ґрунтується на його визначенні і на контексті. Цей інструмент не такий простий, як може здатися на перший погляд, адже є такі слова, які в залежності від позиції в тексті можуть виступати різними частинами мови. Отже, неможливо просто обійтися списком “слово – частина мови”, необхідно додатково аналізувати контекст, зв’язки слова із суміжними та інші деталі.
- *Аналізатор залежностей*. Аналіз залежностей в реченні – це процес аналізу граматичної структури для визначення пов’язаних між собою слів та типів зв’язків між ними. За допомогою отриманих таким аналізатором даних можлива побудова синтаксичної розмітки, що є корисним в задачах автоматизованого пошуку фактів в текстах, пошуку ключових слів, аналізу змісту, тощо.

Python NLTK має ще більше інструментів, які можуть бути корисними в конкретних задачах; вище було перелічено лише найосновніші, які використовуються найчастіше. NLTK вважається потужним та популярним інструментом для оброблення природномовних текстів на Python, через що досить легко знайти детальні інструкції для новачків про вирішення тої чи іншої лінгвістичної проблеми за допомогою цього пакету що підходить для досліджень. Отже, було вирішено обрати його для створення програмної

реалізації запропонованої модифікації методу через велику кількість реалізованих алгоритмів, інструментів, та простоту вивчення і використання.

3.1.4. Бібліотека AllenNLP на основі spaCy

AllenNLP є лінгвістичною бібліотекою, що прискорює дослідження NLP та надає новітні та передові інструменти для вирішення задач комп'ютерного оброблення природних мов [57]. Бібліотека побудована на основі іншої популярної бібліотеки spaCy.

spaCy є безкоштовною бібліотекою для потужного оброблення природних мов з відкритим кодом для Python [58]. Вона розроблена спеціально для використання в реальних застосунках та сервісах із фокусом на продуктивність та оброблення великих масивів даних. За допомогою неї можна проводити видобуток інформації з текстів або створювати системи розуміння природних мов.

Розробники наводять такі сильні сторони бібліотеки:

- Підтримка 73+ мов.
- 84 натренованих pipelines для 25 мов.
- Багатозадачне навчання за допомогою попередньо підготовлених трансформаторів, таких як BERT.
- Попередньо підготовлені вектори слів.
- Швидкість, забезпечена найновітнішими технологіями.
- Готова до використання в реальних проєктах система навчання.
- Лінгвістично мотивована токенизація.
- Компоненти для розпізнавання іменованих сутностей, тегування частин мови, аналізу залежностей, сегментації речень, класифікації тексту, лематизації, морфологічного аналізу, зв'язування сутностей тощо.
- Легке розширення за допомогою спеціальних компонентів і атрибутів.

- Підтримка користувацьких моделей у PyTorch, TensorFlow та інших фреймворках.
- Вбудовані візуалізатори для синтаксису та іменованих сутностей.
- Просте пакування моделі, розгортання та керування робочим процесом

spraCy одразу позиціонує себе як бібліотеку не для дослідників та досліджень, а як інструмент для промислового використання, тож розробники мають розуміти, що вони роблять і для чого, аби ефективно і за призначенням використовувати бібліотеку. Однією із ключових особливостей бібліотеки є *tailored pipelines*, або ж досконало підлаштовані виробничі потоки, які дозволяють максимізувати ефективність бібліотеки для конкретної задачі користувача.

AllenNLP було обрано як допоміжний пакет через кращі результати ніж у NLTK в деяких випадках, таких як визначення кореференційних зв'язків, про що йтиметься далі.

3.1.5. Бібліотека JiWER

JiWER є простою та швидкою бібліотекою для Python для оцінювання і тестування систем автоматичного розпізнавання мовлення або машинного перекладу [59]. Підтримує визначення таких метрик:

- Word error rate (WER);
- Match error rate (MER);
- Word information lost (WIL);
- Word information preserved (WIP);
- Character error rate (CER).

JiWER було використано для визначення значень метрик WER для автоматично знайдених ключових термінів з кількох слів. На основі цих значень необов'язковий модуль обчислює значення абсолютної точності та

повноти пошуку ключових слів за Жаккаром, якщо користувач надає шлях до файлу з еталонними ключовими термінами до тексту.

3.2. Архітектура розробленого програмного забезпечення

Програмне забезпечення, яке реалізує запропонований у роботі модифікований метод для автоматизованого пошуку ключових слів та термінів в тексті, виконано у вигляді консольного додатку на мові Python з підтримкою CLI і має модульну архітектуру (рис. 3.1).

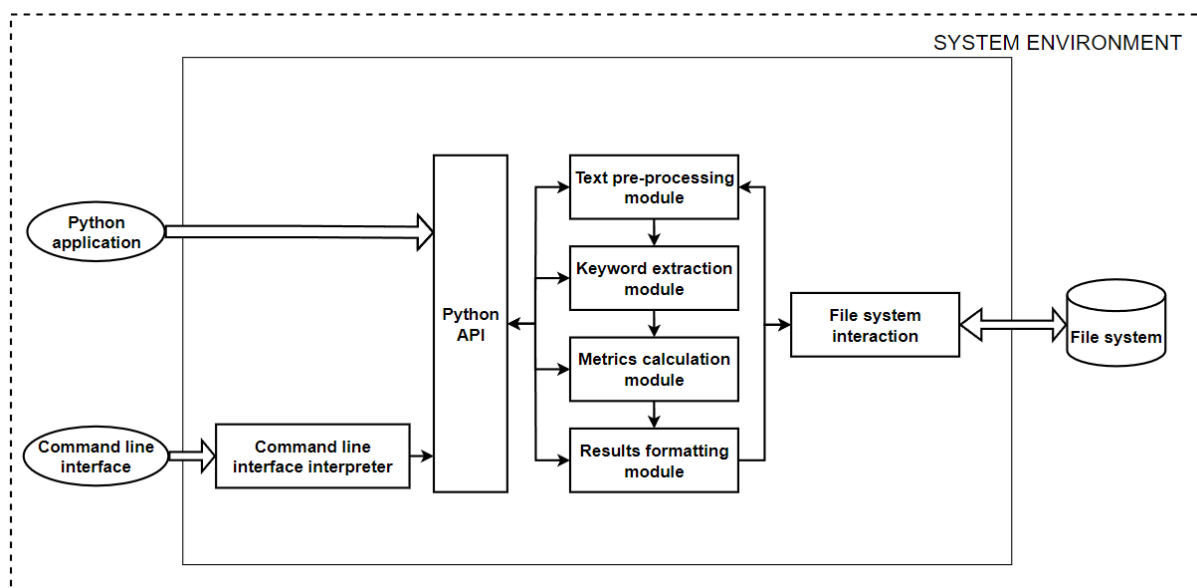


Рис. 3.1. Архітектура розробленого програмного забезпечення

В програмній реалізації можна виділити такі модулі:

1. *Python API*. Надає можливості застосування вбудованих модулів Python, базові можливості роботи з даними, можливості підключення додаткових пакетів, таких як NLTK та AllenNLP.
2. *Модуль роботи з командним рядком (Command line interface interpreter)*. Дозволяє запускати роботу консольного додатку за допомогою командного рядка. Відповідає за інтерпретацію введеної користувачем команди в командному рядку і розбір аргументів та їх значень. Так, шлях до вхідного та вихідного

файлів вказується як аргументи командного рядка. Після інтерпретації відповідає також за правильну ініціалізацію та запуск програми з коректними параметрами.

3. *Модуль для роботи з файловою системою (File system interaction).* Відповідає за зчитування файлів із вихідними текстовими даними та запис результатів роботи алгоритму у вигляді текстових файлів звітів для подальшого збереження у файловій системі користувача. Якщо користувач вказує шлях до файлу з еталонними ключовими термінами до тексту, якщо такі відомо наперед, зчитує еталонні ключові слова з файлу для подальшого обчислення метрик.
4. *Модуль попереднього оброблення вхідних текстів (Text pre-processing module).* Відповідає за оброблення вхідних текстових даних, а саме прибирання Non-ASCII символів, які можуть “зламати” парсер, виправлення очевидних помилок пунктуації (наприклад, дві коми, що йдуть підряд будуть замінені на одну кому і таке інше).
5. *Модуль пошуку ключових слів та термінів (Keyword extraction module).* Отримує оброблені текстові дані з модулю попереднього оброблення та виконує пошук ключових слів та термінів. Використовує Python NLTK для виконання етапів алгоритму та формує результат у вигляді списків ключових слів та термінів. Для збереження результатів у вигляді текстових звітів використовує модулі форматування результатів та роботи з файловою системою.
6. *Модуль обчислення метрик точності та повноти (Metrics calculation module).* У випадку, якщо користувачем було вказано файл з переліком еталонних ключових термінів, модуль порівнює отримані в результаті роботи методи ключові терміни з еталонними на збіг, і обчислить значення абсолютної точності

та повноти пошуку ключових слів за Жаккаром. Модуль дозволяє оцінити точність роботи методу, що було використано при порівнянні запропонованого в роботі модифікованого методу з існуючими аналогами.

7. *Модуль форматування результатів (Results formatting module).* За допомогою отриманих в результаті пошуку ключових слів та термінів генерує файли звітів для подальшого запису у файлову систему за допомогою відповідного модуля.

В свою чергу модуль пошуку ключових слів та термінів складається з кількох підмодулів. Кожен підмодуль реалізує свій етап в процесі пошуку ключових слів. Коротко їх опишемо:

1. *Модуль отримання пар слів зі зв'язками.* Використовує можливості програмного пакету Python NLTK для аналізу тексту та отримання зв'язків між словами, формуючи список пар слів.
2. *Модуль пошуку багатослівних виразів.* На основі знайдених модулем отримання пар слів зі зв'язками зв'язків шукає і складає багатослівні вирази, що містяться в тексті.
3. *Модуль фільтрації пар слів з неінформативними типами зв'язків.* Фільтрує пари слів, які містять типи зв'язків, що визначені як неінформативні.
4. *Модуль заміни займенників на відповідні іменники.* Проводить кореференційний аналіз тексту, і на основі отриманої інформації замінює займенники відповідними іменниками.
5. *Модуль фільтрації слів з неінформативних частин мови.* Видаляє з подальшого розгляду слова, які відносяться до неінформативних частин мови.
6. *Модуль фільтрації стоп-слів.* Видаляє зі слів-кандидатів ті, що є “шумними” або стоп-словами.

7. *Модуль зважування слів-кандидатів та отримання ключових слів.* Обчислює кількість зв'язків для всіх слів, що лишилися на даному етапі і відбирає лише потрібну кількість. Також обробляє випадки колізії.
8. *Модуль відбору ключових виразів.* На основі відібраних ключових слів відбирає багатослівні вирази до списку ключових.

На рисунку 3.2 зображено структурну схему модуля пошуку ключових слів та термінів.

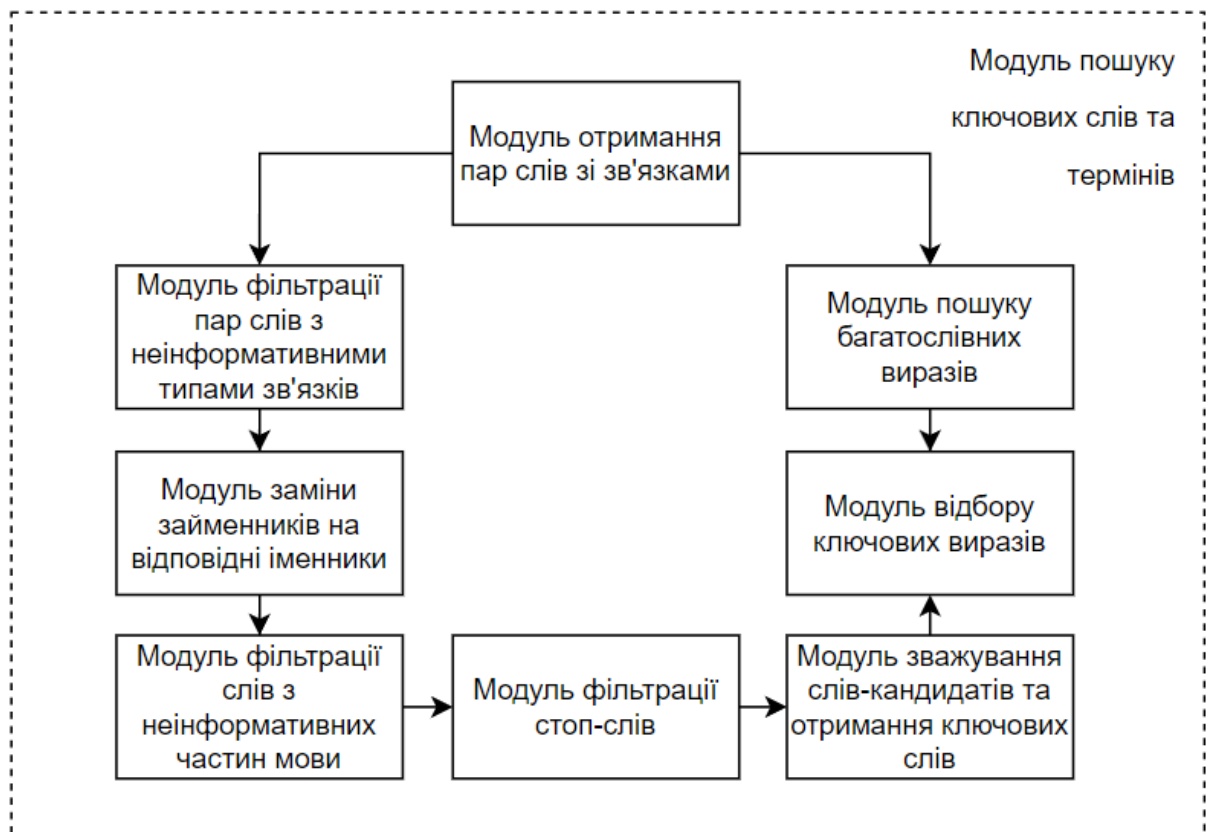


Рис. 3.2. Структурна схема модуля пошуку ключових слів та термінів

3.3. Особливості реалізації програмного забезпечення

Під час реалізації методу у вигляді програмного забезпечення було прийнято кілька рішень, які відрізняють цю реалізацію від тої, що була запропонована розробниками оригінального методу при реалізації їх версії програмного забезпечення. Ці рішення було прийнято для покращення

можливостей роботи ПЗ, або з інших практичних міркувань, про які буде йти мова в цьому підрозділі.

3.3.1. Використання Python NLTK разом із Stanford Dependency Parser для виконання базових лінгвістичних операцій над текстовими даними

Розробники оригінального методу вирішили використовувати мову програмування Java разом із DK Pro Core для створення програмної реалізації методу. Таке рішення було пояснено насамперед тим, що розробники вже мали досвід зі створення додатків на Java.

Дана реалізація, в свою чергу, створена за допомогою Python NLTK, через переваги цієї технології, що описані в розділі 3.1.

Розглянемо, які інструменти були використані для виконання базових операцій над текстами в сфері комп'ютерної лінгвістики.

Для пошуку зв'язків між парами лем в тексті рекомендується розглядати кожне речення окремо. Для цього необхідно розбити текст на речення, для чого використовується функція `sent_tokenize` з модуля `nltk.tokenize`.

Після того, як текст було розбито на речення, необхідно проаналізувати кожне і отримати список всіх зв'язків, або залежностей (dependency) між парами слів. Для цього застосовується модуль `StanfordDependencyParser`, а саме його метод `raw_parse`, який приймає на вхід речення в строковому вигляді, і на вихід віддає складну структуру-дерево з усіма зв'язками між парами слів.

Насправді, ця структура досить непростя для оброблення. Наприклад, для речення “The Software Product Line approach offers benefits such as savings, large-scale productivity and increased product quality to the software development because it is based on software architecture reuse, which is more planned and aimed to a specific domain” результат роботи методу `raw_parse` буде виглядати наступним чином (див. Додаток 1. Результат роботи методу

raw_parse). Його не просто обходити і він містить багато зайвих даних, коли для реалізації методу потрібно знати лише інформацію про тип зв'язку та тег частин мови слів, що використовуються в цих зв'язках.

На Інтернет-ресурсах, присвячених роботі з Python NLTK, рекомендують застосовувати метод *.triples* для отримання з об'єкту-дерева корисної інформації для аналізу. Після застосування методу отримуємо читабельний список кортежів, який вже легко сприймати та обробляти (див. Додаток 1. Застосування *.triples* до результату роботи *raw_parse*).

Але в цього перетворення є суттєвий недолік: було втрачено адреси слів в тексті. Одним із кроків методу є заміна займенників на відповідні іменники, причому для цього оброблюється одразу весь текст, адже аналізатор дивиться на використання одного й того ж самого слова та його займенників протягом кількох речень тексту. Після цього потрібно підставити знайдені іменники у пару лем (залежність), але це неможливо зробити, не маючи точних адрес слів в тексті. Для вирішення цієї проблеми було реалізовані власні процедури обходу структури-дерева із даними про залежності, які отримуємо після роботи методу *.raw_parse*.

Спочатку рекурсивна функція *traverse_and_write_to_list* перебирає вузли структури-дерева, витягує лише колонку з даними про *deps* і записує всі сутності в список для зручної ітерації (Лістинг 3.1).

Лістинг 3.1. Перебір вузлів структури-дерева залежностей і витягання потрібної інформації в список

```
def traverse_and_write_to_list(deps, addr, list):
    dep = deps.get_by_address(addr)
    list.append(dep)
    for d in dep['deps']:
        for addr2 in dep['deps'][d]:
            traverse_and_write_to_list(deps, addr2, list)
```

Після цього на вхід функції *get_prases_from_deps* подається цей список часток структури-дерева і після її відпрацювання отримується

сформований список сутностей типу `Phrase`, при цьому відслідковуючи абсолютну позицію (адресу) слова в тексті (Лістинг 3.2).

Лістинг 3.2. Формування структури `Phrase`

```
def get_phrases_from_deps(deps, last_index, lemmatized_words):
    phrases = []
    for dep in deps:
        if dep['word'] is not None:
            main_dep = None
            for i_dep in deps:
                if (i_dep['address'] == dep['head']) and (i_dep['word']
is not None):
                    main_dep = i_dep
                    break
            if main_dep is not None:
                new_phrase = {"main_word": {"word": main_dep['word'],
"address": main_dep['address'] + last_index, "pos": main_dep['tag'],
"lemma": lemmatized_words[main_dep['address'] + last_index - 1]},
"dependent_word": {"word": dep['word'], "address": dep['address'] +
last_index, "pos": dep['tag'], "lemma": lemmatized_words[dep['address'] +
last_index - 1]}, "dep_type": dep['rel']}
                phrases.append(new_phrase)
    return phrases
```

Кожна `Phrase` містить дані про тип зв'язку, головне і залежне слово, їх абсолютні адреси в тексті, теги частин мови, до яких слова відносяться, та основні форми слів, для подальшого обчислення кількості зв'язків та зважування.

Для отримання більш точних результатів за алгоритмом необхідно приводити слова до основної форми перед визначенням кількості зв'язків. Це прибере похибку результатів, у випадку, якщо одна й та сама сутність або поняття малися на увазі в різних частинах тексту, і вживалися в різних формах, через що будуть розцінені як різні слова при обчисленні кількості зв'язків і зважуванні, що може значно знизити вагу поняття в кінцевому випадку. Це дуже актуально для мов, де є відмінки, наприклад, української, але в англійській також корисне, наприклад, коли слово вживається в однині і в множині (knife та knives).

Для приведення слів до основної форми використовується модуль *WordNetLemmatizer*, що імпортується з *nlk.stem*. Можна застосувати його метод *lemmatize*, який, однак, потребує ще й вказану частину мови, адже слово, яке пишеться однаково в різних випадках, та виступаючи різними частинами мови, буде мати різні основні форми в залежності від контексту вживання. Для цього використовується метод *pos_tag*, що дозволяє при надаванні тексту отримати для кожного слова інформацію про те, який тег частини мови має кожне слово з цього тексту. До речі, інформація про кожне слово, отримана після виконання даного методу, пізніше використовується на етапі фільтрування отриманих кандидатів у ключові слова за інформативними частинами мови.

Для фільтрації стоп-слів використовується словник стоп-слів, що надається модулем *stopwords* із набору *nlk.corpus*. В нашому випадку, це словник стоп-слів для англійської мови.

3.3.2. Застосування AllenNLP для більш точної заміни займенників на відповідні іменники

Як вже було зазначено раніше в даній роботі, одним із кроків алгоритму є заміна займенників на відповідні іменники до обчислення кількості зв'язків для кожного слова-кандидата. Це дозволяє уникнути ситуацій, коли вага частовживаного поняття буде занижена через те, що воно представлено відповідними займенниками упродовж тексту.

Для цього застосовується аналіз кореференційних зв'язків. Синтаксичний аналізатор намагається для кожної сутності, яка вживається в тексті кілька разів у різних формах, знайти всі такі позиції і згрупувати їх.

На тематичних Інтернет-ресурсах було знайдено два різні інструменти для аналізу кореференції: *Stanfordcorenlp* та *AllenNLP*. Разом з *Python NLTK* частіше всього застосовують саме першу бібліотеку. Але випробування показали, що при застосуванні *AllenNLP* для цієї операції можна отримати більш точні результати.

Наведемо приклад аналізу кореференційних зв'язків для такого короткого тексту: “Born in Honolulu, Hawaii, Obama is a graduate of Columbia University and Harvard Law School, where he was president of the Harvard Law Review. He was a community organizer in Chicago before earning his law degree.” При застосуванні інструменту StanfordCoreNLP було отримано наступний результат:

Лістинг 3.3. Результат аналізу кореференційних зв'язків за допомогою StanfordCoreNLP

```
('14', [  
  
  {'id': 8, 'text': 'he', 'type': 'PRONOMINAL', 'number': 'SINGULAR',  
    'gender': 'MALE', 'animacy': 'ANIMATE', 'startIndex': 20, 'endIndex': 21,  
    'headIndex': 20, 'sentNum': 1, 'position': [1, 9],  
    'isRepresentativeMention': True},  
  
  {'id': 12, 'text': 'He', 'type': 'PRONOMINAL', 'number': 'SINGULAR',  
    'gender': 'MALE', 'animacy': 'ANIMATE', 'startIndex': 1, 'endIndex': 2,  
    'headIndex': 1, 'sentNum': 2, 'position': [2, 3],  
    'isRepresentativeMention': False},  
  
  {'id': 14, 'text': 'his', 'type': 'PRONOMINAL', 'number': 'SINGULAR',  
    'gender': 'MALE', 'animacy': 'ANIMATE', 'startIndex': 10, 'endIndex': 11,  
    'headIndex': 10, 'sentNum': 2, 'position': [2, 5],  
    'isRepresentativeMention': False}])
```

Як видно з результату, аналізатор знайшов одну сутність, яка представлена в тексті словами “he”, “He”, “his”, але не зміг пов’язати зі словом “Obama”, до якого насправді і відносяться займенники.

При обробці цих самих двох речень за допомогою Allennlp отримуємо наступний результат:

Лістинг 3.4. Результат аналізу кореференційних зв'язків за допомогою Allennlp

```
[[6, 6], [19, 19], [28, 28], [37, 37]]
```

Отримані індекси відповідають адресам слів “Obama”, “he”, “He”, “his” в тексті. Як бачимо, Allennlp зміг отримати точніший результат – тепер

при заміні займенників буде замінено всі наступні займенники, що представляють сутність “Obama”, відповідним іменником. Це збільшить вагу слова “Obama” при обчисленні ваг при зважуванні слів-кандидатів у ключові слова. Для даного короткого тексту слово “Obama” дійсно є ключовим.

3.3.3. Застосування бібліотеки JiWER для обчислення метрик абсолютної точності та повноти за Жаккаром

Для того, аби зручно оцінювати, наскільки точно метод шукає ключові слова та терміни, в даній магістерській роботі було розроблено модуль для обчислення значень абсолютної точності та повноти пошуку за Жаккаром для ключових слів, отриманих в результаті роботи програмного забезпечення. Алгоритм проходить списком еталонних ключових термінів і перевіряє, чи є серед автоматично знайдених програмою ключових термінів такі, що мають значення WER, менше за порогове. Для цього з бібліотеки JiWER використано метод `wer`, який приймає першим аргументом “довідкове” значення (назвемо його еталонним), і другим – значення, яке було знайдено автоматично (лістинг 3.5).

Лістинг 3.5. Використання методу `wer` з бібліотеки JiWER

```
is_matched = wer(ref_keyword, found_keyword) < 1/3 + 0.01:
```

Після того, як всі вирази було перевірено, використовуючи кількість правильно знайдених термінів, додаток обчислює значення абсолютної точності та повноти за Жаккаром для результатів поточного запуску додатку. Значення будуть записані у файл з результатами пошуку.

3.4. Приклад використання розробленого програмного забезпечення

Розроблене програмне забезпечення для автоматизованого пошуку ключових слів та термінів в тексті передбачає взаємодію за допомогою інтерфейсу командного рядка та використання в програмному коді в якості бібліотеки.

3.4.1. Взаємодія через інтерфейс командного рядка

Сценарії використання додатку користувачем за допомогою CLI зображено на UML діаграмі (рис. 3.3).

Користувач може:

- підготувати файл з текстом для пошуку ключових слів;
- переглянути можливі аргументи командного рядка;
- вказати шлях до файлу з вхідним текстом;
- вказати бажаний шлях до файлу з результатами пошуку, який буде створено;
- за необхідності вказати кількість бажаних однослівних ключових слів;
- шлях до файлу з еталонними (визначеними автором тексту) ключовими словами та виразами для отримання кількісних характеристик точності та повноти результату.

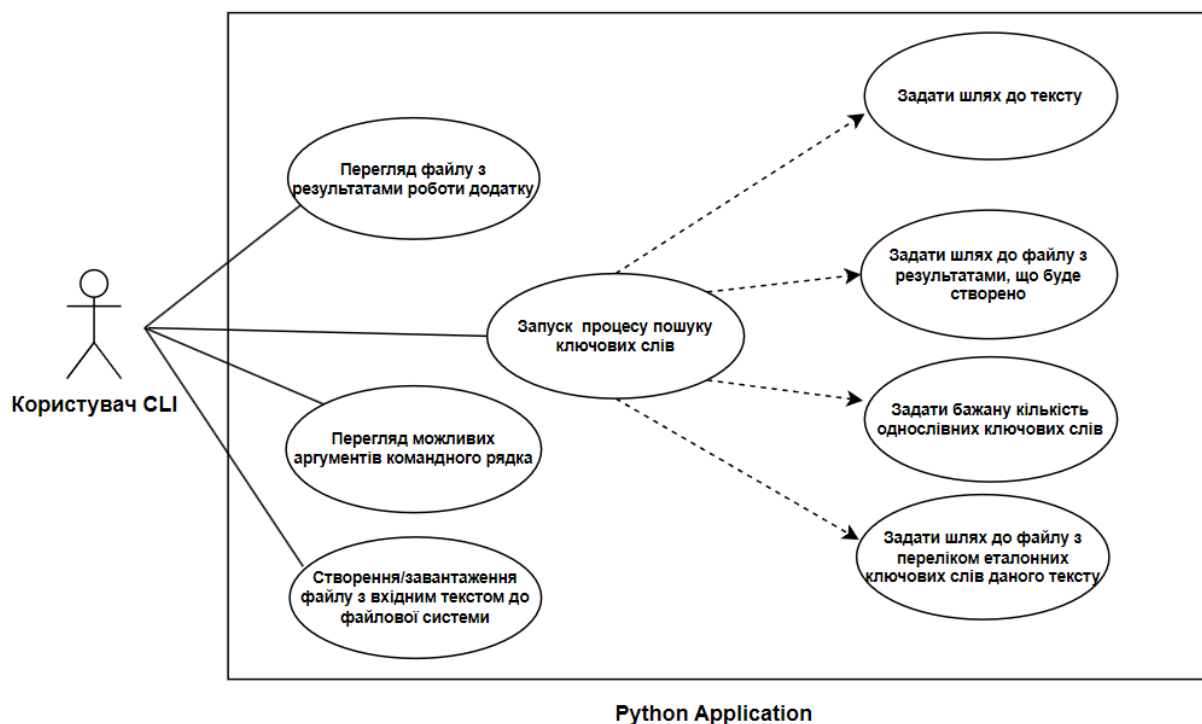


Рис. 3.3. UML діаграма сценаріїв використання розробленого програмного забезпечення для автоматизованого пошуку ключових слів та термінів в тексті користувачем

Для взаємодії з додатком за допомогою CLI потрібно, щоб в системі користувача було встановлено Python версії 3.8 (рис. 3.4).

```
(venv) PS C:\Users\3A43Mka\Documents\diplomma-testing> python --version
Python 3.8.10
```

Рис. 3.4. Перевірка версії Python в командному рядку

Для запуску додатку необхідно викликати виконуваний python файл і вказати обов'язкові аргументи командного рядка, а за необхідністю – і додаткові (рис. 3.5).

```
(venv) PS C:\Users\3A43Mka\Documents\diplomma-testing> python main.py -i input.txt -o out.txt -n 15 -r .\reference.txt
```

Рис. 3.5. Приклад запуску додатку за допомогою CLI

Перелік та опис можливих аргументів командного рядка наведено у табл. 3.1.

Таблиця 3.1

Можливі аргументи командного рядка

Аргумент CLI	Допустимі значення	Опис
-i	Шлях до файлу у файловій системі	Файл, з якого буде зчитано вхідний текст. <i>Обов'язковий параметр</i>
-o	Шлях до файлу, що буде створено, у файловій системі	Файл з результатами, який буде створено після аналізу тексту. <i>Обов'язковий параметр</i>
-n	Додатне ціле число	Кількість бажаних окремих ключових слів в результаті
-r	Шлях до файлу у файловій системі	Файл з переліком еталонних ключових слів, якщо такі є. Використовується для перевірки точності роботи методу для тексту з наперед визначеними ключовими словами

Файл з переліком еталонних ключових слів має містити список ключових слів або термінів, розділених комою та пробілом (рис. 3.6).

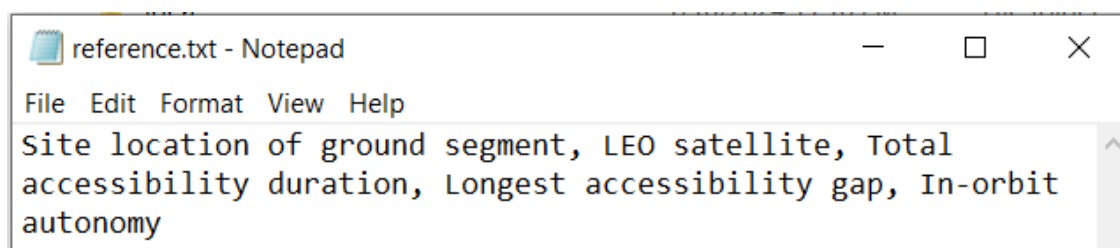


Рис. 3.6. Приклад вмісту файлу з переліком еталонних ключових слів

Після виконання додатку буде створено файл з результатами, який буде форматовано наступним чином (рис. 3.7):

- у першому рядку йдуть знайдені ключові слова, що складаються з одного слова;
- у другому рядку йдуть знайдені ключові терміни з кількох слів;
- якщо було вказано файл з еталонними ключовими словами, на їх основі буде обчислено абсолютну точність та повноту пошуку ключових слів за Жаккардом для цього запуску, ці значення будуть записані у двох останніх рядках файлу.

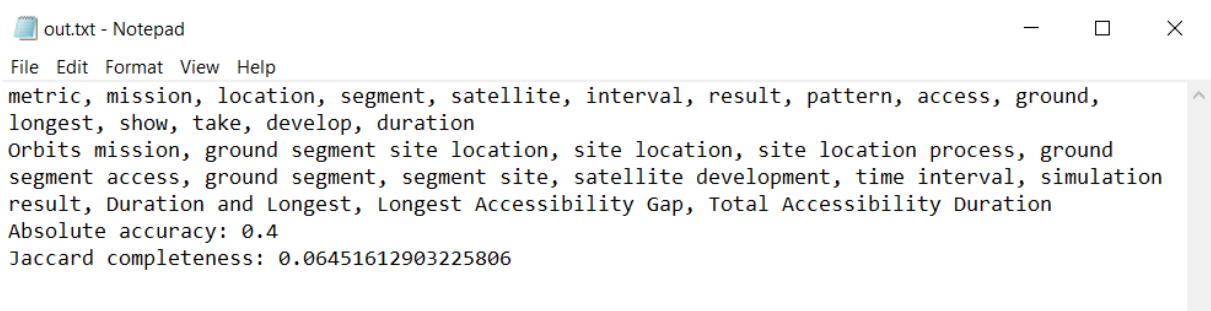


Рис. 3.7. Приклад файлу результатів

3.4.2. Використання в програмному коді в якості бібліотеки

Як вже було зазначено, реалізований метод може бути корисним як частина якоїсь комплексної пошукової системи. Для цього зручно мати розроблений модуль у вигляді бібліотеки, яку зручно використовувати в програмному коді.

При використанні розробленого пз як бібліотеки, розробнику надаються всі ті ж функціональні можливості, як і при використанні за допомогою CLI. Потрібно імпортувати функцію `get_keywords` і викликати її в коді в потрібному місці. В аргументи функції потрібно передати параметри, аналогічно до того, як задаються аргументи при роботі за допомогою CLI (рис. 3.8).

```

from keyword_lib import get_keywords

text = "A conceptual control model for the Reynolds number test based on isentropic r
num_of_keywords = 11
reference = "Blowdown wind tunnel, Pressure control, Mach number control, Reynolds nu

result = get_keywords(text, num_of_keywords, reference)

print("Library returns an object with such attributes:")
print(result['one_word_keywords'])
print(result['multi_word_keywords'])
print(result['absolute_accuracy'])
print(result['jaccard_completeness'])

```

Рис. 3.8. Приклад використання бібліотеки в програмному коді

3.5. Висновки до третього розділу

У даному розділі обґрунтовано та описано використані засоби реалізації програмного забезпечення:

1. Програмне забезпечення реалізовано у вигляді консольного додатку із можливістю запуску з параметрами командного рядка.
2. Мова програмування Python.
3. Лінгвістичний програмний пакет Python NLTK.
4. Допоміжна лінгвістична бібліотека AllenNLP.
5. Бібліотека для обчислення метрик JiWER.

Наведено схему модулів розробленого програмного забезпечення, описано принцип їх роботи та взаємодію.

Особливостями розробленої програмної реалізації запропонованого методу є:

1. Використання Python NLTK разом із Stanford Dependency Parser для виконання базових лінгвістичних операцій над текстовими даними.
2. Застосування AllenNLP для більш точної заміни займенників на відповідні іменники.

3. Застосування бібліотеки JiWER для обчислення метрик абсолютної точності та повноти за Жаккаром

Розглянуто приклади використання розробленого програмного забезпечення за допомогою командного рядка та у вигляді бібліотеки в програмному коді.

4. АНАЛІЗ ЕФЕКТИВНОСТІ ЗАСТОСУВАННЯ МОДИФІКОВАНОГО МЕТОДУ

4.1. Критерії оцінювання та аналіз ефективності запропонованого методу

З огляду на постановку задачі, а саме модифікацію методу з метою уможливлення отримання в результаті аналізу не тільки ключових слів, а й багатослівних ключових термінів та виразів, що може давати більш точні результати, ніж існуючі аналоги, час виконання не має такого вирішального значення, як точність отриманих результатів, отже оцінювання ефективності розробленого програмного забезпечення, що реалізує запропоновану модифікацію гібридного методу автоматизованого пошуку ключових слів та термінів у тексті проводилося за двома метриками:

1. Абсолютна точність пошуку ключових слів.
2. Повнота пошуку ключових слів за Жаккаром.

Порівняння розробленого програмного забезпечення проводилися з наступними існуючими сервісами, що надають подібні можливості з пошуку ключових слів у тексті:

1. MonkeyLearn на основі машинного навчання.
2. WordCount, що використовує аналіз N-грам.
3. Komprehend, що побудований на статистичному аналізі.

4.1.1. Абсолютна точність пошуку ключових слів

Одним із критеріїв для вимірювання кількісних характеристик ефективності розробленого методу було обрано метрику абсолютної точності. Вона виражається відношенням кількості правильно автоматично знайдених ключових термінів до загальної кількості ключових слів (формула 2.1), тобто, в нашому випадку, це ключові слова, що були задані авторами тексту.

Для випробувань було взято 200 довільних текстів тез до статей з наукового технічного журналу [60]. Це невеликі тексти обсягом 150-400 слів, із наперед зазначеним переліком ключових слів, тож їх зручно використовувати для швидкого тестування.

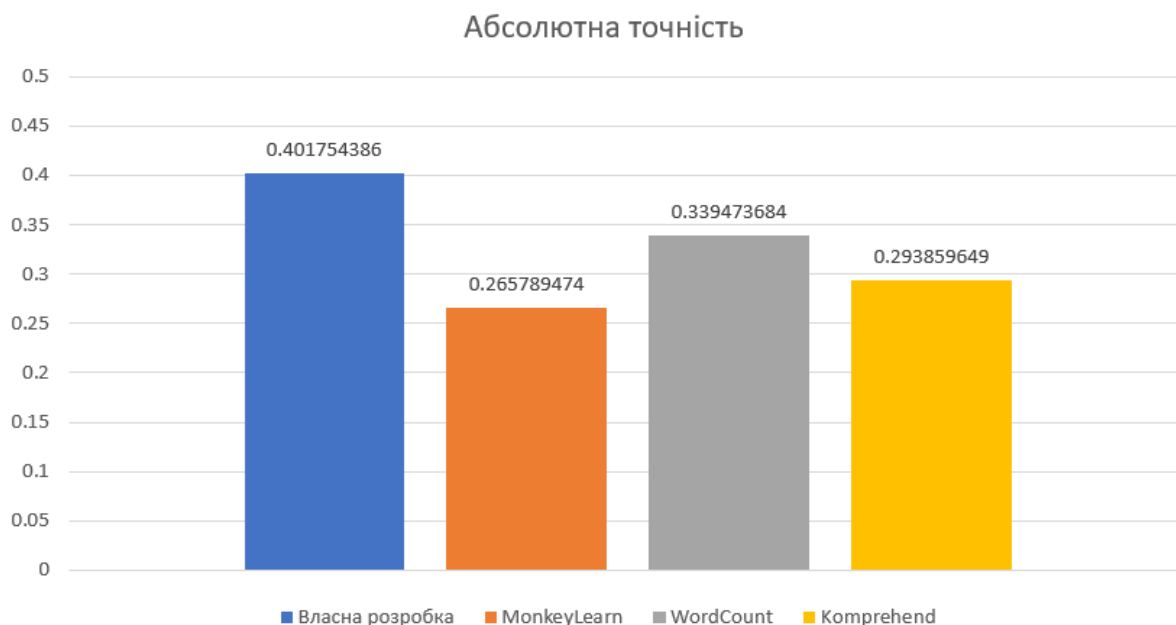


Рис. 4.7. Гістограма порівняння значень абсолютної точності пошуку ключових слів для текстів обсягом 150-400 слів

Результати випробувань (рис. 4.7) демонструють, що середнє значення абсолютної точності пошуку ключових слів для власної розробки становить 0,402, для сервісу MonkeyLearn – 0,266, WordCount – 0.34, Komprehend – 0.294. Отже, власна розробка збільшує абсолютну точність пошуку ключових слів у межах від 6.2% до 13,6% у порівнянні з аналогами в текстах обсягом 150-400 слів.

Додатково були проведені випробування на текстах більших розмірів, близько 1500-2500 слів. Для цього було взято 100 довільних текстів статей з того ж наукового журналу.

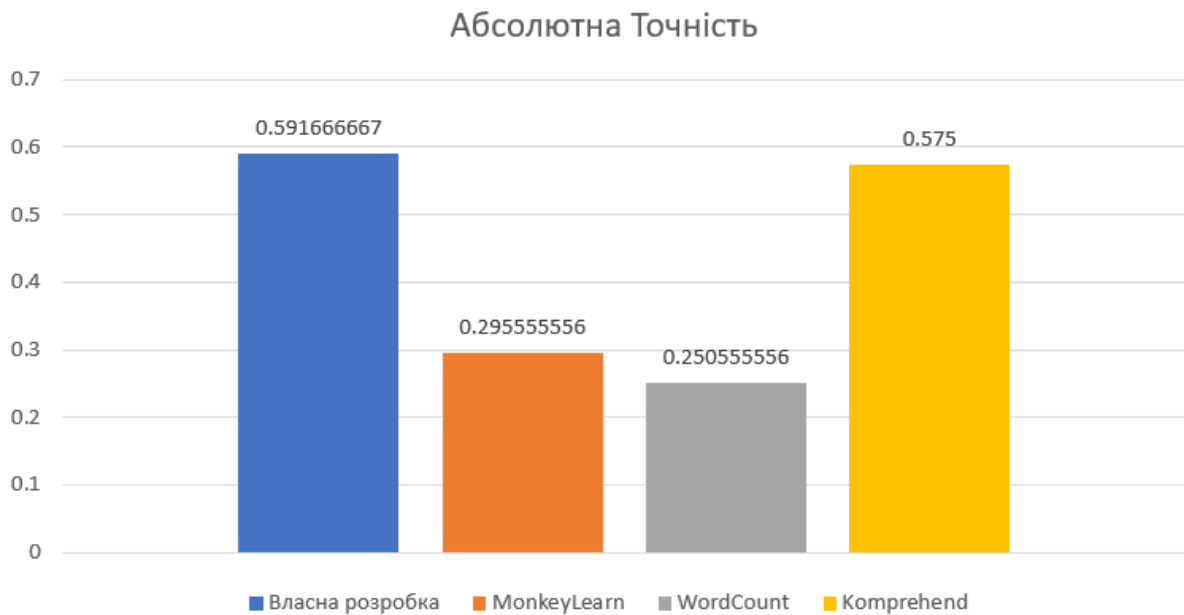


Рис. 4.8. Гістограма порівняння значень абсолютної точності пошуку ключових слів для текстів обсягом 1500-2500 слів

Результати випробувань (рис. 4.8) демонструють, що середнє значення абсолютної точності пошуку ключових слів для власної розробки становить 0,592, для сервісу MonkeyLearn – 0,296, WordCount – 0.251, Komprehend – 0.575. Отже, власна розробка збільшує абсолютну точність пошуку ключових слів у межах від 1,7% до 34,1% у порівнянні з аналогами в текстах обсягом 1500-2500 слів.

За результатами отриманих значень абсолютної точності пошуку ключових слів можна відмітити наступне:

1. На текстах невеликого обсягу (150-400 слів) запропонований модифікований метод має явну перевагу над іншими аналогами.
2. На текстах більшого обсягу (1500-2500 слів) запропонований метод все ще має найкращі результати, однак сервіс Komprehend, що базується на методах статистики, майже його наздоганяє.
3. Збільшення середньої точності майже всіх 4 інструментів зі збільшенням обсягу тексту можна пояснити більш явним проявом статистичних закономірностей. Таким чином, за

достатньо великого обсягу тексту, ключові вирази будуть неодноразово повторюватися, що збільшить їх вагу при зважуванні кандидатів у ключові терміни.

На основі отриманих результатів порівнянь можна стверджувати, що запропонований модифікований метод є кращим за критерієм абсолютної точності. Гіпотеза про використання інформації, отриманої з синтаксичного аналізатора, для пошуку багатослівних виразів в тексті надає модифікованому методу можливість шукати ключові терміни, які складаються з кількох слів, що і було очікуваним. Причому зі збільшенням обсягу тексту збільшується і кількість правильно знайдених ключових термінів, що можна пояснити більш частим входженням ключового терміну в текст. Виявлення ключових термінів з кількох слів було неможливим для методу, взятого за основу модифікації.

4.1.2. Повнота пошуку ключових слів за Жаккаром

Іншим критерієм для оцінювання ефективності методів було обрано повноту пошуку ключових слів за Жаккаром. Значення цієї метрики виражається відношенням кількості правильно знайдених ключових слів автоматичною системою до кількості елементів об'єднання множини всіх знайдених автоматично ключових слів та множини еталонних ключових слів (формула 2.2). На відміну від метрики абсолютної точності, повнота пошуку дозволяє оцінити ефективність пошуку відносно вербального шуму у результатах.

Випробування проводилися на тих самих 200 довільно обраних текстах тез статей з наукового технічного журналу [60].

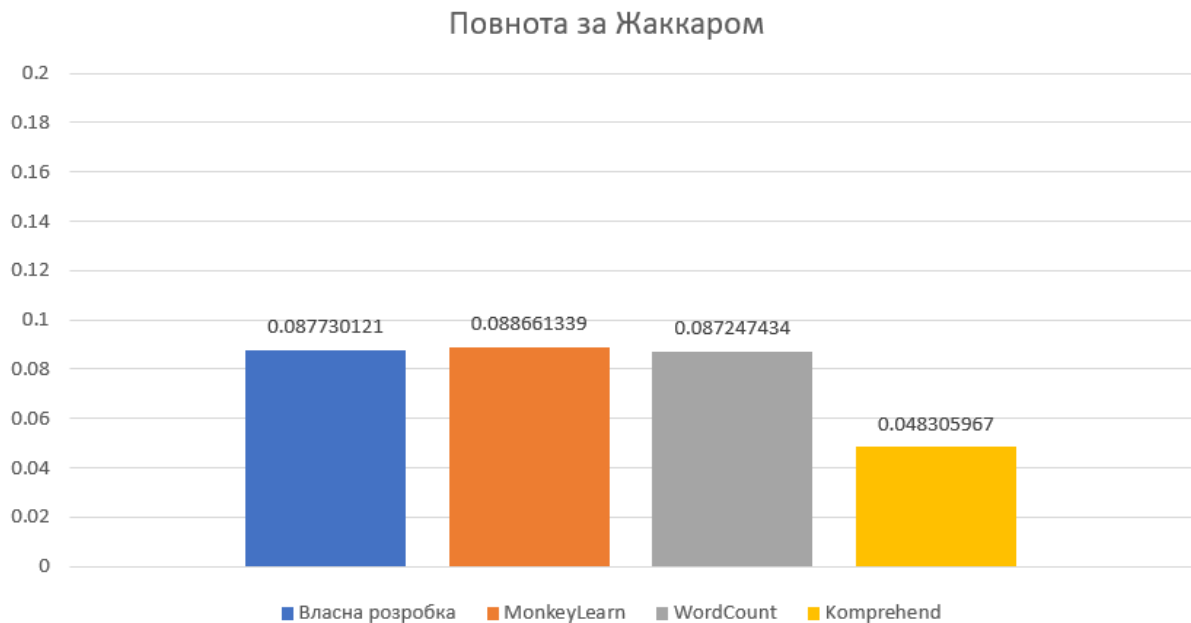


Рис. 4.9. Гістограма порівняння значень повноти пошуку ключових слів за Жаккаром для текстів обсягом 150-400 слів

Результати отриманих значень повноти пошуку ключових слів за Жаккаром для запропонованої модифікації та існуючих аналогів (рис. 4.9) демонструють, що середнє значення для власної розробки становить 0,088, для сервісу MonkeyLearn – 0,089, WordCount – 0.087, Komprehend – 0.048. Отже маємо зменшення повноти пошуку ключових слів за Жаккаром на 0,1% у порівнянні з найкращим результатом іншого метода при застосуванні на текстах обсягом 150-400 слів.

Аналогічно були проведені випробування на текстах обсягом 1500-2500 слів.

Результати отриманих значень повноти пошуку ключових слів за Жаккаром (рис. 4.10) демонструють, що середнє значення для власної розробки становить 0,058, для сервісу MonkeyLearn – 0,108, WordCount – 0.065, Komprehend – 0.012. Отже маємо зменшення повноти пошуку ключових слів за Жаккаром на 5% у порівнянні з найкращим результатом іншого метода при застосуванні на текстах обсягом 1500-2500 слів.

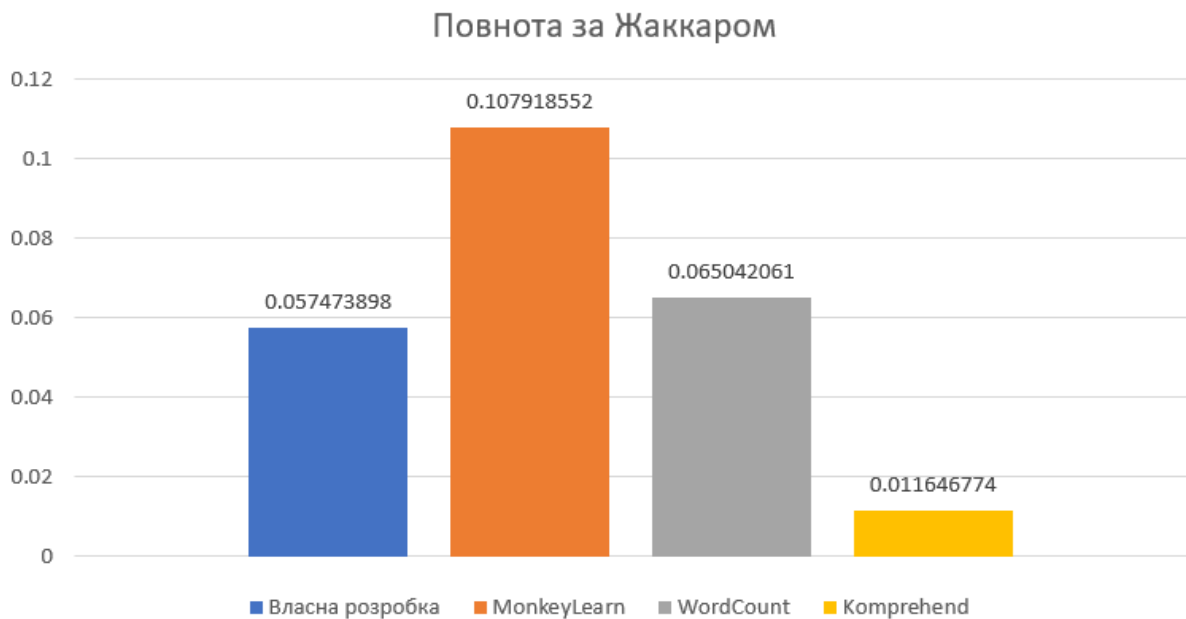


Рис. 4.10. Гістограма порівняння значень повноти пошуку ключових слів за Жаккаром для текстів обсягом 1500-2500 слів

За результатами отриманих значень повноти пошуку ключових слів за Жаккаром можна відмітити наступне:

1. На текстах невеликого обсягу (150-400 слів) значення повноти майже однакові, окрім значень сервісу Komprehend, що базується на використанні методів статистики. Це можна пояснити тим, що даний сервіс видавав помітно більше ключових слів у результатах, чим значно понижував повноту через збільшення вербального шуму.
2. На текстах більшого обсягу (1500-2500 слів) запропонована модифікація має третій результат, що можна пояснити більшою кількістю ключових слів на виході, у порівнянні з такими аналогами як MonkeyLearn та WordCount. MonkeyLearn має найкращий результат, адже сервіс завжди обмежує кількість ключових слів до 10. Найгірший результат – у сервіса Komprehend, який отримує в результаті занадто багато вербального шуму і має повноту пошуку близько 1%.

На основі отриманих результатів порівнянь можна стверджувати, що запропонований модифікований метод має меншу повноту пошуку ключових слів за Жаккаром у порівнянні з аналогами, однак не найменшу. На етапі пошуку ключових термінів, що складаються з кількох слів, дещо збільшується вербальний шум, що зменшує значення повноти пошуку, причому кількість вербального шуму збільшується зі збільшенням обсягу тексту.

4.2. Обґрунтування отриманих результатів

Продемонструємо роботу програмного забезпечення і спробуємо пояснити отримані результати.

Для початку продемонструємо роботу програмної реалізації методу на короткому тексті, що складається з двох речень, і покажемо крок за кроком як змінюється перелік слів кандидатів після кожного етапу.

Вміст тексту для випробування наступний: “Born in Honolulu, Hawaii, Obama is a graduate of Columbia University and Harvard Law School, where he was president of the Harvard Law Review. He was a community organizer in Chicago before earning his law degree” [61]. Даний текст хоч і не має зазначений авторами перелік ключових термінів, але його часто використовують під час тестування або демонстрації комп’ютерних інструментів для обробки текстів. Незважаючи на свій невеликий обсяг, даний текст допоможе продемонструвати особливості використання кореференційних зв’язків для заміни займенників на відповідні іменники, а також використання інформації про наявні в тексті синтаксичні зв’язки для отримання ключових термінів з кількох слів.

Перед запуском додатку вміщуємо текст для обробки у текстовий файл input.txt (рис. 4.11).

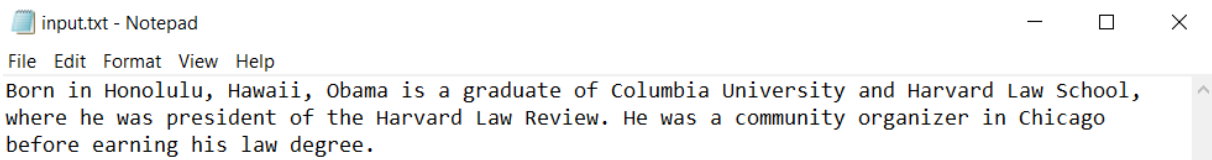


Рис. 4.11. Вміщення короткого тексту в окремий текстовий файл для випробування програмної реалізації методу

Оскільки для цього тексту в нас немає зазначеного авторами переліку ключових термінів, то аргументи запуску, що відповідають за встановлення бажаної кількості знайдених однослівних ключових термінів та шлях до файлу з правильними ключовими термінами, не вказуються (рис. 4.12).

```
python .\keyword_lib.py -i input.txt -o out.txt
```

Рис. 4.12. Команда запуску додатку для пошуку ключових термінів в тексті

Після етапу отримання всіх зв'язків між словами в тексті, маємо список пар слів зі зв'язками між ними та деякою важливою додатковою інформацією (лістинг 4.1).

Лістинг 4.1. Пари слів, отримані після етапу виявлення зв'язків між словами в тексті [61]

```
[{'main_word': {'word': 'graduate', 'address': 10, 'pos': 'NN', 'lemma': 'graduate'}, 'dependent_word': {'word': 'Born', 'address': 1, 'pos': 'VBN', 'lemma': 'Born'}, 'dep_type': 'advcl'}, {'main_word': {'word': 'Born', 'address': 1, 'pos': 'VBN', 'lemma': 'Born'}, 'dependent_word': {'word': 'Hawaii', 'address': 5, 'pos': 'NNP', 'lemma': 'Hawaii'}, 'dep_type': 'obl'}, {'main_word': {'word': 'Hawaii', 'address': 5, 'pos': 'NNP', 'lemma': 'Hawaii'}, 'dependent_word': {'word': 'in', 'address': 2, 'pos': 'IN', 'lemma': 'in'}, 'dep_type': 'case'}, {'main_word': {'word': 'Hawaii', 'address': 5, 'pos': 'NNP', 'lemma': 'Hawaii'}, 'dependent_word': {'word': 'Honolulu', 'address': 3, 'pos': 'NNP', 'lemma': 'Honolulu'}, 'dep_type': 'compound'}, {'main_word': {'word': 'graduate', 'address': 10, 'pos': 'NN', 'lemma': 'graduate'}, 'dependent_word': {'word': 'president', 'address': 22, 'pos': 'NN', 'lemma': 'president'}, 'dep_type': 'advcl'}, {'main_word': {'word': 'president', 'address': 22, 'pos': 'NN', 'lemma': 'president'}, 'dependent_word': {'word': 'where', 'address': 19, 'pos': 'WRB', 'lemma': 'where'}, 'dep_type': 'advmod'}, {'main_word': {'word': 'president', 'address': 22, 'pos': 'NN', 'lemma': 'president'}, 'dependent_word': {'word': 'he', 'address': 20, 'pos': 'PRP', 'lemma': 'he'}, 'dep_type': 'nsubj'}, {'main_word': {'word': 'president', 'address': 22, 'pos': 'NN', 'lemma': 'president'}, 'dependent_word': {'word': 'was', 'address': 21,
```

```

'pos': 'VBD', 'lemma': 'be'}, 'dep_type': 'cop'}, {'main_word': {'word':
'president', 'address': 22, 'pos': 'NN', 'lemma': 'president'},
'dependent_word': {'word': 'Review', 'address': 27, 'pos': 'NNP',
'lemma': 'Review'}, 'dep_type': 'nmod'}, {'main_word': {'word': 'Review',
'address': 27, 'pos': 'NNP', 'lemma': 'Review'}, 'dependent_word':
{'word': 'of', 'address': 23, 'pos': 'IN', 'lemma': 'of'}, 'dep_type':
'case'}, {'main_word': {'word': 'Review', 'address': 27, 'pos': 'NNP',
'lemma': 'Review'}, 'dependent_word': {'word': 'the', 'address': 24,
'pos': 'DT', 'lemma': 'the'}, 'dep_type': 'det'}, {'main_word': {'word':
'Review', 'address': 27, 'pos': 'NNP', 'lemma': 'Review'},
'dependent_word': {'word': 'Harvard', 'address': 25, 'pos': 'NNP',
'lemma': 'Harvard'}, 'dep_type': 'compound'}, {'main_word': {'word':
'Review', 'address': 27, 'pos': 'NNP', 'lemma': 'Review'},
'dependent_word': {'word': 'Law', 'address': 26, 'pos': 'NNP', 'lemma':
'Law'}, 'dep_type': 'compound'}, {'main_word': {'word': 'graduate',
'address': 10, 'pos': 'NN', 'lemma': 'graduate'}, 'dependent_word':
{'word': 'Obama', 'address': 7, 'pos': 'NNP', 'lemma': 'Obama'},
'dep_type': 'nsubj'}, {'main_word': {'word': 'graduate', 'address': 10,
'pos': 'NN', 'lemma': 'graduate'}, 'dependent_word': {'word': 'is',
'address': 8, 'pos': 'VBZ', 'lemma': 'be'}, 'dep_type': 'cop'},
{'main_word': {'word': 'graduate', 'address': 10, 'pos': 'NN', 'lemma':
'graduate'}, 'dependent_word': {'word': 'a', 'address': 9, 'pos': 'DT',
'lemma': 'a'}, 'dep_type': 'det'}, {'main_word': {'word': 'graduate',
'address': 10, 'pos': 'NN', 'lemma': 'graduate'}, 'dependent_word':
{'word': 'University', 'address': 13, 'pos': 'NNP', 'lemma':
'University'}, 'dep_type': 'nmod'}, {'main_word': {'word': 'University',
'address': 13, 'pos': 'NNP', 'lemma': 'University'}, 'dependent_word':
{'word': 'of', 'address': 11, 'pos': 'IN', 'lemma': 'of'}, 'dep_type':
'case'}, {'main_word': {'word': 'University', 'address': 13, 'pos':
'NNP', 'lemma': 'University'}, 'dependent_word': {'word': 'Columbia',
'address': 12, 'pos': 'NNP', 'lemma': 'Columbia'}, 'dep_type':
'compound'}, {'main_word': {'word': 'University', 'address': 13, 'pos':
'NNP', 'lemma': 'University'}, 'dependent_word': {'word': 'School',
'address': 17, 'pos': 'NNP', 'lemma': 'School'}, 'dep_type': 'conj'},
{'main_word': {'word': 'School', 'address': 17, 'pos': 'NNP', 'lemma':
'School'}, 'dependent_word': {'word': 'and', 'address': 14, 'pos': 'CC',
'lemma': 'and'}, 'dep_type': 'cc'}, {'main_word': {'word': 'School',
'address': 17, 'pos': 'NNP', 'lemma': 'School'}, 'dependent_word':
{'word': 'Harvard', 'address': 15, 'pos': 'NNP', 'lemma': 'Harvard'},
'dep_type': 'compound'}, {'main_word': {'word': 'School', 'address': 17,
'pos': 'NNP', 'lemma': 'School'}, 'dependent_word': {'word': 'Law',
'address': 16, 'pos': 'NNP', 'lemma': 'Law'}, 'dep_type': 'compound'},
{'main_word': {'word': 'organizer', 'address': 33, 'pos': 'NN', 'lemma':
'organizer'}, 'dependent_word': {'word': 'He', 'address': 29, 'pos':
'PRP', 'lemma': 'He'}, 'dep_type': 'nsubj'}, {'main_word': {'word':
'organizer', 'address': 33, 'pos': 'NN', 'lemma': 'organizer'},
'dependent_word': {'word': 'was', 'address': 30, 'pos': 'VBD', 'lemma':
'be'}, 'dep_type': 'cop'}, {'main_word': {'word': 'organizer', 'address':
33, 'pos': 'NN', 'lemma': 'organizer'}, 'dependent_word': {'word': 'a',
'address': 31, 'pos': 'DT', 'lemma': 'a'}, 'dep_type': 'det'},
{'main_word': {'word': 'organizer', 'address': 33, 'pos': 'NN', 'lemma':
'organizer'}, 'dependent_word': {'word': 'community', 'address': 32,
'pos': 'NN', 'lemma': 'community'}, 'dep_type': 'compound'},
{'main_word': {'word': 'organizer', 'address': 33, 'pos': 'NN', 'lemma':
'organizer'}, 'dependent_word': {'word': 'Chicago', 'address': 35, 'pos':
'NNP', 'lemma': 'Chicago'}, 'dep_type': 'nmod'}, {'main_word': {'word':
'Chicago', 'address': 35, 'pos': 'NNP', 'lemma': 'Chicago'},
'dependent_word': {'word': 'in', 'address': 34, 'pos': 'IN', 'lemma':
'in'}, 'dep_type': 'case'}, {'main_word': {'word': 'organizer',
'address': 33, 'pos': 'NN', 'lemma': 'organizer'}, 'dependent_word':
{'word': 'earning', 'address': 37, 'pos': 'VBG', 'lemma': 'earn'},
'dep_type': 'advcl'}, {'main_word': {'word': 'earning', 'address': 37,
'pos': 'VBG', 'lemma': 'earn'}, 'dependent_word': {'word': 'before',
'address': 36, 'pos': 'IN', 'lemma': 'before'}, 'dep_type': 'mark'},

```

```
{'main_word': {'word': 'earning', 'address': 37, 'pos': 'VBG', 'lemma': 'earn'}, 'dependent_word': {'word': 'degree', 'address': 40, 'pos': 'NN', 'lemma': 'degree'}, 'dep_type': 'obj'}, {'main_word': {'word': 'degree', 'address': 40, 'pos': 'NN', 'lemma': 'degree'}, 'dependent_word': {'word': 'his', 'address': 38, 'pos': 'PRP$', 'lemma': 'his'}, 'dep_type': 'nmod:poss'}, {'main_word': {'word': 'degree', 'address': 40, 'pos': 'NN', 'lemma': 'degree'}, 'dependent_word': {'word': 'law', 'address': 39, 'pos': 'NN', 'lemma': 'law'}, 'dep_type': 'compound'}]
```

Кожна пара слів представлена об'єктом з наступними атрибутами:

- `main_word` – головне слово в зв'язку, в свою чергу є об'єктом з атрибутами:
 - `word` – як слово виражене в тексті;
 - `address` – абсолютна позиція слова в тексті;
 - `pos` – частина мови, до якої належить слово;
 - `lemma` – основна форма слова;
- `dependent_word` – залежне слово в зв'язку, має такі ж атрибути як і головне слово;
- `dep_type` – тип синтаксичного зв'язку.

Одразу після цього етапу вже можна отримати список багатослівних виразів в тексті, між словами яких визначено зв'язки типу `flat` та `compound` (лістинг 4.2).

Лістинг 4.2. Багатослівні вирази в тексті, між словами яких було визначено зв'язки типу `flat` та `compound`

```
"Honolulu Hawaii", "Harvard Law Review", "Columbia University", "Harvard Law School", "community organizer", "law degree"
```

Наступний етап методу полягає у виключенні пар з неінформативними типами зв'язків. Після цього етапу були видалені наступні пари (лістинг 4.3).

Лістинг 4.3. Пари слів в тексті, які було видалено внаслідок фільтрування неінформативних зв'язків

```
{ 'main_word': { 'word': 'Review', 'address': 27, 'pos': 'NNP', 'lemma': 'Review' }, 'dependent_word': { 'word': 'the', 'address': 24, 'pos': 'DT', 'lemma': 'the' }, 'dep_type': 'det' }  
{ 'main_word': { 'word': 'graduate', 'address': 10, 'pos': 'NN', 'lemma': 'graduate' }, 'dependent_word': { 'word': 'a', 'address': 9, 'pos': 'DT', 'lemma': 'a' }, 'dep_type': 'det' }  
{ 'main_word': { 'word': 'School', 'address': 17, 'pos': 'NNP', 'lemma': 'School' }, 'dependent_word': { 'word': 'and', 'address': 14, 'pos': 'CC', 'lemma': 'and' }, 'dep_type': 'cc' }  
{ 'main_word': { 'word': 'organizer', 'address': 33, 'pos': 'NN', 'lemma': 'organizer' }, 'dependent_word': { 'word': 'a', 'address': 31, 'pos': 'DT', 'lemma': 'a' }, 'dep_type': 'det' }
```

З лістингу 4.3 видно, що видалені пари дійсно мали неінформативні типи зв'язків, а саме “CC” та “DET”.

Наступний етап передбачає заміну займенників на відповідні іменники, для збільшення ваги важливих слів при зважуванні слів-кандидатів у ключові слова. Для цього наперед проводиться аналіз кореференційних зв'язків в тексті. В результаті такого аналізу, отримується адреси усіх сутностей та відповідних займенників в тексті. В результатах аналізатора кореференційних зв'язків такі структури називаються кластерами.

Для тексту [61] було отримано наступний список кластерів (лістинг 4.4).

Лістинг 4.4. Кластери кореференційних зв'язків, які було отримано після аналізу тексту

```
[[[6, 6], [19, 19], [28, 28], [37, 37]]]
```

Структура кластерів показує, що в даному тексті знайдено лише одну сутність, яка неодноразово згадується як у вигляді іменників та займенників. За відповідними адресами в тексті знаходяться слова “Obama”, “he”, “He”, “his”. Отже, у всіх парах слів, що містять ці займенники, проводиться заміна займенника на іменник “Obama” (лістинг 4.5).

Лістинг 4.5. Заміна займенників на відповідні іменники в парах слів з

тексту

```
{ 'main_word': { 'word': 'president', 'address': 22, 'pos': 'NN', 'lemma': 'president' }, 'dependent_word': { 'word': 'he', 'address': 20, 'pos': 'PRP', 'lemma': 'he' }, 'dep_type': 'nsubj' } ⇒ { 'main_word': { 'word': 'president', 'address': 22, 'pos': 'NN', 'lemma': 'president' }, 'dependent_word': { 'word': 'he', 'address': 20, 'pos': 'NNP', 'lemma': 'Obama' }, 'dep_type': 'nsubj' }  
{ 'main_word': { 'word': 'organizer', 'address': 33, 'pos': 'NN', 'lemma': 'organizer' }, 'dependent_word': { 'word': 'He', 'address': 29, 'pos': 'PRP', 'lemma': 'He' }, 'dep_type': 'nsubj' } ⇒ { 'main_word': { 'word': 'organizer', 'address': 33, 'pos': 'NN', 'lemma': 'organizer' }, 'dependent_word': { 'word': 'He', 'address': 29, 'pos': 'NNP', 'lemma': 'Obama' }, 'dep_type': 'nsubj' }  
{ 'main_word': { 'word': 'degree', 'address': 40, 'pos': 'NN', 'lemma': 'degree' }, 'dependent_word': { 'word': 'his', 'address': 38, 'pos': 'PRP$', 'lemma': 'his' }, 'dep_type': 'nmod:poss' } ⇒ { 'main_word': { 'word': 'degree', 'address': 40, 'pos': 'NN', 'lemma': 'degree' }, 'dependent_word': { 'word': 'his', 'address': 38, 'pos': 'NNP', 'lemma': 'Obama' }, 'dep_type': 'nmod:poss' }
```

Наступний крок полягає у видаленні слів, що відносяться до неінформативних частин мови, тому попередня структура пар не підходить для видалення окремих слів. На цьому кроці для методу вже не важлива інформація про типи зв'язків між словами, тож для зручності пари розбиваються на окремі слова (лістинг 4.6).

Лістинг 4.6. Список слів після розбиття пар для тексту

```
[ { 'word': 'graduate', 'address': 10, 'pos': 'NN', 'lemma': 'graduate' },  
  { 'word': 'Born', 'address': 1, 'pos': 'VBN', 'lemma': 'Born' }, { 'word': 'Born', 'address': 1, 'pos': 'VBN', 'lemma': 'Born' }, { 'word': 'Hawaii', 'address': 5, 'pos': 'NNP', 'lemma': 'Hawaii' }, { 'word': 'Hawaii', 'address': 5, 'pos': 'NNP', 'lemma': 'Hawaii' }, { 'word': 'in', 'address': 2, 'pos': 'IN', 'lemma': 'in' }, { 'word': 'Hawaii', 'address': 5, 'pos': 'NNP', 'lemma': 'Hawaii' }, { 'word': 'Honolulu', 'address': 3, 'pos': 'NNP', 'lemma': 'Honolulu' }, { 'word': 'graduate', 'address': 10, 'pos': 'NN', 'lemma': 'graduate' }, { 'word': 'president', 'address': 22, 'pos': 'NN', 'lemma': 'president' }, { 'word': 'president', 'address': 22, 'pos': 'NN', 'lemma': 'president' }, { 'word': 'where', 'address': 19, 'pos': 'WRB', 'lemma': 'where' }, { 'word': 'president', 'address': 22, 'pos': 'NN', 'lemma': 'president' }, { 'word': 'he', 'address': 20, 'pos': 'NNP', 'lemma': 'Obama' }, { 'word': 'president', 'address': 22, 'pos': 'NN', 'lemma': 'president' }, { 'word': 'was', 'address': 21, 'pos': 'VBD', 'lemma': 'be' }, { 'word': 'president', 'address': 22, 'pos': 'NN', 'lemma': 'president' }, { 'word': 'Review', 'address': 27, 'pos': 'NNP', 'lemma': 'Review' }, { 'word': 'Review', 'address': 27, 'pos': 'NNP', 'lemma': 'Review' }, { 'word': 'of', 'address': 23, 'pos': 'IN', 'lemma': 'of' }, { 'word': 'Review', 'address': 27, 'pos': 'NNP', 'lemma': 'Review' }, { 'word': 'Harvard', 'address': 25, 'pos': 'NNP', 'lemma': 'Harvard' }
```

```
'Harvard'}, {'word': 'Review', 'address': 27, 'pos': 'NNP', 'lemma':
'Review'}, {'word': 'Law', 'address': 26, 'pos': 'NNP', 'lemma': 'Law'},
{'word': 'graduate', 'address': 10, 'pos': 'NN', 'lemma': 'graduate'},
{'word': 'Obama', 'address': 7, 'pos': 'NNP', 'lemma': 'Obama'}, {'word':
'graduate', 'address': 10, 'pos': 'NN', 'lemma': 'graduate'}, {'word':
'is', 'address': 8, 'pos': 'VBZ', 'lemma': 'be'}, {'word': 'graduate',
'address': 10, 'pos': 'NN', 'lemma': 'graduate'}, {'word': 'University',
'address': 13, 'pos': 'NNP', 'lemma': 'University'}, {'word':
'University', 'address': 13, 'pos': 'NNP', 'lemma': 'University'},
{'word': 'of', 'address': 11, 'pos': 'IN', 'lemma': 'of'}, {'word':
'University', 'address': 13, 'pos': 'NNP', 'lemma': 'University'},
{'word': 'Columbia', 'address': 12, 'pos': 'NNP', 'lemma': 'Columbia'},
{'word': 'University', 'address': 13, 'pos': 'NNP', 'lemma':
'University'}, {'word': 'School', 'address': 17, 'pos': 'NNP', 'lemma':
'School'}, {'word': 'School', 'address': 17, 'pos': 'NNP', 'lemma':
'School'}, {'word': 'Harvard', 'address': 15, 'pos': 'NNP', 'lemma':
'Harvard'}, {'word': 'School', 'address': 17, 'pos': 'NNP', 'lemma':
'School'}, {'word': 'Law', 'address': 16, 'pos': 'NNP', 'lemma': 'Law'},
{'word': 'organizer', 'address': 33, 'pos': 'NN', 'lemma': 'organizer'},
{'word': 'He', 'address': 29, 'pos': 'NNP', 'lemma': 'Obama'}, {'word':
'organizer', 'address': 33, 'pos': 'NN', 'lemma': 'organizer'}, {'word':
'was', 'address': 30, 'pos': 'VBD', 'lemma': 'be'}, {'word': 'organizer',
'address': 33, 'pos': 'NN', 'lemma': 'organizer'}, {'word': 'community',
'address': 32, 'pos': 'NN', 'lemma': 'community'}, {'word': 'organizer',
'address': 33, 'pos': 'NN', 'lemma': 'organizer'}, {'word': 'Chicago',
'address': 35, 'pos': 'NNP', 'lemma': 'Chicago'}, {'word': 'Chicago',
'address': 35, 'pos': 'NNP', 'lemma': 'Chicago'}, {'word': 'in',
'address': 34, 'pos': 'IN', 'lemma': 'in'}, {'word': 'organizer',
'address': 33, 'pos': 'NN', 'lemma': 'organizer'}, {'word': 'earning',
'address': 37, 'pos': 'VBG', 'lemma': 'earn'}, {'word': 'earning',
'address': 37, 'pos': 'VBG', 'lemma': 'earn'}, {'word': 'before',
'address': 36, 'pos': 'IN', 'lemma': 'before'}, {'word': 'earning',
'address': 37, 'pos': 'VBG', 'lemma': 'earn'}, {'word': 'degree',
'address': 40, 'pos': 'NN', 'lemma': 'degree'}, {'word': 'degree',
'address': 40, 'pos': 'NN', 'lemma': 'degree'}, {'word': 'his',
'address': 38, 'pos': 'NNP', 'lemma': 'Obama'}, {'word': 'degree',
'address': 40, 'pos': 'NN', 'lemma': 'degree'}, {'word': 'law',
'address': 39, 'pos': 'NN', 'lemma': 'law'}}
```

На етапі видалення слів, що відносяться до неінформативних частин мови було видалено такі слова (лістинг 4.7).

Лістинг 4.7. Вилучені слова на етапі виключення слів з неінформативних частин мови для тексту

```
{'word': 'in', 'address': 2, 'pos': 'IN', 'lemma': 'in'}
{'word': 'where', 'address': 19, 'pos': 'WRB', 'lemma': 'where'}
{'word': 'of', 'address': 23, 'pos': 'IN', 'lemma': 'of'}
{'word': 'of', 'address': 11, 'pos': 'IN', 'lemma': 'of'}
{'word': 'in', 'address': 34, 'pos': 'IN', 'lemma': 'in'}
{'word': 'before', 'address': 36, 'pos': 'IN', 'lemma': 'before'}
```

Як видно зі списку видалених слів (лістинг 4.7), більшість з них є прийменниками та прислівниками.

Наступним є етап видалення стоп-слів. Для цього використовується стандартний словник стоп-слів для англійської мови, що вбудований в NLTK. На даному етапі зі списку слів були видалені стоп-слова (лістинг 4.8).

Лістинг 4.8. Вилучені слова на етапі виключення стоп-слів для тексту

```
{'word': 'was', 'address': 21, 'pos': 'VBD', 'lemma': 'be'}  
{'word': 'is', 'address': 8, 'pos': 'VBZ', 'lemma': 'be'}  
{'word': 'was', 'address': 30, 'pos': 'VBD', 'lemma': 'be'}
```

Далі відбувається відбір n перших слів з найбільшою кількістю зв'язків. Для цього варто лише обчислити скільки разів кожне слово, а саме його основна форма, міститься в цьому списку. Кількість зв'язків для кожного слова можна вважати вагою слова при зважуванні (лістинг 4.9).

Лістинг 4.9. Відсортовані за кількістю зв'язків слова з тексту

```
[{'lemma': 'graduate', 'freq': 5}, {'lemma': 'president', 'freq': 5},  
{ 'lemma': 'organizer', 'freq': 5}, {'lemma': 'obama', 'freq': 4},  
{ 'lemma': 'review', 'freq': 4}, {'lemma': 'university', 'freq': 4},  
{ 'lemma': 'hawaii', 'freq': 3}, {'lemma': 'law', 'freq': 3}, {'lemma':  
'school', 'freq': 3}, {'lemma': 'earn', 'freq': 3}, {'lemma': 'degree',  
'freq': 3}, {'lemma': 'born', 'freq': 2}, {'lemma': 'harvard', 'freq':  
2}, {'lemma': 'chicago', 'freq': 2}, {'lemma': 'honolulu', 'freq': 1},  
{ 'lemma': 'columbia', 'freq': 1}, {'lemma': 'community', 'freq': 1}]
```

В даному випадку для тексту не було вказано n під час запуску програми, тож ключовими будуть вважатися слова, що залишились після всіх етапів фільтрування, отже слова з лістингу 4.9.

Варто зазначити, що в цьому випадку колізії не відбулося, адже вона може відбутися тільки тоді, коли кількість слів-кандидатів у ключові слова буде більшою за вказаний параметр n . Цей випадок продемонстровано в наступному прикладі.

Для отримання списку ключових термінів, що складаються з кількох слів, відповідно до методу необхідно відфільтрувати всі багатослівні вирази з лістингу 4.2, спираючись на отриманий список ключових термінів з одного слова. В даному випадку всі багатослівні вирази були визначені ключовими,

отже в результаті об'єднання термінів з лістингу 4.9 та 4.2 отримуємо повний перелік ключових термінів для тексту (лістинг 4.10).

Лістинг 4.10. Ключові терміни для тексту, що були знайдені програмною реалізацією модифікованого методу

```
graduate, president, organizer, obama, review, university, hawaii, law,  
school, earn, degree, born, harvard, chicago, honolulu, columbia,  
community, community organizer, Harvard Law Review, Columbia University,  
Honolulu Hawaii, Harvard Law School, law degree
```

Як вже було зазначено, текст з минулого прикладу був досить коротким і не мав заданого авторами списку “еталонних” ключових термінів. Це робить неможливим перевірку точності результатів роботи методу.

Для цього проаналізуємо текст, для якого наданий перелік визначених авторами ключових термінів. Шукати ключові терміни будемо в тексті тез статті “Design Optimization of Micro Air Launch Vehicle Using Differential Evolution” [62]. Текст тез візьмемо для зручності, тому що саме для коротких текстів результати досить легко аналізувати, а також прослідковуються закономірності, які можуть вказати на ті чи інші причини отриманих результатів.

На вхід подається текст тез в окремому текстовому файлі *input.txt* (рис. 4.13).

In this paper, we have used the differential evolution to optimize the design of a Micro Air Launch Vehicle and its launch trajectory. Since trajectory design of a launch vehicle requires prior knowledge of the masses and propulsion performance parameters of the Micro Air Launch Vehicle, whereas the vehicle design requires prior knowledge of the required velocity (ΔV) to insert the required payload into the target orbit, a two-step optimization cycle was adopted. A Micro Air Launch Vehicle was designed to launch a 20-kg payload into a 400-km circular polar orbit. The preliminary design of the Micro Air Launch Vehicle was conducted given the required (ΔV), which was obtained from trajectory optimization, and then applied in mission analysis to obtain the initial masses. These initial masses were used in the vehicle design to get the performance and geometry parameters. The objective function of the Micro Air Launch Vehicle design optimization is to minimize the initial mass under specified constraints on the insertion orbit. The objective of trajectory optimization is to maximize the payload mass under constraints on orbit specifications and design variables. For the 20-kg payload mass, the optimal initial mass is 1267.8 kg and optimal payload is 20.6 kg, which slightly exceeds the mission requirements.

Рис. 4.13. Текст тез статті “Design Optimization of Micro Air Launch Vehicle Using Differential Evolution”

Для даної статті авторами був заданий такий набір ключових термінів: *Micro Air Launch, Vehicle design, Optimization, Differential evolution*. З чотирьох ключових термінів три складаються з кількох слів, що має продемонструвати особливості модифікованого методу. Ці терміни подаються в окремому текстовому файлі `reference.txt` через кому.

Для запуску роботи програми можемо вказати бажану кількість однослівних ключових слів, аналогічно до того, як працює метод, взятий за основу модифікації. Рекомендується обирати число, що відповідає кількості всіх слів зі списку заданого авторами тексту. Для тексту з даного прикладу в переліку ключових слів містяться 8 окремих слів, тож при запуску вказуємо число 8 (рис. 4.14).

```
(venv) PS C:\Users\3A43Mka\Documents\diploma-testing> python .\main.py -i input.txt -o out.txt -n 8 -r .\reference.txt
```

Рис. 4.14. Команда запуску програми для пошуку ключових термінів в тексті

Знову наводити повний покроковий розбір не будемо, але прокоментуємо випадок колізії, який саме присутній при пошуку ключових термінів в даному тексті. Нагадаємо, що колізія виникає, коли кількість слів-кандидатів, що пройшли всі етапи фільтрування, більша, ніж бажана

кількість шуканих ключових слів (задається під час запуску як параметр n), і потрібно вибирати серед слів-кандидатів з однаковою кількістю зв'язків.

Після етапу виключення стоп-слів під час оброблення тексту [62], маємо наступний перелік слів-кандидатів, відсортований за кількістю зв'язків (лістинг 4.11).

Лістинг 4.11. Відсортовані за кількістю зв'язків слова-кандидати з тексту [62]

```
[{'lemma': 'vehicle', 'freq': 25}, {'lemma': 'design', 'freq': 18},  
{ 'lemma': 'mass', 'freq': 18}, {'lemma': 'launch', 'freq': 15}, {'lemma':  
'optimization', 'freq': 12}, {'lemma': 'orbit', 'freq': 11}, {'lemma':  
'require', 'freq': 10}, {'lemma': 'use', 'freq': 9}, {'lemma': 'obtain',  
'freq': 9}, {'lemma': 'parameter', 'freq': 7}, {'lemma': 'payload',  
'freq': 7}, {'lemma': 'constraint', 'freq': 7}, {'lemma': 'trajectory',  
'freq': 6}, {'lemma': 'knowledge', 'freq': 6}, {'lemma': 'micro', 'freq':  
5}, {'lemma': 'air', 'freq': 5}, {'lemma': 'velocity', 'freq': 4},  
{ 'lemma': 'insert', 'freq': 4}, {'lemma': 'apply', 'freq': 4}, {'lemma':  
'initial', 'freq': 4}, {'lemma': 'minimize', 'freq': 4}, {'lemma':  
'maximize', 'freq': 4}, {'lemma': 'specification', 'freq': 4}, {'lemma':  
'exceed', 'freq': 4}, {'lemma': 'optimize', 'freq': 3}, {'lemma':  
'adopt', 'freq': 3}, {'lemma': 'performance', 'freq': 3}, {'lemma':  
'cycle', 'freq': 3}, {'lemma': 'conduct', 'freq': 3}, {'lemma':  
'analysis', 'freq': 3}, {'lemma': 'get', 'freq': 3}, {'lemma':  
'function', 'freq': 3}, {'lemma': 'objective', 'freq': 3}, {'lemma':  
'paper', 'freq': 2}, {'lemma': 'evolution', 'freq': 2}, {'lemma':  
'prior', 'freq': 2}, {'lemma': 'required', 'freq': 2}, {'lemma': 'step',  
'freq': 2}, {'lemma': 'mission', 'freq': 2}, {'lemma': 'variable',  
'freq': 2}, {'lemma': 'optimal', 'freq': 2}, {'lemma': 'requirement',  
'freq': 2}, {'lemma': 'differential', 'freq': 1}, {'lemma': 'propulsion',  
'freq': 1}, {'lemma': 'target', 'freq': 1}, {'lemma': 'circular', 'freq':  
1}, {'lemma': 'polar', 'freq': 1}, {'lemma': 'preliminary', 'freq': 1},  
{ 'lemma': 'give', 'freq': 1}, {'lemma': 'geometry', 'freq': 1}, {'lemma':  
'specify', 'freq': 1}, {'lemma': 'insertion', 'freq': 1}, {'lemma':  
'slightly', 'freq': 1}]
```

Після цього за етапами методу необхідно відібрати перші n слів з найбільшою кількістю зв'язків. Перші 7 відібрати легко – це слова: “vehicle”, “design”, “mass”, “launch”, “optimization”, “orbit”, “require”. Однак під час вибору восьмого терміну, залишаються два слова “use” та “obtain” з однаковою вагою – обидва слова мають по 9 зв'язків.

Колізія вирішується наступним чином: для всіх спірних слів з однаковою кількістю зв'язків будується новий “рейтинг” з кількості зв'язків кожного слова зі словами, що вже були визначені як ключові на попередніх

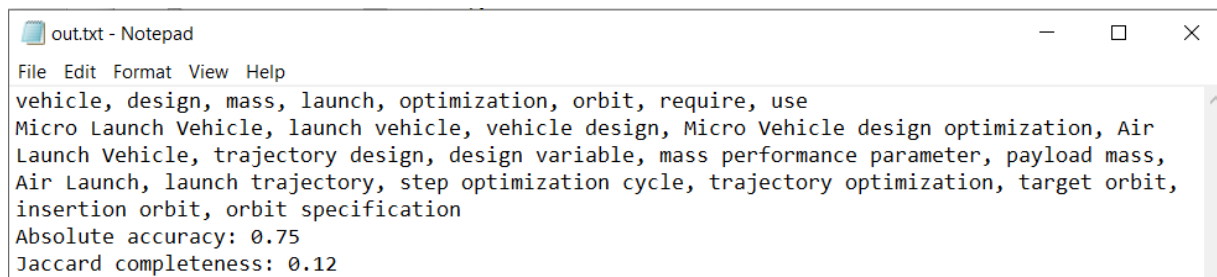
етапах. Для даного прикладу отримуємо наступний оновлений рейтинг (лістинг 4.12).

Лістинг 4.12. Відсортовані за кількістю зв'язків слова-кандидати з тексту

```
[{'lemma': 'use', 'freq': 2},  
{ 'lemma': 'obtain', 'freq': 1}]
```

З прикладу видно, що слово “use” за новим рейтингом має більшу вагу ніж “obtain”, тож обирається як ключове. Остаточний список ключових термінів, що складаються з одного слова, є таким: “vehicle”, “design”, “mass”, “launch”, “optimization”, “orbit”, “require”, “use”.

Після роботи програми і оброблення вхідного тексту результати пошуку ключових слів можна знайти в файлі виводу *out.txt* (рис. 4.15).



```
vehicle, design, mass, launch, optimization, orbit, require, use  
Micro Launch Vehicle, launch vehicle, vehicle design, Micro Vehicle design optimization, Air  
Launch Vehicle, trajectory design, design variable, mass performance parameter, payload mass,  
Air launch, launch trajectory, step optimization cycle, trajectory optimization, target orbit,  
insertion orbit, orbit specification  
Absolute accuracy: 0.75  
Jaccard completeness: 0.12
```

Рис. 4.15. Файл результатів роботи програми для тексту тез статті “Design Optimization of Micro Air Launch Vehicle Using Differential Evolution”

В першому рядку файлу результатів перелічені знайдені ключові терміни, що складаються з одного слова. В другому рядку вказані знайдені ключові терміни, що складаються з кількох слів.

Нижче подані значення метрик абсолютної точності та повноти пошуку ключових слів за Жаккаром. Для даного випробування абсолютна точність пошуку дорівнює 0.75. Це значить, що було правильно знайдено три з чотирьох заданих авторами ключових слів. За допомогою додаткових повідомлень у консоль під час виконання програми можна побачити, що

правильно було знайдено терміни "*Optimization*", "*Vehicle design*" та "*Micro Air Launch*".

Слово "*optimization*" зустрічається 4 рази в тексті, тож очевидно, що доволі висока для тексту невеликого обсягу частота входження забезпечила появу слова в списку ключових.

Термін "*vehicle design*" зустрічається в тексті 3 рази, а окремо слова "*vehicle*" та "*design*" зустрічаються 8 разів, тому цей термін потрапив в список ключових.

Термін "*micro air launch*" зустрічається в тексті 5 разів, окремо слова "*micro*" та "*air*" – по 5 разів, слово "*launch*" – 8 разів, що в сукупності надає терміну достатню вагу для потрапляння в список ключових.

Метод не зміг правильно визначити термін "*differential evolution*" як ключовий вірогідно через низьку частоту. Даний термін зустрічається в тексті лише один раз, як і окремо слова "*differential*" та "*evolution*". Можна вважати, що через низьку частоту цей термін було втрачено методом.

Значення повноти пошуку за Жаккаром для даного тексту дорівнює 0.12. На основі численних випробувань було помічено, що майже завжди це значення буде нижче за значення абсолютної точності. Метрика повноти за Жаккаром враховує загальну кількість знайдених програмою ключових слів, на відміну від метрики абсолютної точності, яка враховує лише кількість правильно знайдених ключових слів. Для даного випробування програма знайшла 24 ключових терміни, що значно більше за 4, зазначені авторами статті. Як було зазначено у попередньому підрозділі, відбір ключових термінів зі списку багатослівних виразів додає вербального шуму, але в порівнянні з аналогами його кількість не можна назвати занадто великою.

Для збільшення цього показника є кілька способів. Наприклад, сервіс MonkeyLearn, з яким велося порівняння в попередньому підрозділі, завжди обмежує кількість результатів до 10. Але якщо кількість правильно

знайдених ключових слів замала, обмеження не допоможе підвищити повноту.

Найкращим способом, але найскладнішим в реалізації, є зменшення вербального шуму в результатах пошуку, а отже збільшення точності пошуку ключових слів. Неможливо однозначно визначити найкраще рішення, але до таких належить додатковий аналіз слів та зв'язків в тексті за деякими інформативними параметрами, що допоможуть при зважуванні слів-кандидатів. Дослідження в цьому напрямку продовжують вести спеціалісти з всього світу.

Є ще один випадок, коли методу складно або навіть неможливо правильно визначити деяке ключове слово. Це той випадок, коли авторами тексту в якості ключового вказано термін, який зовсім не зустрічається в оброблюваному тексті, але підходить за тематикою тексту.

Для статті “The role of antioxidant on propellant binder reactivity during thermal aging” [63] наведено такий перелік ключових слів: *Antioxidant*, *Composite propellant*, *HTPB propellant*, *Polyurethane resin*, *Storage*. Однак в тексті тез статті немає слів “*antioxidant*”, “*composite*”, “*propellant*”, “*resin*” та “*storage*”, що робить знаходження будь-якого з термінів за допомогою методу неможливим. Варто зауважити, що сервіси, з якими виконувалися порівняння в попередньому підрозділі, теж не змогли правильно відшукати жоден з ключових термінів для даного тексту.

Можливим рішенням даної нетривіальної проблеми міг би бути додатковий аналіз тематики текстів для подальшого підбору відповідних термінів з тематичних словників.

4.3. Напрямки подальшої роботи

Напрямки подальшої роботи над покращенням запропонованого модифікованого гібридного методу автоматизованого пошуку ключових слів та термінів у тексті полягають у підвищенні ефективності роботи даного методу та розширенні його можливостей. Потрібним вбачається

збільшення абсолютної точності та повноти за Жаккаром. Пропонується кілька перспективних способів покращення методу:

1. Провести додаткове тренування моделей, що використовуються в синтаксичних парсерах. Інструменти Python NLTK та AllenNLP, які були застосовані для створення програмної реалізації запропонованої модифікації методу, надають стандартні усереднені моделі даних за замовчуванням для простоти використання парсерів. Однак розробники цих програмних пакетів зазначають, що для досягнення кращих результатів, пропонується додатково натренувати моделі для своїх потреб. Необхідно дослідити, яким чином можна покращити моделі саме для задачі автоматизованого пошуку ключових слів в текстах, що може збільшити абсолютну точність.
2. Використовувати синонімічні та тематичні словники для пошуку ключових термінів. Як було показано на прикладі в попередньому підрозділі, найбільш влучні ключові слова не завжди містяться в самому тексті, але вони є дотичними до тематики тексту. В такому випадку більшість методів, що розглядають лише набори слів з тексту, не зможуть відшукати такі ключові терміни. Додатковий аналіз синонімів чи тематичних словників може допомогти методу “не губити” відсутні в тексті ключові терміни, які тим не менш є релевантними для тексту.
3. Зменшити кількість вербального шуму в результатах пошуку. Це можна досягти кількома способами, один з яких полягає у покращенні алгоритму відсіювання багатослівних виразів, що не є ключовими. Потрібно дослідити, яким чином аналіз складних залежностей між словами багатослівних виразів та іншими словами в тексті може допомогти у ранжуванні та

зважуванні результатів, що має позитивно вплинути на значення повноти пошуку за Жаккаром.

4. Провести більшу кількість досліджень на текстах різних тематик та стилю. Наразі дослідження були проведені на текстах науково-технічної тематики, адже досить легко отримати список ключових слів, заданих авторами, бо вони додаються до статей. Аналіз роботи методів при застосуванні на текстах художнього стилю може продемонструвати інші результати, які допоможуть зробити більше правильних висновків.
5. Додати підтримку пошуку ключових слів в текстах інших природних мов. В даній програмній реалізації підтримується лише оброблення текстів англійською мовою, але в теорії можливо додати підтримку інших природних мов, якщо вдасться знайти або розробити моделі та дані для мовних аналізаторів з відповідної мови.

4.4. Висновки до четвертого розділу

У даному розділі проаналізовано ефективність розробленого програмного забезпечення, що реалізує запропонований модифікований метод для автоматизованого пошуку ключових слів, у порівнянні з сервісами MonkeyLearn (машинне навчання і статистичний аналіз), WordCount (аналіз N-грам) та Komprehend (статистичний аналіз) за критеріями абсолютної точності та повноти пошуку за Жаккаром. В результаті порівнянь з'ясовано, що запропонований модифікований метод має кращі показники абсолютної точності, але повнота може бути нижчою ніж в інших аналогів.

Наведено напрямки подальших досліджень, які полягають в підвищенні точності та повноти пошуку ключових слів, проведенні

випробувань на інших видах текстів та додаванні підтримки пошуку в текстах інших природних мов.

ВИСНОВКИ

Дана дисертація присвячена створенню модифікованого методу та програмного забезпечення автоматизованого пошуку ключових слів у природномовних текстових даних.

В ході дослідження розглянуто основні типи методів пошуку ключових слів у природномовних текстових даних, наведено відомі методи та їх особливості. Також розглянуто існуючі програмні рішення, що надають інструменти для пошуку ключових слів в текстах. Помічено спільні закономірності та прийоми в різних методах і зроблено висновки про перспективні напрямки дослідження теми.

Для підвищення ефективності засобів для автоматизованого пошуку ключових слів в тексті вирішено розробити модифікацію одного з сучасних методів. Наведено гіпотези, які були сформульовано щодо шляхів покращення оригінального методу, зазначено причини відхилення першої гіпотези і перспективність другої гіпотези, де пропонується аналіз даних про багатослівні вирази в тексті для пошуку ключових термінів, що складаються з кількох слів. Описано етапи запропонованої модифікації методу. Розглянуто метрики, за якими пропонується оцінювати ефективність запропонованого методу та порівнювати його з існуючими аналогами.

Розроблено програмну реалізацію запропонованого модифікованого методу у вигляді додатку на мові Python, який може бути використаний за допомогою CLI як консольний додаток для демонстрації роботи методу, а також у вигляді бібліотеки в програмному коді. Розроблене програмне забезпечення створено з використанням сучасних лінгвістичних програмних пакетів та бібліотек для роботи з текстовими даними. Розглянуто архітектуру, особливості реалізації та наведено приклад використання для програмної реалізації.

Проведено аналіз ефективності запропонованого модифікованого методу за критеріями абсолютної точності та повноти пошуку ключових

слів за Жаккаром. Наведено обґрунтування отриманих результатів. Порівняння з існуючими аналогами показує, що запропонований метод дозволяє підвищити абсолютну точність пошуку ключових слів у межах від 6.2% до 13,6% в текстах обсягом 150-400 слів та у межах від 1.7% до 34,1% в текстах обсягом 1500-2500 слів, але має дещо меншу повноту за Жаккаром.

Подальші напрямки роботи над удосконаленням запропонованого методу полягають у підвищенні ефективності роботи методу та розширенні його можливостей, а саме: проведенні додаткового тренування моделей для синтаксичних парсерів; використанні синонімічних та тематичних словників для пошуку ключових термінів, які не зустрічаються в тексті; зменшенні кількості вербального шуму в результатах пошуку; проведенні більшої кількості досліджень на текстах різних тематик та стилю; додаванні підтримки пошуку ключових слів в текстах інших природних мов.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. C. Zhang, H. Wang, Y. Liu, D. Wu, Y. Liao, and B. Wang, "Automatic keyword extraction from documents using conditional random fields", *Journal of Computational Information Systems* №4, pp. 1169-1180, 2008.
2. Palshikar, "Keyword extraction from a single document using centrality measures", *Lecture Notes in Computer Science Pattern Recognition and Machine Intelligence*, pp. 503-510, 2007.
3. S. Lahiri, S. Choudhury, and C. Caragea, "Keyword and keyphrase extraction using centrality measures on collocation networks", *arXiv preprint arXiv:1401.6571*, 2014.
4. N. Schluter, "Centrality Measures for Non-Contextual Graph-Based Unsupervised Single Document Keyword Extraction. In *Proceedings of TALN 2014 (Volume 2: Short Papers)*", Marseille, France. Association pour le Traitement Automatique des Langues, pp. 455-460, 2014.
5. O. Bisikalo, A. Lisovenko, O. Jahumovuch, S. Trachenko, and M. Pradivliannyi, "System of computational linguistic on base of the figurative text comprehension", *Proceedings of the 2016 IEEE 1st International Conference on Data Stream Mining and Processing (DSMP 2016)*, pp. 69-74, 2016. DOI: 10.1109/DSMP.2016.7583510.
6. Y. HaCohen-Kerner, "Automatic extraction of keywords from abstracts", *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pp. 843-849, 2003
7. N. Pudota, A. Dattolo, A. Baruzzo, and C. Tasso, "A New Domain Independent Keyphrase Extraction System", *Communications in Computer and Information Science Digital Libraries*, pp. 67-78, 2010. doi: 10.1007/978-3-642-15850-6_8
8. A. Hulth, B. Megyesi, "A Study on Automatically Extracted Keywords in Text Categorization", *In Proceedings of the 21st International Conference on*

- Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, pp. 537-544, 2006.
9. Le Quan Ha, E. I. Sicilia-Garcia, Ji Ming, F. J. Smith, "Extension of Zipf's Law to Words and Phrases", The 19th International Conference on Computational Linguistics, 2002.
 10. J. Blaták, E. Mráková, L. Popelínský, "Fragments and Text Categorization", In Proceedings of the ACL Interactive Poster and Demonstration Sessions, Barcelona, Spain, pp. 226-229, 2004.
 11. T. Joachims, "Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization", Carnegie-mellon Univ Pittsburgh PA Dept of computer science, No. CMU-CS-96-118, pp. 143-151, 1996.
 12. Zhang, M., Yang, H., Ji, D., Teng, C., Wu, H., "Discourse Coherence: Lexical Chain, Complex Network and Semantic Field", Chinese Lexical Semantics, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, vol 7717, 2013.
 13. Casimiro, Y. Ledeneva, G. Hernández, R. Arnulfo, R. Corchado, "Lexical Patterns Based on Maximal Frequent Sequences for Automatic Keyphrase Extraction", *Computación y Sistemas*. 25., 2021.
 14. T. Ruas, W. Grosky, "Keyword Extraction Through Contextual Semantic Analysis of Documents" pp. 150-156, 2017, doi:10.1145/3167020.3167043.
 15. M. RizoIU, J. Velcin. "Topic Extraction for Ontology Learning", pp. 38-61, 2011.
 16. M. Baroni, "BootCaT: Bootstrapping Corpora and Terms from the Web", Proceedings of LREC. Lisbon: ELDA, pp. 1313-1316, 2004.
 17. K. Frantzi, S. Ananiadou, and H. Mima, "Automatic recognition of multi-word terms: the C-value/NC-value method", *International Journal on Digital Libraries*, Vol. 3, pp. 115-130, 2000.
 18. A. Andrenucci, "Creating a Bilingual Psychology Lexicon for Cross Lingual Question Answering", pp. 129-136, 2007.

19. Yamada, Yasuhiro & Himeno, Yuusuke & Nakatoh, Tetsuya. (2018). Weighting of Noun Phrases Based on Local Frequency of Nouns. 10.1007/978-3-319-72550-5_42.
20. O. Medelyan, and I. Witten, "Thesaurus based automatic keyphrase indexing", Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries, pp. 296-297, 2006. doi:10.1145/1141753.1141819
21. A. Przepiórkowski, "Slavic Information Extraction and Partial Parsing", In Proceedings of the Workshop on Balto-Slavonic Natural Language Processing, Prague, Czech Republic., pp. 1-10, 2007.
22. P. Turney, "Learning to extract keyphrases from text", Technical report, National Research Council, Institute for Information Technology, pp. 143-147, 1999.
23. P. Chen, and S. Lin, "Automatic keyword prediction using Google similarity distance", Expert Systems with Applications, №37, pp. 1928-1938, 2010. doi:10.1016/j.eswa.2009.07.016
24. K. Scannell, "Corpus building for under-resourced languages", In Building and Exploring Web Corpora: Proceedings of the 3rd Web as Corpus Workshop, Vol. 4, pp. 5-15, 2007.
25. J. Reed, Y. Jiao, T. Potok, B. Klump, M. Elmore, and A. Hurson, "TF-ICF: A New Term Weighting Scheme for Clustering Dynamic Data Streams", Proc. Machine Learning and Applications, pp. 258-263, 2006.
26. R. Mihalcea, and A. Csomai, "Wikify!: linking documents to encyclopedic knowledge", Proceedings of the sixteenth ACM conference on Conference on information and knowledge management, pp. 233-242, 2007.
27. Salton, "The SMART Retrieval System - experiments in automatic", Prentice-Hall, Inc., Englewood Cliffs, pp. 111-121, 1971.
28. Z. Dejin, and M. Rosson, "How and why people Twitter: the role that micro blogging plays in informal communication at work", Proceedings of the ACM 2009 international conference on Supporting group work, pp. 31-40, 2009.

29. B. Huberman, D. Romero, and F. Wu, "Social networks that matter: Twitter under the microscope", *First Monday*, Vol. 14, №1, pp. 14-21, 2008.
30. О.В. Яхимович, "Інформаційна технологія пошуку ключових слів на основі парсингу англomовних текстів", Вінниця, 2021.
31. W. Wong, W. Liu, and M. Bennamoun, "Ontology Learning from Text", *ACM Computing Surveys* 44, no. 4 (August 1, 2012), pp. 1-36, 2012. doi:10.1145/2333112.2333115
32. Keyword Extractor – MonkeyLearn [Електронний ресурс]. Режим доступу: <https://monkeylearn.com/keyword-extractor-online/>
33. Keyword Extractor – WordCount [Електронний ресурс]. Режим доступу: <https://wordcount.com/keyword-extractor>
34. Keyword Extractor – Komprehend [Електронний ресурс]. Режим доступу: <https://komprehend.io/keyword-extractor>
35. Termine Web Demonstrator, [Електронний ресурс]. Режим доступу: <http://www.nactem.ac.uk/software/termine/>
36. Maui Indexer, [Електронний ресурс]. Режим доступу: <https://code.google.com/archive/p/maui-indexer/>
37. Semrush Content Analyzer, [Електронний ресурс]. Режим доступу: <https://www.semrush.com/kb/699-content-analyzer>
38. Acrolinx, [Електронний ресурс]. Режим доступу: <https://www.acrolinx.com/integrations/>
39. TreeTagger, [Електронний ресурс]. Режим доступу: <https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/>
40. Five Filters Term Extractor, [Електронний ресурс]. Режим доступу: <https://www.fivefilters.org/term-extraction/>
41. Five Filters Term Extraction Description, [Електронний ресурс]. Режим доступу: <https://www.fivefilters.org/2013/term-extraction/>
42. MultiCorpora Term Base, [Електронний ресурс]. Режим доступу: https://www.multicorpora.ca/termbase_e.html

43. Multiterm Desktop, [Электронный ресурс]. Режим доступа: <https://www.trados.com/products/multiterm-desktop/>
44. Terminotix, [Электронный ресурс]. Режим доступа: <https://terminotix.com/index.asp?content=category&cat=6&lang=en>
45. Marie-Catherine de Marneffe, Christopher D. Manning (2008). Stanford typed dependencies manual [Online] – Available from: https://downloads.cs.stanford.edu/nlp/software/dependencies_manual.pdf, last accessed 2024/01/08
46. Beatrice Santorini (1990). Part-of-Speech Tag-ging Guidelines for the Penn Treebank Project [Online] – Available from: <https://www.cis.upenn.edu/~bries/manuals/tagguide.pdf>, last accessed 2024/01/08.
47. L. Burgareli, "Variability management in software product lines using adaptive object and reflection. Journal of Aerospace Technology and Management, V. 1, № 2.", Journal of Aerospace Technology and Management, V. 1, № 2. (2009, Jul.-Dec.)
48. About keywords in Search Network campaigns, [Электронный ресурс]. Режим доступа: https://support.google.com/google-ads/answer/1704371?hl=en&ref_topic=3119131&sjid=7992566250880472617-EU
49. Universal Dependencies, [Электронный ресурс]. Режим доступа: <https://universaldependencies.org/u/dep/>
50. Fixed dependency [Online] – Available from: <https://universaldependencies.org/u/dep/fixed.html>, last accessed 2024/01/08
51. Flat dependency [Online] – Available from: <https://universaldependencies.org/u/dep/flat.html>, last accessed 2024/01/08.
52. Compound dependency [Online] – Available from: <https://universaldependencies.org/u/dep/compound>, last accessed 2024/01/08.

53. Maurício G. Silva, Vitor O.R. Gamarra, Vitor Koldaev, “Control of Reynolds number in a high speed wind tunnel”, Journal of Aerospace Technology and Management, V. 1, № 1. (2009, Jan.-Jul.) [Online].
54. Klakow, Dietrich; Jochen Peters (September 2002). "Testing the correlation of word error rate and perplexity". Speech Communication. 38 (1–2): 19–28. doi:10.1016/S0167-6393(01)00041-3. ISSN 0167-6393
55. Jawadekar, Waman S (2011), "8. Knowledge Management: Tools and Technology", Knowledge Management: Text & Cases, New Delhi: Tata McGraw-Hill Education Private Ltd, p. 278, ISBN 978-0-07-07-0086-4, retrieved November 23, 2012
56. Natural Language Toolkit, [Электронный ресурс]. Режим доступа: <https://www.nltk.org/>
57. A Guide to Natural Language Processing With AllenNLP, [Электронный ресурс]. Режим доступа: <https://guide.allennlp.org/>
58. spaCy: Industrial-Strength Natural Language Processing, [Электронный ресурс]. Режим доступа: <https://spacy.io/>
59. JiWER [Электронный ресурс]. Режим доступа: <https://jitsi.github.io/jiwer/>
60. Journal of Aerospace Technology and Management [Электронный ресурс]. Режим доступа: <https://jatm.com.br/jatm/issue/archive>
61. US Presidential Doctrines Handbook - Volume 1 President Barack Obama Doctrine - Strategic Information and Materials, с. 18
62. Aldheeb, Mohammed & Kafafy, Raed & Idres, Moumen & Omar, Hanafy & Abido, Mohammed. (2012). Design Optimization of Micro Air Launch Vehicle Using Differential Evolution. Journal of Aerospace Technology and Management [Электронный ресурс]. Режим доступа: <https://jatm.com.br/jatm/article/view/154>
63. Villar, Luciene & Silva, Rafael & Diniz, Milton & Takahashi, Marta & Rezende, Luis. (2010). The role of antioxidant on propellant binder reactivity during thermal aging. Journal of Aerospace Technology and Management

[Электронный ресурс]. Режим доступа:
<https://jatm.com.br/jatm/article/view/53>

ДОДАТКИ

Додаток 1
Лістинги

Результат роботи методу raw_parse

```
defaultdict(<function DependencyGraph.__init__.<locals>.<lambda> at
0x0000024DB3803E50>,
{0: {'address': 0,
     'ctag': 'TOP',
     'deps': defaultdict(<class 'list'>, {'root': [6]}),
     'feats': None,
     'head': None,
     'lemma': None,
     'rel': None,
     'tag': 'TOP',
     'word': None},
 1: {'address': 1,
     'ctag': 'DT',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 5,
     'lemma': '_',
     'rel': 'det',
     'tag': 'DT',
     'word': 'The'},
 2: {'address': 2,
     'ctag': 'NNP',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 4,
     'lemma': '_',
     'rel': 'compound',
     'tag': 'NNP',
     'word': 'Software'},
 3: {'address': 3,
     'ctag': 'NNP',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 4,
     'lemma': '_',
     'rel': 'compound',
     'tag': 'NNP',
     'word': 'Product'},
 4: {'address': 4,
     'ctag': 'NNP',
     'deps': defaultdict(<class 'list'>, {'compound': [2, 3]}),
     'feats': '_',
     'head': 5,
     'lemma': '_',
     'rel': 'compound',
     'tag': 'NNP',
     'word': 'Line'},
 5: {'address': 5,
     'ctag': 'NN',
     'deps': defaultdict(<class 'list'>,
                        {'compound': [4],
                         'det': [1]}),
     'feats': '_',
     'head': 6,
     'lemma': '_',
     'rel': 'nsubj',
```

```

        'tag': 'NN',
        'word': 'approach'},
6: {'address': 6,
    'ctag': 'VBZ',
    'deps': defaultdict(<class 'list'>,
                        {'conj': [17],
                         'nsubj': [5],
                         'obj': [7]}),

    'feats': '_',
    'head': 0,
    'lemma': '_',
    'rel': 'root',
    'tag': 'VBZ',
    'word': 'offers'},
7: {'address': 7,
    'ctag': 'NNS',
    'deps': defaultdict(<class 'list'>, {'nmod': [10]}),
    'feats': '_',
    'head': 6,
    'lemma': '_',
    'rel': 'obj',
    'tag': 'NNS',
    'word': 'benefits'},
8: {'address': 8,
    'ctag': 'JJ',
    'deps': defaultdict(<class 'list'>, {'fixed': [9]}),
    'feats': '_',
    'head': 10,
    'lemma': '_',
    'rel': 'case',
    'tag': 'JJ',
    'word': 'such'},
9: {'address': 9,
    'ctag': 'IN',
    'deps': defaultdict(<class 'list'>, {}),
    'feats': '_',
    'head': 8,
    'lemma': '_',
    'rel': 'fixed',
    'tag': 'IN',
    'word': 'as'},
10: {'address': 10,
    'ctag': 'NNS',
    'deps': defaultdict(<class 'list'>,
                       {'appos': [15],
                        'case': [8]}),

    'feats': '_',
    'head': 7,
    'lemma': '_',
    'rel': 'nmod',
    'tag': 'NNS',
    'word': 'savings'},
12: {'address': 12,
    'ctag': 'JJ',
    'deps': defaultdict(<class 'list'>, {}),
    'feats': '_',
    'head': 14,

```

```

        'lemma': '_',
        'rel': 'amod',
        'tag': 'JJ',
        'word': 'large'},
13: {'address': 13,
     'ctag': 'HYPH',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 14,
     'lemma': '_',
     'rel': 'punct',
     'tag': 'HYPH',
     'word': '-'},
14: {'address': 14,
     'ctag': 'NN',
     'deps': defaultdict(<class 'list'>,
                          {'amod': [12],
                           'punct': [13]}),
     'feats': '_',
     'head': 15,
     'lemma': '_',
     'rel': 'compound',
     'tag': 'NN',
     'word': 'scale'},
15: {'address': 15,
     'ctag': 'NN',
     'deps': defaultdict(<class 'list'>, {'compound': 14}),
     'feats': '_',
     'head': 10,
     'lemma': '_',
     'rel': 'appos',
     'tag': 'NN',
     'word': 'productivity'},
16: {'address': 16,
     'ctag': 'CC',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 17,
     'lemma': '_',
     'rel': 'cc',
     'tag': 'CC',
     'word': 'and'},
17: {'address': 17,
     'ctag': 'VBD',
     'deps': defaultdict(<class 'list'>,
                          {'advcl': [27],
                           'cc': [16],
                           'obj': [19],
                           'obl': [23]}),
     'feats': '_',
     'head': 6,
     'lemma': '_',
     'rel': 'conj',
     'tag': 'VBD',
     'word': 'increased'},
18: {'address': 18,
     'ctag': 'NN',

```

```

        'deps': defaultdict(<class 'list'>, {}),
        'feats': '_',
        'head': 19,
        'lemma': '_',
        'rel': 'compound',
        'tag': 'NN',
        'word': 'product'},
19: {'address': 19,
     'ctag': 'NN',
     'deps': defaultdict(<class 'list'>, {'compound': 18})),
     'feats': '_',
     'head': 17,
     'lemma': '_',
     'rel': 'obj',
     'tag': 'NN',
     'word': 'quality'},
20: {'address': 20,
     'ctag': 'IN',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 23,
     'lemma': '_',
     'rel': 'case',
     'tag': 'IN',
     'word': 'to'},
21: {'address': 21,
     'ctag': 'DT',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 23,
     'lemma': '_',
     'rel': 'det',
     'tag': 'DT',
     'word': 'the'},
22: {'address': 22,
     'ctag': 'NN',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 23,
     'lemma': '_',
     'rel': 'compound',
     'tag': 'NN',
     'word': 'software'},
23: {'address': 23,
     'ctag': 'NN',
     'deps': defaultdict(<class 'list'>,
                        {'case': [20],
                         'compound': [22],
                         'det': [21]}),
     'feats': '_',
     'head': 17,
     'lemma': '_',
     'rel': 'obl',
     'tag': 'NN',
     'word': 'development'},
24: {'address': 24,
     'ctag': 'IN',
     'deps': defaultdict(<class 'list'>, {}),

```

```

        'feats': '_',
        'head': 27,
        'lemma': '_',
        'rel': 'mark',
        'tag': 'IN',
        'word': 'because'},
25: {'address': 25,
     'ctag': 'PRP',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 27,
     'lemma': '_',
     'rel': 'nsubj:pass',
     'tag': 'PRP',
     'word': 'it'},
26: {'address': 26,
     'ctag': 'VBZ',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 27,
     'lemma': '_',
     'rel': 'aux:pass',
     'tag': 'VBZ',
     'word': 'is'},
27: {'address': 27,
     'ctag': 'VBN',
     'deps': defaultdict(<class 'list'>,
                        {'aux:pass': [26],
                         'conj': [38],
                         'mark': [24],
                         'nsubj:pass': [25],
                         'obl': [31]}),
     'feats': '_',
     'head': 17,
     'lemma': '_',
     'rel': 'advcl',
     'tag': 'VBN',
     'word': 'based'},
28: {'address': 28,
     'ctag': 'IN',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 31,
     'lemma': '_',
     'rel': 'case',
     'tag': 'IN',
     'word': 'on'},
29: {'address': 29,
     'ctag': 'NN',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 31,
     'lemma': '_',
     'rel': 'compound',
     'tag': 'NN',
     'word': 'software'},
30: {'address': 30,

```

```

        'ctag': 'NN',
        'deps': defaultdict(<class 'list'>, {}),
        'feats': '_',
        'head': 31,
        'lemma': '_',
        'rel': 'compound',
        'tag': 'NN',
        'word': 'architecture'},
31: {'address': 31,
     'ctag': 'NNS',
     'deps': defaultdict(<class 'list'>,
                          {'acl:relcl': [36],
                           'case': [28],
                           'compound': [29, 30]}),
     'feats': '_',
     'head': 27,
     'lemma': '_',
     'rel': 'obl',
     'tag': 'NNS',
     'word': 'reuse'},
33: {'address': 33,
     'ctag': 'WDT',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 36,
     'lemma': '_',
     'rel': 'nsubj:pass',
     'tag': 'WDT',
     'word': 'which'},
34: {'address': 34,
     'ctag': 'VBZ',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 36,
     'lemma': '_',
     'rel': 'aux:pass',
     'tag': 'VBZ',
     'word': 'is'},
35: {'address': 35,
     'ctag': 'RBR',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 36,
     'lemma': '_',
     'rel': 'advmod',
     'tag': 'RBR',
     'word': 'more'},
36: {'address': 36,
     'ctag': 'VBN',
     'deps': defaultdict(<class 'list'>,
                          {'advmod': [35],
                           'aux:pass': [34],
                           'nsubj:pass': [33]}),
     'feats': '_',
     'head': 31,
     'lemma': '_',
     'rel': 'acl:relcl',

```

```

        'tag': 'VBN',
        'word': 'planned'},
37: {'address': 37,
     'ctag': 'CC',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 38,
     'lemma': '_',
     'rel': 'cc',
     'tag': 'CC',
     'word': 'and'},
38: {'address': 38,
     'ctag': 'VBN',
     'deps': defaultdict(<class 'list'>,
                          {'cc': [37],
                           'obl': [42]}),
     'feats': '_',
     'head': 27,
     'lemma': '_',
     'rel': 'conj',
     'tag': 'VBN',
     'word': 'aimed'},
39: {'address': 39,
     'ctag': 'IN',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 42,
     'lemma': '_',
     'rel': 'case',
     'tag': 'IN',
     'word': 'to'},
40: {'address': 40,
     'ctag': 'DT',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 42,
     'lemma': '_',
     'rel': 'det',
     'tag': 'DT',
     'word': 'a'},
41: {'address': 41,
     'ctag': 'JJ',
     'deps': defaultdict(<class 'list'>, {}),
     'feats': '_',
     'head': 42,
     'lemma': '_',
     'rel': 'amod',
     'tag': 'JJ',
     'word': 'specific'},
42: {'address': 42,
     'ctag': 'NN',
     'deps': defaultdict(<class 'list'>,
                          {'amod': [41],
                           'case': [39],
                           'det': [40]}),
     'feats': '_',
     'head': 38,

```

```
'lemma': '_',  
'rel': 'obl',  
'tag': 'NN',  
'word': 'domain'}})
```

Застосування .triples до результату роботи raw_parse

```
[('offers', 'VBZ'), ('nsubj', ('approach', 'NN')), ('approach', 'NN'),  
'det', ('The', 'DT')), ('approach', 'NN'), 'compound', ('Line', 'NNP')),  
(('Line', 'NNP'), 'compound', ('Software', 'NNP')), ('Line', 'NNP'),  
'compound', ('Product', 'NNP')), ('offers', 'VBZ'), 'obj', ('benefits',  
'NNS')), ('benefits', 'NNS'), 'nmod', ('savings', 'NNS')), ('savings',  
'NNS'), 'case', ('such', 'JJ')), ('such', 'JJ'), 'fixed', ('as', 'IN')),  
(('savings', 'NNS'), 'appos', ('productivity', 'NN')), ('productivity',  
'NN'), 'compound', ('scale', 'NN')), ('scale', 'NN'), 'amod', ('large',  
'JJ')), ('scale', 'NN'), 'punct', ('-', 'HYPH')), ('offers', 'VBZ'),  
'conj', ('increased', 'VBD')), ('increased', 'VBD'), 'cc', ('and', 'CC')),  
(('increased', 'VBD'), 'obj', ('quality', 'NN')), ('quality', 'NN'),  
'compound', ('product', 'NN')), ('increased', 'VBD'), 'obl', ('development',  
'NN')), ('development', 'NN'), 'case', ('to', 'IN')), ('development',  
'NN'), 'det', ('the', 'DT')), ('development', 'NN'), 'compound',  
(('software', 'NN')), ('increased', 'VBD'), 'advcl', ('based', 'VBN')),  
(('based', 'VBN'), 'mark', ('because', 'IN')), ('based', 'VBN'),  
'nsubj:pass', ('it', 'PRP')), ('based', 'VBN'), 'aux:pass', ('is', 'VBZ')),  
(('based', 'VBN'), 'obl', ('reuse', 'NNS')), ('reuse', 'NNS'), 'case',  
(('on', 'IN')), ('reuse', 'NNS'), 'compound', ('software', 'NN')), ('reuse',  
'NNS'), 'compound', ('architecture', 'NN')), ('reuse', 'NNS'), 'acl:relcl',  
(('planned', 'VBN')), ('planned', 'VBN'), 'nsubj:pass', ('which', 'WDT')),  
(('planned', 'VBN'), 'aux:pass', ('is', 'VBZ')), ('planned', 'VBN'),  
'advmod', ('more', 'RBR')), ('based', 'VBN'), 'conj', ('aimed', 'VBN')),  
(('aimed', 'VBN'), 'cc', ('and', 'CC')), ('aimed', 'VBN'), 'obl', ('domain',  
'NN')), ('domain', 'NN'), 'case', ('to', 'IN')), ('domain', 'NN'), 'det',  
(('a', 'DT')), ('domain', 'NN'), 'amod', ('specific', 'JJ'))]
```

Додаток 2
Копії графічних матеріалів

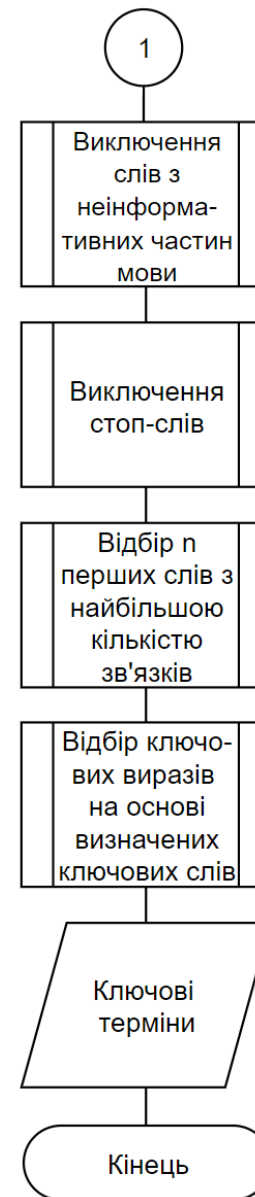
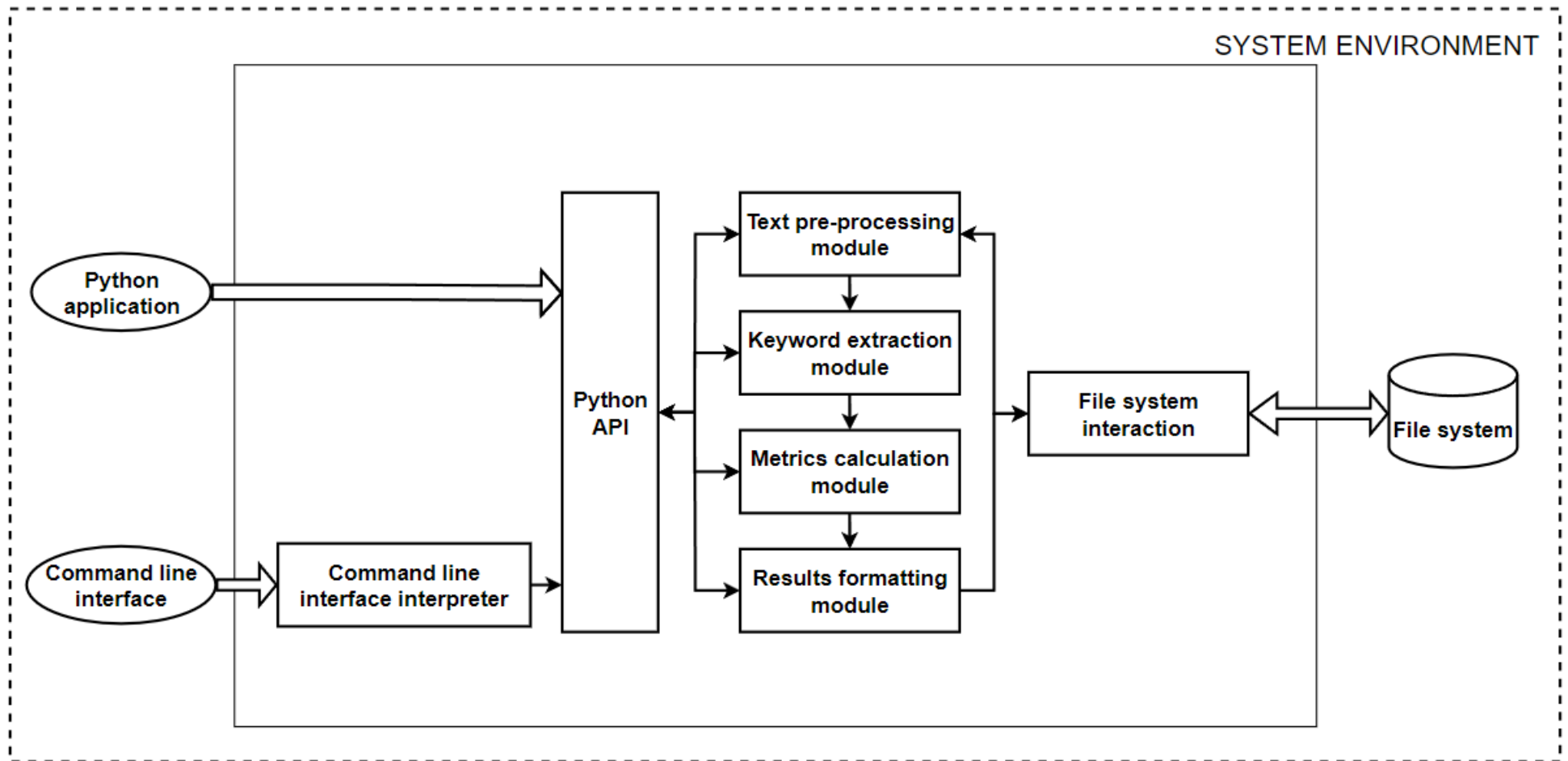
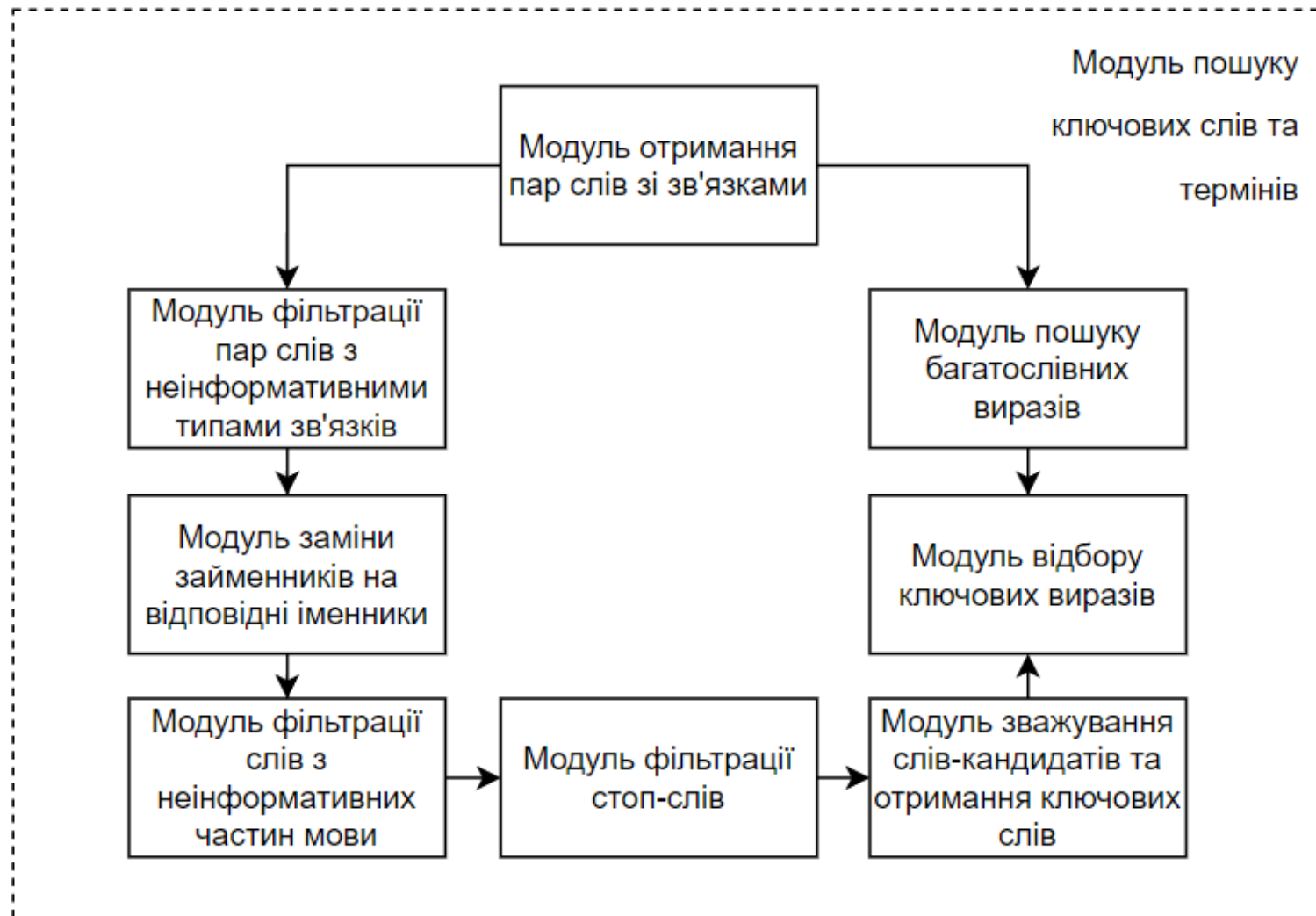


Схема етапів модифікованого методу для автоматизованого пошуку ключових слів у природномовних текстових даних

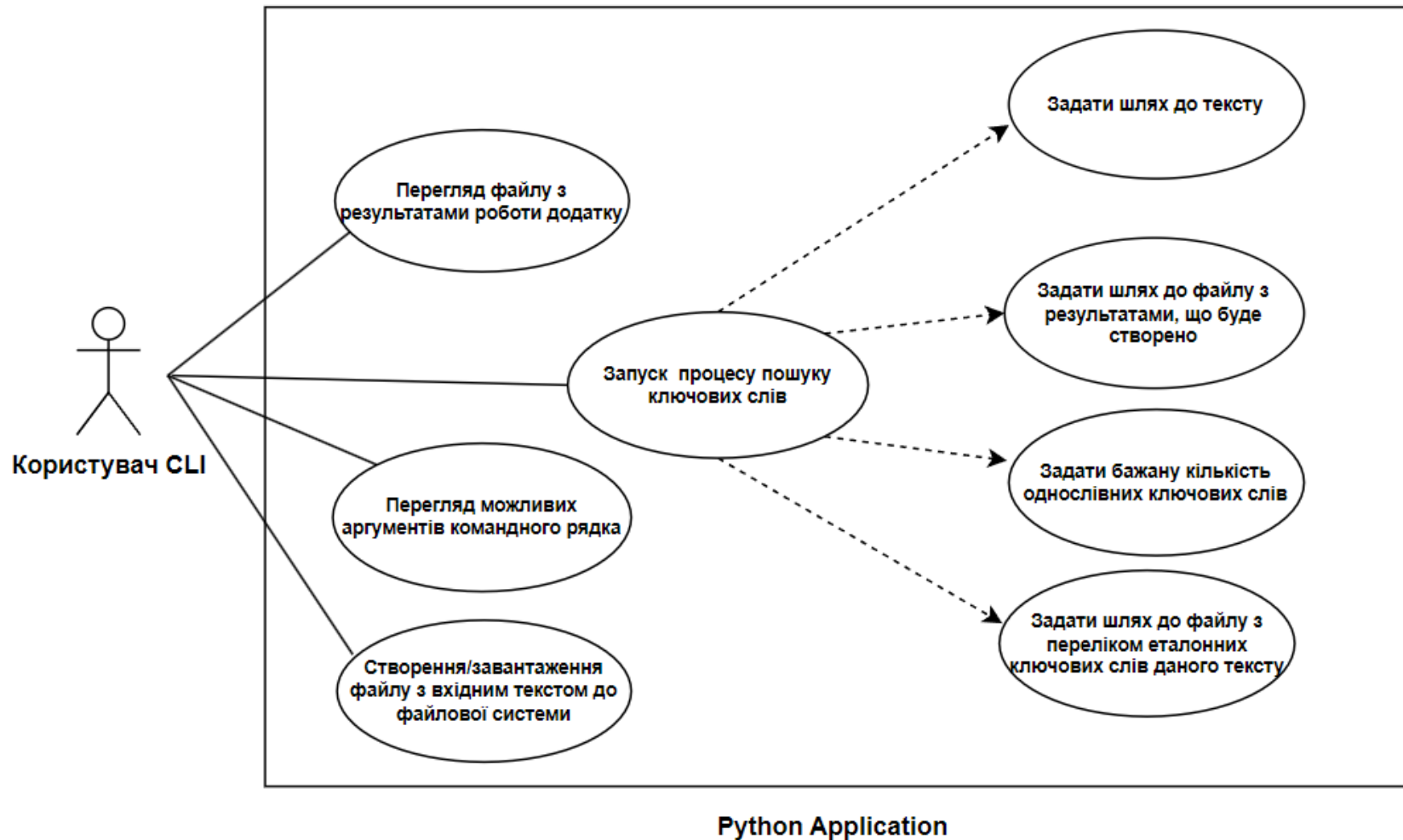
Бухаленков Д.О., група КП-21мн



Архітектура розробленого програмного забезпечення
автоматизованого пошуку ключових слів у
природномовних текстових даних
Бухаленков Д.О., група КП-21мн

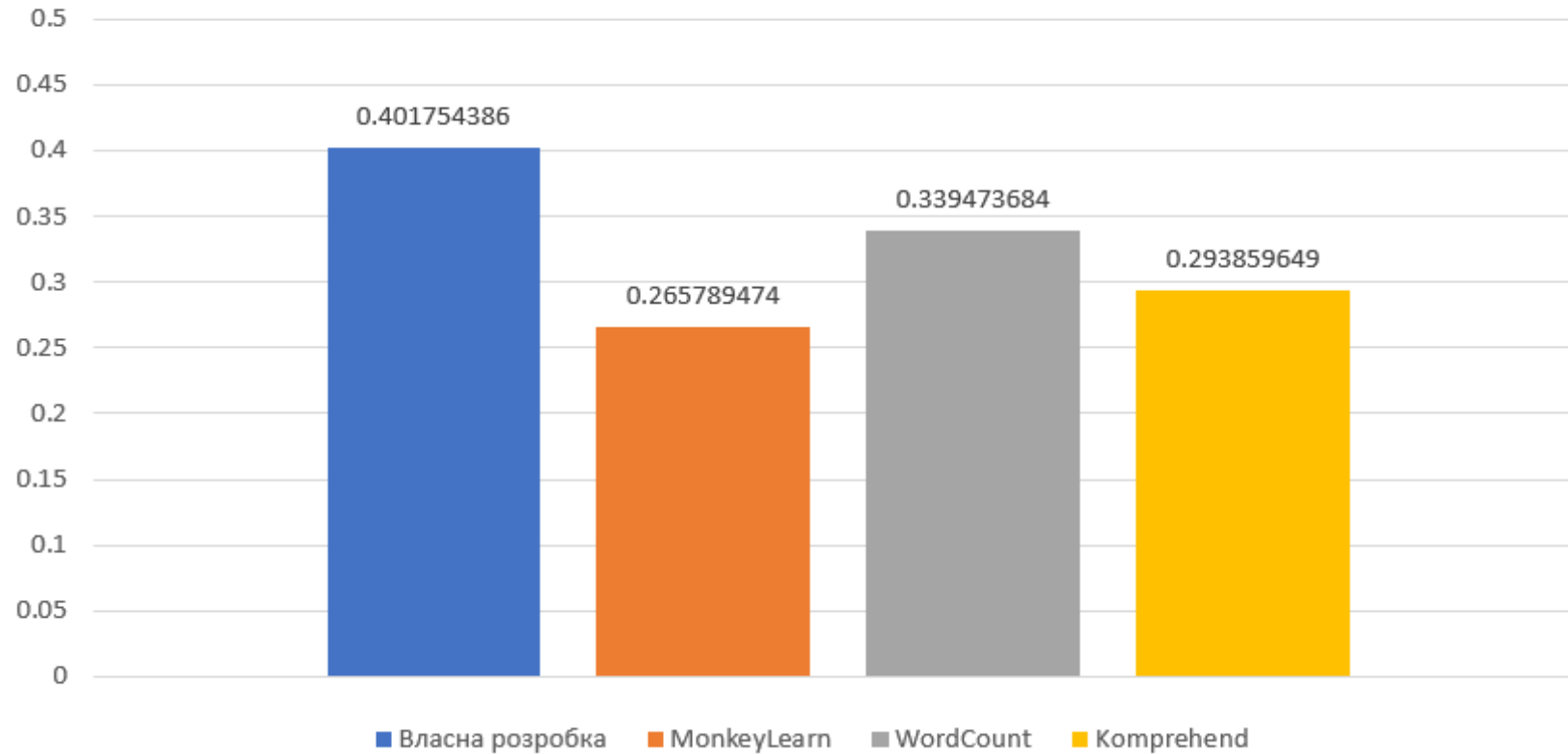


Структурна схема модуля пошуку ключових слів та термінів розробленого програмного забезпечення автоматизованого пошуку ключових слів у природномовних текстових даних
Бухаленков Д.О., група КП-21мн



UML діаграма сценаріїв використання розробленого програмного забезпечення для автоматизованого пошуку ключових слів та термінів в тексті користувачем
Бухаленков Д.О., група КП-21мн

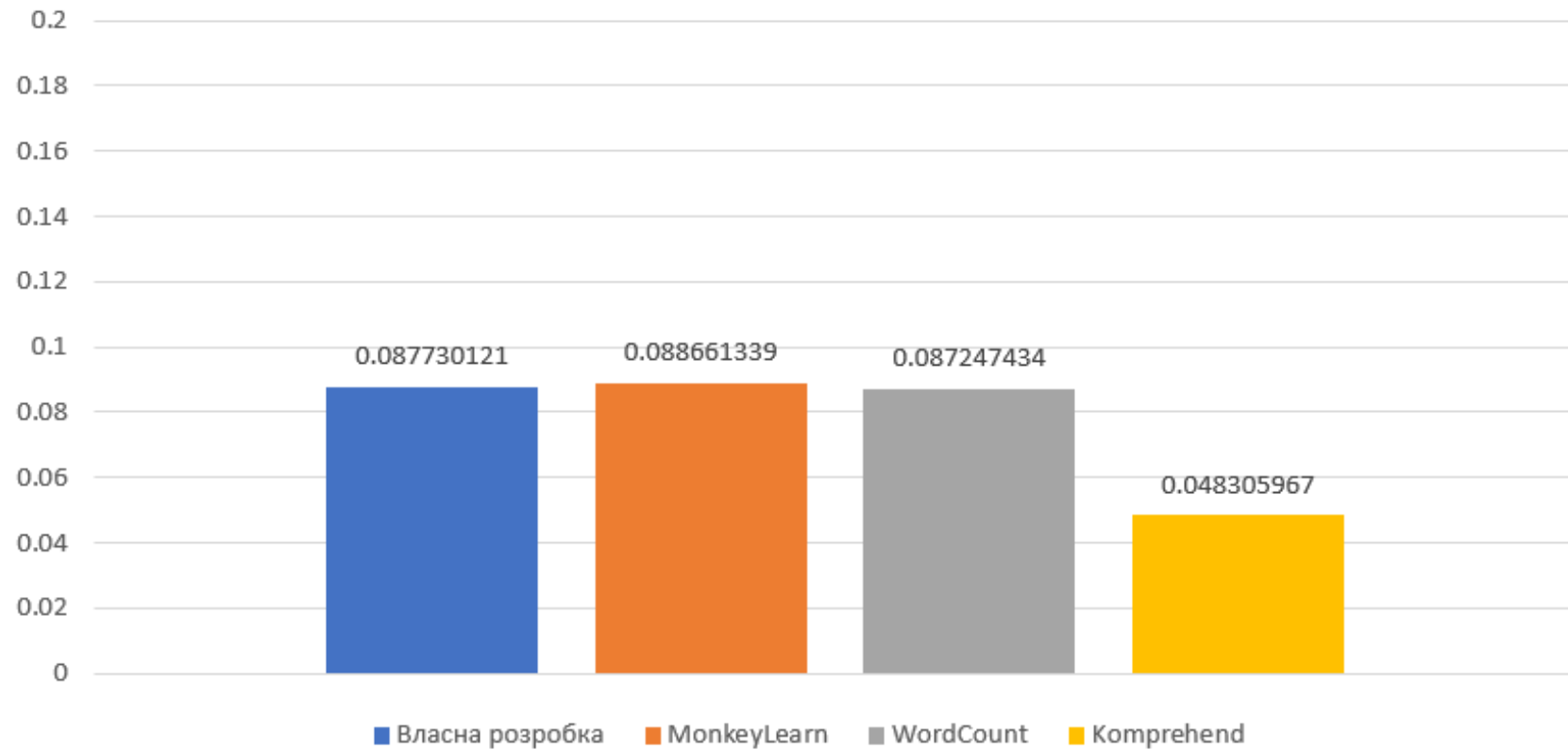
Абсолютна точність



Гістограма порівняння значень абсолютної точності пошуку ключових слів, отриманих розробленим програмним забезпеченням та існуючими аналогами, для текстів обсягом 150-400 слів

Бухаленков Д.О., група КП-21мн

Повнота за Жаккаром



Гістограма порівняння значень повноти пошуку ключових слів за Жаккаром, отриманих розробленим програмним забезпеченням та існуючими аналогами, для текстів обсягом 150-400 слів

Бухаленков Д.О., група КП-21мн

Додаток 3
Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО”



ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

**МОДИФІКОВАНИЙ МЕТОД ТА ПРОГРАМНЕ
ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНОГО
ПОШУКУ КЛЮЧОВИХ СЛІВ У
ПРИРОДНОМОВНИХ ТЕКСТОВИХ ДАНИХ**

Виконав: Бухаленков Дмитро Олександрович

Науковий керівник: доцент кафедри ПЗКС, к.т.н., доцент,

Заболотня Тетяна Миколаївна

Київ – 2024

АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ



- Покращення точності методів та засобів для пошуку ключових слів у текстах є важливим для задач повнотекстового пошуку, класифікації текстів, тегування, що використовується в пошукових системах.
- Досі не існує точно визначеного алгоритму однозначного визначення ключових слів людиною, тому створення автоматизованого методу є нетривіальною задачею.



ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ



Об'єкт дослідження: процес автоматизованої обробки природномовних текстових даних.

Предмет дослідження: методи, алгоритми та програмні засоби автоматизованого пошуку ключових слів у текстових даних.

ПОСТАНОВКА ЗАДАЧІ



Мета дослідження: підвищення ефективності пошуку ключових слів в тексті комп'ютерними системами автоматизованої обробки природномовних текстів за рахунок розробки та реалізації модифікованого гібридного методу автоматизованого пошуку ключових слів.

1. Проаналізувати існуючі методи та засоби автоматизованого пошуку ключових слів.
2. Сформулювати гіпотезу про покращення методу.
3. Розробити модифікований метод пошуку ключових слів.
4. Обґрунтувати вибір засобів для програмної реалізації.
5. Створити програмну реалізацію запропонованого методу.
6. Проаналізувати отримані результати та зробити висновки.

ІСНУЮЧІ МЕТОДИ



- **Статистичні** – використовують частоту вживання слова в тексті. **Переваги:** незалежність від мови та швидкодія. **Недоліки:** неточність, занадто великий обсяг знайдених ключових слів, непридатність до аналізу текстів малих розмірів.
- **На основі словників** – використовують зібрані словникові дані або тезауруси деякої тематики. **Переваги:** отримання досить точних результатів з невеликою кількістю вербального шуму з якісно взятим тезаурусом. **Недоліки:** потрібні якісні словникові дані з тематики.
- **Гібридні** – поєднання переваг статистичних та словникових методів.



СУЧАСНИЙ ГІБРИДНИЙ МЕТОД



«Гібридний метод» – запропонований в 2021 році українським спеціалістом О. Яхимовичем. Застосовує можливості сучасних програмних лінгвістичних пакетів для побудови розмітки тексту і аналізу слів.

Головна особливість методу – використання даних складних залежностей між парами слів та даних про частини мови, що отримуються за допомогою синтаксичного аналізатора.

Розробка збільшує у межах від 8,1% до 12,7% **повноту за метрикою Жаккара** та від 9,1% до 14,3% **абсолютну точність** пошуку ключових слів для англomовних текстів обсягом 140-1400 слів у порівнянні з аналогами.

СУЧАСНИЙ ГІБРИДНИЙ МЕТОД



- 1) Синтаксичний аналіз тексту і отримання даних про зв'язки між парами слів і частини мови, до яких належать слова тексту.
- 2) Фільтрування пар слів, зв'язки між якими належать до переліку неінформативних.
- 3) Заміна займенників в парах слів відповідними іменниками.
- 4) Відсіювання слів, які при синтаксичному аналізі було віднесено до неінформативних частин мови.
- 5) Фільтрування стоп-слів.
- 6) Визначення кількості зв'язків для кожного слова з пари.
- 7) Прийняття перших n слів з найбільшою кількістю зв'язків як ключових (де n - бажана кількість шуканих ключових слів).

НЕДОЛІК ГІБРИДНОГО МЕТОДУ



Суттєвий **недолік** – пошук лише **окремих ключових слів**.

Відсутність **багатослівних ключових термінів** негативно впливає на якість видачі релевантних результатів в пошукових системах.

Рішення: модифікація методу, що дозволить також шукати ключові терміни з кількох слів.

ТИПИ СИНТАКСИЧНИХ ЗВ'ЯЗКІВ В БАГАТОСЛІВНИХ ВИРАЗАХ



Universal Dependencies (стенфордська класифікація) – одна з найбільш відомих систем класифікацій синтаксичних зв'язків.

Інформацію про зв'язки типу **MWE (Multiword Expressions)** в тексті можливо використати для вирішення задачі пошуку ключових термінів.

	Nominals	Clauses	Modifier words	Function words
Core arguments	nsubj obj iobj	csbj ccomp xcomp		
Non-core dependents	obl vocative expl dislocated	advcl	advmod discourse	aux cop mark
Nominal dependents	nmod appos nummod	acl	amod	det clf case
Coordination	MWE	Loose	Special	Other
conj cc	fixed flat compound	list parataxis	orphan goeswith reparandum	punct room det

ТИПИ СИНТАКСИЧНИХ ЗВ'ЯЗКІВ В БАГАТОСЛІВНИХ ВИРАЗАХ



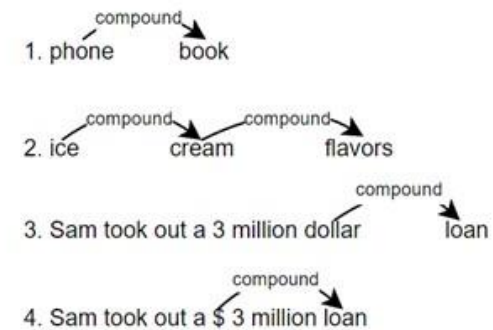
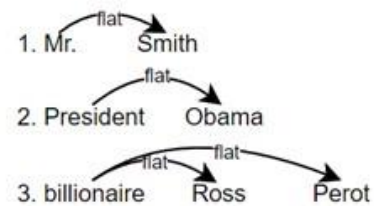
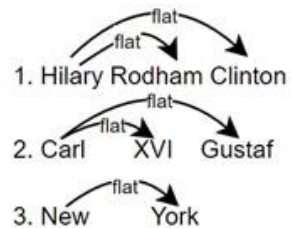
- **Fixed** (службові конструкції, фіксовані вирази, звороти)
- **Flat** (імена і дати)
- **Compound** (іменникові сполучення, дієслівні, прикметникові, їх комбінації)

Fixed – пов'язуючі функціональні частини речень, смислового навантаження самі по собі не несуть.

ЗАПРОПОНОВАНА МОДИФІКАЦІЯ МЕТОДУ



Для пошуку ключових термінів важливі **flat** та **compound**.



ЗАПРОПОНОВАНА МОДИФІКАЦІЯ МЕТОДУ



- 1) Синтаксичний аналіз тексту і отримання даних про зв'язки між парами слів і частини мови, до яких належать слова тексту.
- 2) **Отримання з тексту набору всіх виразів з типами зв'язків flat та compound.**
- 3) Фільтрування пар слів, зв'язки між якими належать до переліку неінформативних.
- 4) Заміна займенників в парах слів відповідними іменниками.
- 5) Відсіювання слів, які при синтаксичному аналізі було віднесено до неінформативних частин мови.
- 6) Фільтрування стоп-слів.
- 7) Визначення кількості зв'язків для кожного слова з пари.
- 8) Прийняття перших n слів з найбільшою кількістю зв'язків як ключові (де n - бажана кількість шуканих ключових слів).
- 9) **Фільтрація отриманих багатослівних виразів за допомогою попередньо отриманих ключових слів.**

СХЕМА ЕТАПІВ МОДИФІКОВАНОГО МЕТОДУ



ЗАСОБИ РЕАЛІЗАЦІЇ



Використані технології:

1. Мова програмування **Python**.
2. Лінгвістичний програмний пакет **NLTK**.
3. Допоміжний лінгвістичний програмний пакет **AllenNLP** на основі **spaCy**.
4. Бібліотека **JiWER** для обчислення метрик.



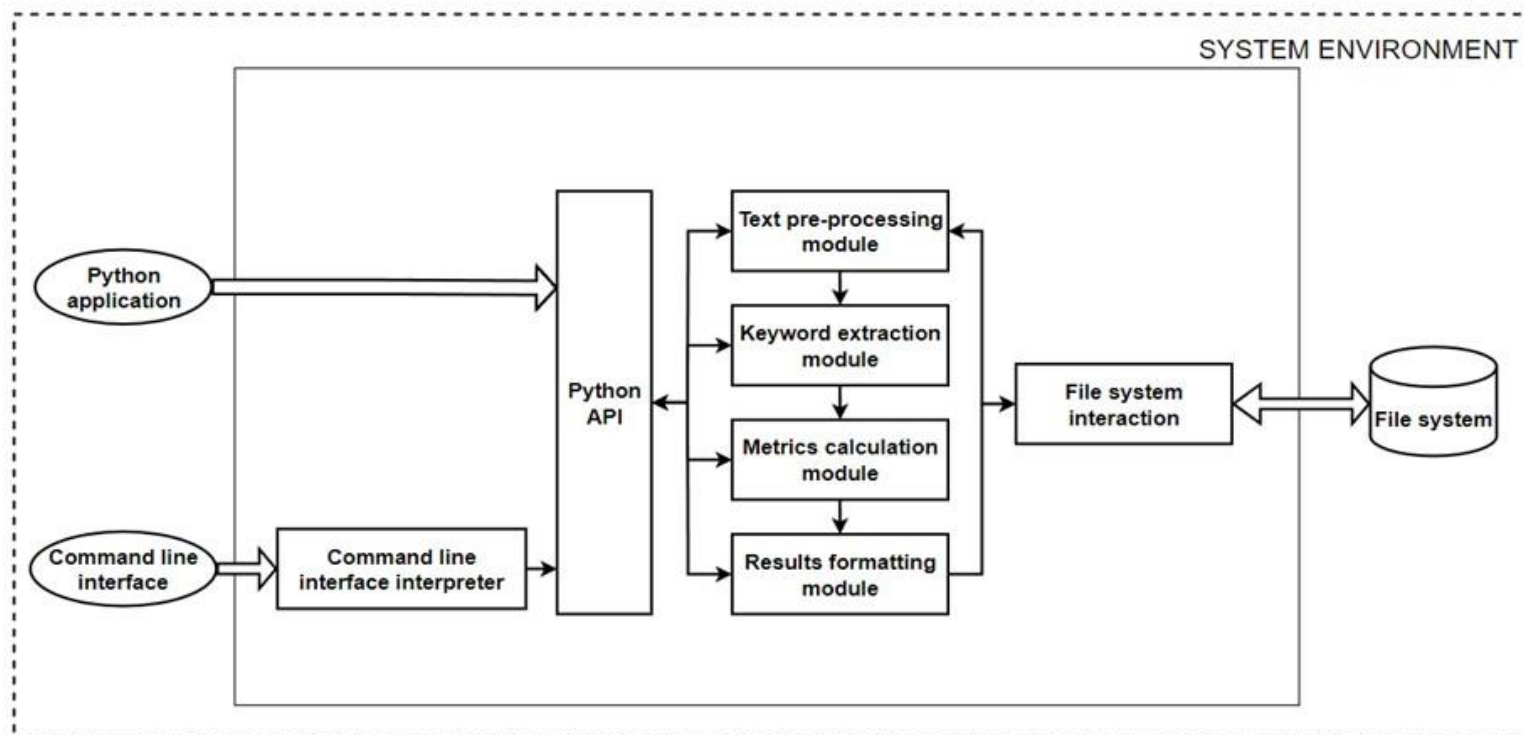
NLTK

AllenNLP

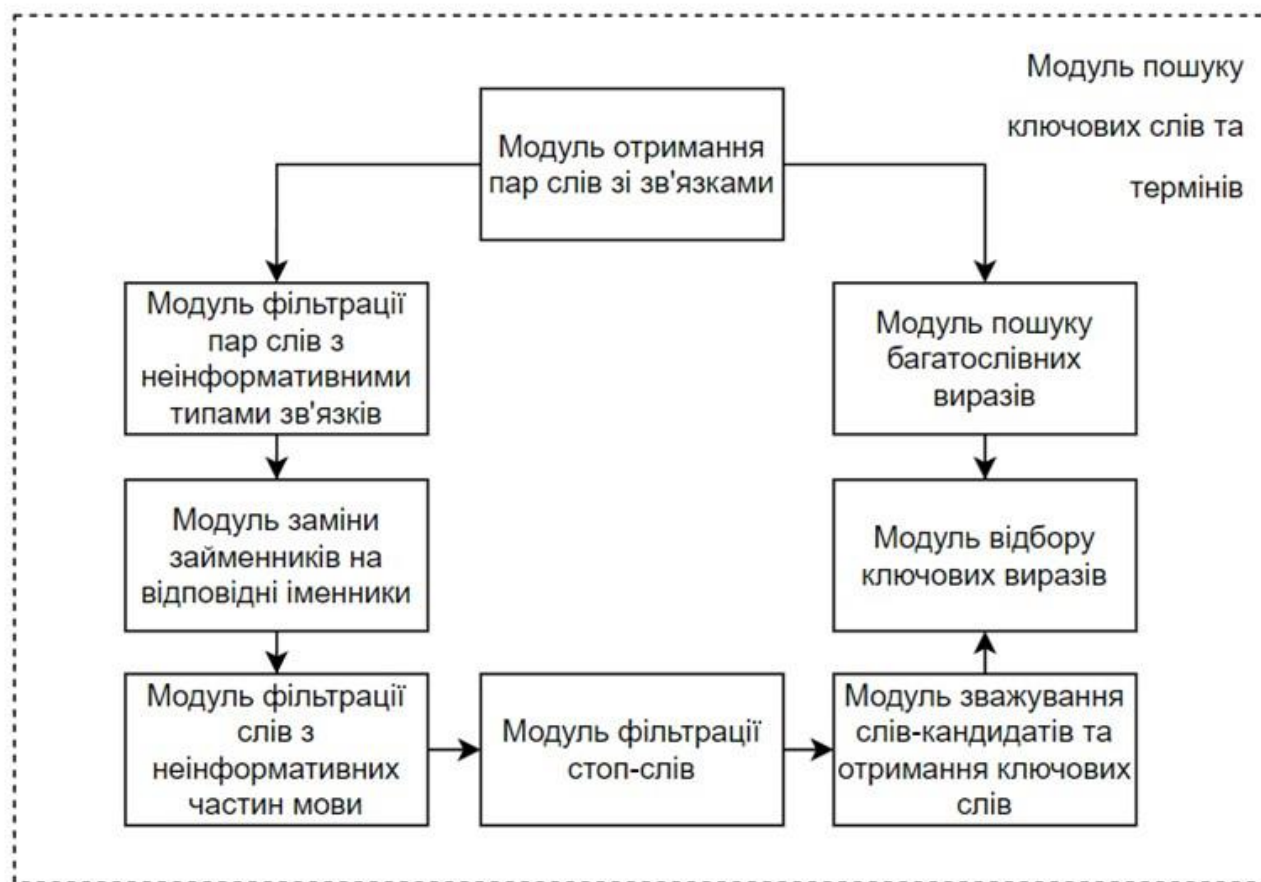


Реалізація у вигляді консольного додатку з підтримкою інтерфейсу командного рядка – за необхідності легке впровадження у комплексну інформаційно-пошукову систему

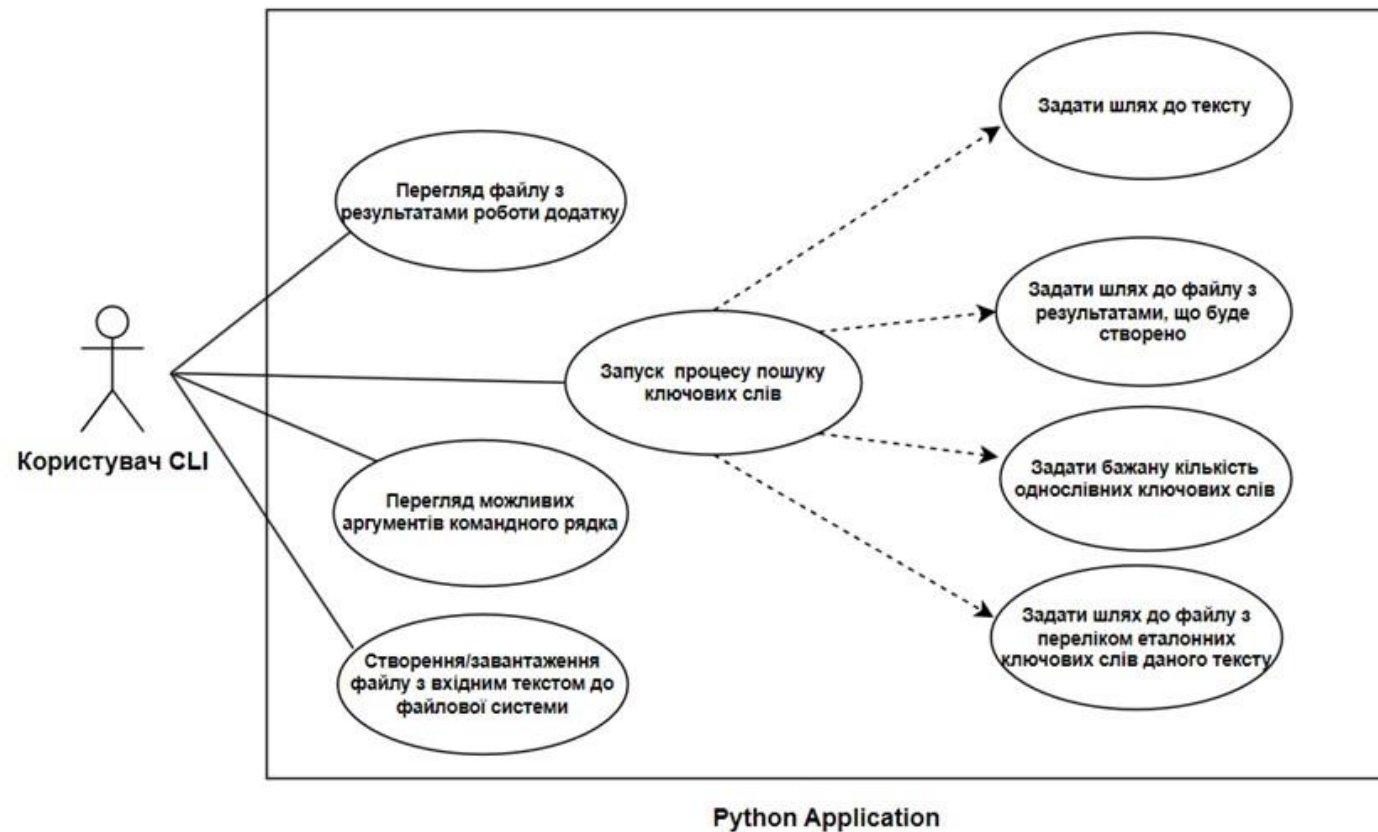
АРХІТЕКТУРА РЕАЛІЗОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



СТРУКТУРНА СХЕМА МОДУЛЯ ПОШУКУ КЛЮЧОВИХ СЛІВ ТА ТЕРМІНІВ



СЦЕНАРІЙ ВИКОРИСТАННЯ РОЗРОБЛЕНОГО ПЗ ЧЕРЕЗ CLI



АРГУМЕНТИ КОМАНДНОГО РЯДКА



Аргумент CLI	Допустимі значення	Опис
-i	Шлях до файлу	Файл, з якого буде зчитано вхідний текст. <i>Обов'язковий параметр.</i>
-o	Шлях до файлу, який буде створено	Файл з результатами, який буде створено після аналізу тексту. <i>Обов'язковий параметр.</i>
-n	Додатне ціле число	Кількість бажаних окремих ключових слів в результаті.
-r	Шлях до файлу	Файл з переліком еталонних ключових слів, якщо такі є. Використовується для перевірки точності роботи методу для тексту з наперед визначеними ключовими словами.

ПРИКЛАД ВИКОРИСТАННЯ ЧЕРЕЗ CLI



Запуск додатку за допомогою CLI:

```
(venv) PS C:\Users\3A43Mka\Documents\diplomma-testing> python .\main.py -i input.txt -o out.txt -n 8 -p .\reference.txt
```

Файл результатів зі знайденими ключовими словами:

```
File Edit Format View Help
vehicle, design, mass, launch, optimization, orbit, require, use
Micro Launch Vehicle, launch vehicle, vehicle design, Micro Vehicle design optimization, Air
Launch Vehicle, trajectory design, design variable, mass performance parameter, payload mass,
Air Launch, launch trajectory, step optimization cycle, trajectory optimization, target orbit,
insertion orbit, orbit specification
Absolute accuracy: 0.75
Jaccard completeness: 0.12
```

ВИКОРИСТАННЯ В ПРОГРАМНОМУ КОДІ В ЯКОСТІ БІБЛІОТЕКИ



```
from keyword_lib import get_keywords

text = "A conceptual control model for the Reynolds number test based on isentropic r
num_of_keywords = 11
reference = "Blowdown wind tunnel, Pressure control, Mach number control, Reynolds nu

result = get_keywords(text, num_of_keywords, reference)

print("Library returns an object with such attributes:")
print(result['one_word_keywords'])
print(result['multi_word_keywords'])
print(result['absolute_accuracy'])
print(result['jaccard_completeness'])
```

КІЛЬКІСНА ОЦІНКА РЕЗУЛЬТАТІВ



Абсолютна точність пошуку ключових слів:

$$a = \frac{n(A \cap B)}{n(A)}$$

де $n(A \cap B)$ – кількість правильно знайдених ключових слів;

$n(A)$ – кількість еталонних ключових слів.

Повнота пошуку за Жаккардом:

$$J = \frac{n(A \cap B)}{n(A) + n(B) - n(A \cap B)} = \frac{n(A \cap B)}{n(A \cup B)}$$

де $n(B)$ – кількість програмно знайдених ключових слів;

$n(A \cup B)$ – кількість елементів об'єднання обох множин.

ПОРІВНЯННЯ З ІСНУЮЧИМИ АНАЛОГАМИ



Порівняння розробленого програмного забезпечення проводилися з наступними існуючими сервісами, що надають подібні можливості з пошуку ключових слів у тексті:

1. MonkeyLearn на основі машинного навчання.
2. WordCount, що використовує аналіз N-грам.
3. Komprehend, що побудований на статистичному аналізі.

ПОРІВНЯННЯ З ІСНУЮЧИМИ АНАЛОГАМИ



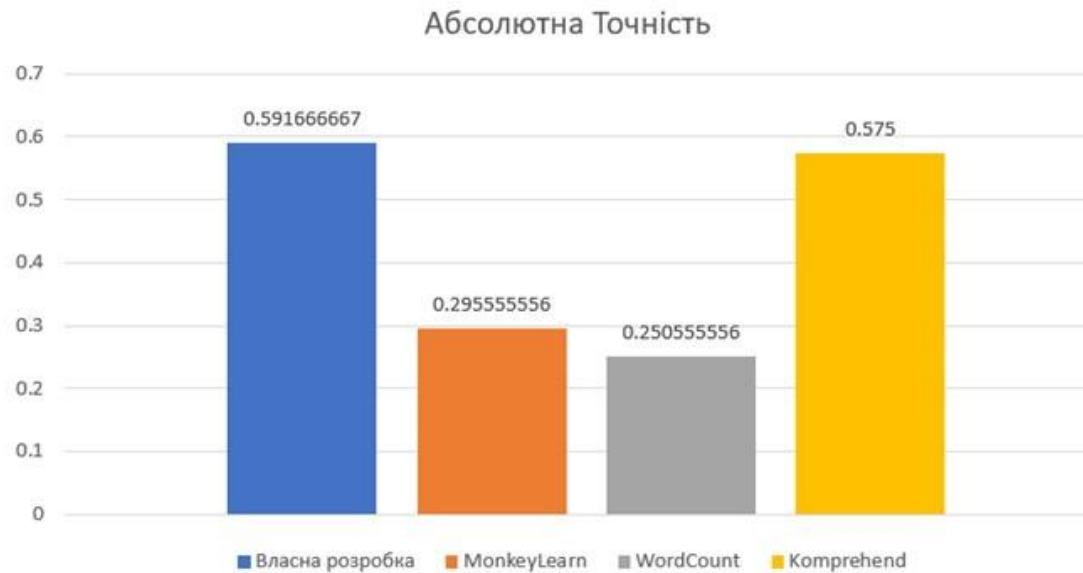
Порівняння значень **абсолютної точності** пошуку ключових слів для текстів обсягом 150-400 слів:



ПОРІВНЯННЯ З ІСНУЮЧИМИ АНАЛОГАМИ



Порівняння значень **абсолютної точності** пошуку ключових слів для текстів обсягом 1500-2500 слів:



ПОРІВНЯННЯ З ІСНУЮЧИМИ АНАЛОГАМИ



Порівняння значень **повноти пошуку** ключових слів за Жаккардом для текстів обсягом 150-400 слів:



ПОРІВНЯННЯ З ІСНУЮЧИМИ АНАЛОГАМИ



Порівняння значень **повноти пошуку** ключових слів за Жаккардом для текстів обсягом 1500-2500 слів:



НАУКОВА НОВИЗНА



Вперше запропоновано метод автоматизованого пошуку ключових слів, який враховує складні залежності між парами слів в тексті для визначення багатослівних виразів, що дозволяє знаходити ключові терміни, які складаються з кількох слів, та підвищує абсолютну точність пошуку ключових слів у межах від 6.2% до 13,6% в текстах обсягом 150-400 слів та у межах від 1.7% до 34,1% в текстах обсягом 1500-2500 слів у порівнянні з існуючими аналогами.

ПРАКТИЧНА ЗНАЧУЩІСТЬ



На основі запропонованого модифікованого методу виконано програмну реалізацію у середовищі Python за допомогою платформи PythonNLTK, та допоміжних пакетів AllenNLP та JiWER.

Проведені випробування створеної модифікації гібридного методу пошуку ключових термінів показали ефективність її використання для пошуку ключових термінів в текстах за критерієм абсолютної точності пошуку у порівнянні з існуючими аналогами.

Розроблена програмна реалізація запропонованого методу може бути використана у вигляді модуля комплексної пошукової системи при індексуванні текстових ресурсів.

АПРОБАЦІЯ



- Основні положення і результати роботи були представлені та обговорювалися на науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК 2023 (28-30 листопада 2023 р., м. Київ).
- Стаття «Модифікований метод пошуку ключових слів та термінів у текстових даних» в фаховому журналі «Проблеми програмування № 1 (2024)»
- Стаття «Дослідження ефективності модифікованого методу автоматизованого пошуку ключових слів у тексті» в фаховому журналі «Інформаційні технології та суспільство №1 (12) за 2024» (прийнято до друку, буде опубліковано 26 червня)

НАПРЯМКИ ПОДАЛЬШОЇ РОБОТИ



1. Провести додаткове тренування моделей, що використовуються в синтаксичних парсерах.
2. Використовувати синонімічні та тематичні словники для пошуку ключових термінів.
3. Зменшити кількість вербального шуму в результатах пошуку.
4. Провести більшу кількість досліджень на текстах різних тематик та стилю.
5. Додати підтримку пошуку ключових слів в текстах інших природних мов.



ВИСНОВКИ

1. Проаналізовано існуючі методи та засоби автоматизованого пошуку ключових слів.
2. Сформульовано гіпотезу про покращення методу, де пропонується аналіз даних про багатослівні вирази в тексті для пошуку ключових термінів, що складаються з кількох слів.
3. Розроблено модифікований метод пошуку ключових слів, який враховує складні залежності між парами слів в тексті для визначення багатослівних виразів, що дозволяє знаходити ключові терміни, які складаються з кількох слів.
4. Розроблено програмну реалізацію запропонованого модифікованого методу у вигляді додатку на мові Python з використанням сучасних лінгвістичних програмних пакетів та бібліотек для роботи з текстовими даними.
5. Запропонований метод дозволяє підвищити абсолютну точність пошуку ключових слів у межах від 6.2% до 13,6% в текстах обсягом 150-400 слів та у межах від 1.7% до 34,1% в текстах обсягом 1500-2500 слів, але має дещо меншу повноту за Жаккардом.
6. Визначено напрямки подальшого дослідження.



ДЯКУЮ ЗА УВАГУ!

Додаток 4
Лістинг програми

keyword_lib.py

```
from metrics import get_metrics
from nltk import pos_tag
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.parse.stanford import StanfordDependencyParser
from nltk.parse.corenlp import CoreNLPDependencyParser
from allennlp.predictors.predictor import Predictor
from nltk.corpus import stopwords
from jiwer import wer
import copy
import argparse
import json
from stanfordcorenlp import StanfordCoreNLP

import os

def traverse(deps, addr):
    dep = deps.get_by_address(addr)
    for d in dep['deps']:
        for addr2 in dep['deps'][d]:
            traverse(deps, addr2)

def traverse_and_write_to_list(deps, addr, list):
    dep = deps.get_by_address(addr)
    list.append(dep)
    for d in dep['deps']:
        for addr2 in dep['deps'][d]:
            traverse_and_write_to_list(deps, addr2, list)

def get_prases_from_deps(deps, last_index, lemmatized_words,
    accum_compounds): # getting these manually because triples doesn't give us
word addresses
    phrases = []
    for dep in deps:
        if dep['word'] is not None:
            if 'compound' in dep['deps']:
                current_compounds_list = []
                for compound_address in dep['deps']['compound']:
                    current_compounds_list.append(compound_address +
last_index)
                accum_compounds.append({'dep_where_found':
dep['address'] + last_index, 'compound_value': current_compounds_list})
            main_dep = None
            for i_dep in deps:
                if (i_dep['address'] == dep['head']) and (i_dep['word']
is not None):
                    main_dep = i_dep
                    break
            if main_dep is not None:
                new_phrase = {"main_word": {"word": main_dep['word'],
"address": main_dep['address'] + last_index, "pos": main_dep['tag'], "lemma":
```

```

lemmatized_words[main_dep['address'] + last_index - 1}}, "dependent_word":
{"word": dep['word'], "address": dep['address'] + last_index, "pos":
dep['tag'], "lemma": lemmatized_words[dep['address'] + last_index - 1}},
"dep_type": dep['rel']]
        phrases.append(new_phrase)
    return phrases

```

```

def lemmatize_words(lemmatizer, sentence):
    lemmatized_words = []
    tokens = word_tokenize(sentence)
    posed_tokend = pos_tag(tokens)
    for token in posed_tokend:
        if get_wordnet_pos(token[1]):
            lemmatized_word = lemmatizer.lemmatize(token[0],
pos=get_wordnet_pos(token[1]))
            lemmatized_words.append(lemmatized_word)
        else:
            lemmatized_words.append(token[0])
    return lemmatized_words

```

```

def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None

```

```

def get_counted_from_phrases(phrases):
    result = {}
    for phrase in phrases:
        if phrase["main_word"]["lemma"].lower() in result:
            result[phrase["main_word"]["lemma"].lower()] += 1
        else:
            result[phrase["main_word"]["lemma"].lower()] = 1
        if phrase["dependent_word"]["lemma"].lower() in result:
            result[phrase["dependent_word"]["lemma"].lower()] += 1
        else:
            result[phrase["dependent_word"]["lemma"].lower()] = 1
    return result

```

```

def get_counted_from_words(words):
    result = {}
    for word in words:
        if word["lemma"].lower() in result:
            result[word["lemma"].lower()] += 1
        else:
            result[word["lemma"].lower()] = 1
    return result

```

```

def get_coref_clusters(text):
    model_url = "https://storage.googleapis.com/allennlp-public-models/coref-spanbert-large-2020.02.27.tar.gz"
    predictor = Predictor.from_path(model_url)
    prediction = predictor.predict(document=text) # get prediction
    clusters = prediction['clusters']
    return clusters

def replace_pronouns(clusters, phrases):
    for cluster in clusters:
        address_of_original_word = cluster[0][1]
        original_word_lemma = None
        original_word_pos = None
        for phrase in phrases:
            if phrase["main_word"]["address"] == address_of_original_word
+ 1:
                original_word_lemma = phrase["main_word"]["lemma"]
                original_word_pos = phrase["main_word"]["pos"]
                break
            if phrase["dependent_word"]["address"] ==
address_of_original_word + 1:
                original_word_lemma = phrase["dependent_word"]["lemma"]
                original_word_pos = phrase["dependent_word"]["pos"]
                break
            for i in range(1, len(cluster)):
                address_of_word_to_replace = cluster[i][1]
                for phrase in phrases:
                    if phrase["main_word"]["address"] ==
address_of_word_to_replace + 1:
                        phrase["main_word"]["lemma"] = original_word_lemma
                        phrase["main_word"]["pos"] = original_word_pos
                    if phrase["dependent_word"]["address"] ==
address_of_word_to_replace + 1:
                        phrase["dependent_word"]["lemma"] =
original_word_lemma
                        phrase["dependent_word"]["pos"] = original_word_pos

def filter_by_dep_types(blacklisted_rels, phrases):
    filtered_phrases = []
    for phrase in phrases:
        if phrase["dep_type"].upper() not in blacklisted_rels:
            filtered_phrases.append(phrase)
    return filtered_phrases

def extract_list_of_words_from_phrases(phrases):
    list_of_words = []
    for phrase in phrases:
        list_of_words.append(phrase["main_word"])
        list_of_words.append(phrase["dependent_word"])
    return list_of_words

```

```

def filter_by_pos(blacklisted_pos, words):
    filtered_words = []
    for word in words:
        if word["pos"] not in blacklisted_pos:
            filtered_words.append(word)
    return filtered_words

def filter_by_stop_words(stop_words, words):
    filtered_words = []
    for word in words:
        if word["lemma"].lower() not in stop_words:
            filtered_words.append(word)
    return filtered_words

def get_lemma_by_address(phrases, address):
    for phrase in phrases:
        if phrase['main_word']['address'] == address:
            return phrase['main_word']
        if phrase['dependent_word']['address'] == address:
            return phrase['dependent_word']

def get_keywords(text, required_amnt_of_keywords, reference_file_text):
    java_path = "C:/Program Files/Java/jdk-17/bin/java.exe"
    os.environ['JAVAHOME'] = java_path

    path_to_jar = "C:/Users/3A43Mka/Downloads/stanford-parser-4.2.0/stanford-parser-full-2020-11-17/stanford-parser.jar"
    path_to_models_jar = "C:/Users/3A43Mka/Downloads/stanford-corenlp-4.2.0-models-english.jar"

    lemmatizer = WordNetLemmatizer()

    blacklisted_rels = ["CC", "DET", "EXPL", "FIXED", "PUNCT", "REF", "ROOT"]
    blacklisted_pos = ["CC", "CD", "DT", "EX", "IN", "LS", "MD", "PDT", "POS", "PRP", "PRP$", "RP", "SYM", "TO", "UH", "WDT", "WP", "WP$", "WRB", "-LRB-", "-RRB-"]

    stop_words = set(stopwords.words('english'))

    text = preprocess_text(text)

    phrases = [] # phrases list which contains main word, dependent word, their details and relation

    sentences = sent_tokenize(text)
    dependency_parser = StanfordDependencyParser(path_to_jar=path_to_jar, path_to_models_jar=path_to_models_jar)

    lastIndex = 0
    accumulated_compounds = []

    # getting lemmatized words from all text at once to get right indexes

```

```

    lemmatized_words_from_whole_text = lemmatize_words(lemmatizer, text)
    for sentence in sentences:
        phrases_from_current_sentence = [] # not used currently, doesn't
have word addresses
        result = dependency_parser.raw_parse(sentence)
        dep = result.__next__()
        triples = list(dep.triples())
        for thing in triples:
            phrase = {"main_word": {"word": thing[0][0], "pos":
thing[0][1]},
                    "dependent_word": {"word": thing[2][0], "pos":
thing[2][1]}, "dep_type": thing[1]}
            phrases_from_current_sentence.append(phrase)
            dep_traversed = []
            traverse_and_write_to_list(dep, 0, dep_traversed)
            current_phrases_from_traversed =
get_phrases_from_deps(dep_traversed, lastIndex,
lemmatized_words_from_whole_text, accumulated_compounds)
            words = word_tokenize(sentence)
            new_last_index = 0
            for word_i, word in enumerate(words, start=1 + lastIndex):
                new_last_index = word_i
            phrases = phrases + current_phrases_from_traversed
            lastIndex = new_last_index

    intact_phrases = copy.deepcopy(phrases)
    print("Word pairs:")
    print(phrases)
    clusters = get_coref_clusters(text)
    print("Clusters:")
    print(clusters)
    counted1 = get_counted_from_phrases(phrases)
    print("counted1:")
    print(counted1)

    phrases = filter_by_dep_types(blacklisted_rels, phrases)
    print("Pairs after filtering by dependency types:")
    print(phrases)
    counted2 = get_counted_from_phrases(phrases)
    print("counted2:")
    print(counted2)

    replace_pronouns(clusters, phrases)
    print("Pairs after pronouns are replaced:")
    print(phrases)
    counted3 = get_counted_from_phrases(phrases)
    print("counted3:")
    print(counted3)

    list_of_words = extract_list_of_words_from_phrases(phrases)
    print("Phraes are now converted into separate words")
    print(list_of_words)

    filtered_words = filter_by_pos(blacklisted_pos, list_of_words)
    print("Words after filtering by pos types")
    print(filtered_words)

```

```

counted4 = get_counted_from_words(filtered_words)
print("counted4:")
print(counted4)

filtered_by_stop_words = filter_by_stop_words(stop_words,
filtered_words)
print("Words after filtering by stop words")
print(filtered_by_stop_words)

counted5 = get_counted_from_words(filtered_by_stop_words)
print("counted5:")
print(counted5)

# sorting by number of rels

counted5_sorted = dict(sorted(counted5.items(), key=lambda item:
item[1], reverse=True))
print("counted5 sorted:")
print(counted5_sorted)

print("filtered_by_stop_words:")
print(filtered_by_stop_words)
print("phrases:")
print(phrases)

# now we will check if we need to sort out the collision problem

sorted_words_with_freqs_list = []

for key in counted5_sorted:
    sorted_words_with_freqs_list.append({"lemma": key, "freq":
counted5_sorted[key]})

print("sorted_words_with_freqs_list:")
print(sorted_words_with_freqs_list)

words_after_collision = []

if len(sorted_words_with_freqs_list) <= required_amnt_of_keywords:
    print("We don't have a collision because number of candidates is
<= required keywords")
    for word in sorted_words_with_freqs_list:
        words_after_collision.append(word['lemma'])

else:
    if sorted_words_with_freqs_list[required_amnt_of_keywords -
1]["freq"] == \
        sorted_words_with_freqs_list[required_amnt_of_keywords -
1]["freq"]:
        problematic_frequency =
sorted_words_with_freqs_list[required_amnt_of_keywords - 1]["freq"]
        guaranteed_keywords_before_collision = []
        for word in sorted_words_with_freqs_list:
            if word["freq"] == problematic_frequency:
                break

```

```

guaranteed_keywords_before_collision.append(word["lemma"])
words_with_problematic_frequency = []
for word in sorted_words_with_freqs_list:
    if word["freq"] == problematic_frequency:

words_with_problematic_frequency.append(word['lemma'])
print("guaranteed_keywords_before_collision:")
print(guaranteed_keywords_before_collision)
print("words_with_problematic_frequency:")
print(words_with_problematic_frequency)
number_of_words_left = required_amnt_of_keywords -
len(guaranteed_keywords_before_collision)
print("number_of_words_left to select:")
print(number_of_words_left)
problematic_rated_list = []
for problematic_word in words_with_problematic_frequency:
    current_freq = 0
    for phrase in phrases:
        if phrase["main_word"]["lemma"].lower() ==
problematic_word:
            if phrase["dependent_word"]["lemma"].lower() in
guaranteed_keywords_before_collision:
                current_freq += 1
            if phrase["dependent_word"]["lemma"].lower() ==
problematic_word:
                if phrase["main_word"]["lemma"].lower() in
guaranteed_keywords_before_collision:
                    current_freq += 1
                problematic_rated_list.append({"lemma":
problematic_word, "freq": current_freq})
print("Newly ranked problematic words:")
print(problematic_rated_list)
sorted_ranked_problematic = sorted(problematic_rated_list,
key=lambda w: w["freq"], reverse=True)
print("Sorted ranked problematic words:")
print(sorted_ranked_problematic)
words_after_collision += guaranteed_keywords_before_collision
count = 0
while count < number_of_words_left:

words_after_collision.append(sorted_ranked_problematic[count]["lemma"])
count += 1
else:
    print("We don't have a collision, we just need to take only
required amount and we are done")
    for word in sorted_words_with_freqs_list:
        if len(words_after_collision) ==
required_amnt_of_keywords:
            break
        words_after_collision.append(word['lemma'])

print("And so the final list of keywords:")
print(words_after_collision)

all_compounded_terms = []

```

```

        for cmp in accumulated_compounds:
            list_of_words_addresses = cmp['compound_value'] +
[cmp['dep_where_found']]
            word = []
            for address in list_of_words_addresses:
                word.append(get_lemma_by_address(intact_phrases,
address)['lemma'])
            all_compounded_terms.append(word)

        print("all_compounded_terms:")
        print(all_compounded_terms)

        filtered_compounds = []

        for word in words_after_collision:
            for compound in all_compounded_terms:
                if word.upper() in map(str.upper, compound):
                    filtered_compounds.append(compound)

        list_of_joined_compounds = []
        for compound in filtered_compounds:
            joined = " ".join(compound)
            if joined not in list_of_joined_compounds:
                list_of_joined_compounds.append(joined)

        print("So the filtered compounds:")
        print(list_of_joined_compounds)

        absolute_accuracy = 0
        jaccard_completeness = 0

        result_to_return = {}
        result_to_return['one_word_keywords'] = words_after_collision
        result_to_return['multi_word_keywords'] = list_of_joined_compounds

        if len(reference_file_text):
            all_keywords = words_after_collision + list_of_joined_compounds
            reference_keywords = [x.strip() for x in
reference_file_text.split(',')]
            metric_results = get_metrics(reference_keywords, all_keywords)
            absolute_accuracy = metric_results['absolute_accuracy']
            jaccard_completeness = metric_results['jaccard_completeness']

            result_to_return['absolute_accuracy'] = absolute_accuracy
            result_to_return['jaccard_completeness'] = jaccard_completeness

            print(f'Absolute accuracy: {absolute_accuracy}')
            print(f'Jaccard completeness: {jaccard_completeness}')

        return result_to_return

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Keyword extractor.')
    parser.add_argument("-i", "--input", help="Input file path",
required=True)

```

```

        parser.add_argument("-o", "--output", help="Output file path",
required=True)
        parser.add_argument("-n", "--number_of_keywords", help="Set number of
required keywords", type=int, default=9999)
        parser.add_argument("-r", "--reference", help="File with reference
keywords to compare", default=False)

    args = parser.parse_args()

    file = open(args.input)
    text = ' '.join(file.readlines())
    file.close()
    print(text)

    required_amnt_of_keywords = args.number_of_keywords
    reference_file_text = ""

    if args.reference:
        reference_file = open(args.reference)
        reference_file_text = ' '.join(reference_file.readlines())
        reference_file.close()
    print(reference_file_text)

    keyword_results = get_keywords(text, required_amnt_of_keywords,
reference_file_text)

    file1 = open(args.output, 'w')
    file1.write(', '.join(keyword_results['one_word_keywords']))
    file1.write("\n")
    file1.write(', '.join(keyword_results['multi_word_keywords']))
    if args.reference:
        file1.write("\n")
        file1.write(f'Absolute accuracy:
{keyword_results["absolute_accuracy"]}')
        file1.write("\n")
        file1.write(f'Jaccard completeness:
{keyword_results["jaccard_completeness"]}')
    file1.close()

```

metrics.py

```

from jiwer import wer
import sys

def get_metrics(references, found):
    correct_compounds = []
    correct_ref_indexes = set()
    for compound in found:
        for ref_ind, ref_word in enumerate(references):
            if wer(ref_word.lower(), compound.lower()) < 1/3 + 0.01:
                print(f'wer says that "{compound}" corresponds to
"{ref_word}"')
                correct_ref_indexes.add(ref_ind)
            if compound not in correct_compounds:
                correct_compounds.append(compound)

```

```

    print("correct compounds: ", correct_compounds)
    print('correct ref indexes:', correct_ref_indexes)
    absolute_accuracy = len(correct_ref_indexes) / len(references)
    jaccard_completeness = len(correct_ref_indexes) / (len(found) +
len(references) - len(correct_ref_indexes))
    return {'absolute_accuracy': absolute_accuracy,
'jaccard_completeness': jaccard_completeness, 'correct_compounds':
correct_compounds, 'correct_ref_indexes': correct_ref_indexes }

```

```

def main():
    reference_file = open('reference.txt')
    reference_file_text = ' '.join(reference_file.readlines())
    reference_file.close()
    reference_keywords = [x.strip() for x in
reference_file_text.split(',')]

    found_file = open('found_test.txt')
    found_file_text = ' '.join(found_file.readlines())
    found_file.close()
    found_keywords = [x.strip() for x in found_file_text.split(',')]

    result = get_metrics(reference_keywords, found_keywords)

    absolute_accuracy = result['absolute_accuracy']
    jaccard_completeness = result['jaccard_completeness']
    print(f'Absolute accuracy: {absolute_accuracy}')
    print(f'Jaccard completeness: {jaccard_completeness}')

if __name__ == "__main__":
    main()

```