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

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.91

«До захисту допущено»
Завідувач кафедри
_____Євгенія СУЛЕМА
«___»_____2025 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-науковою програмою
«Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»
зі спеціальності 121 Інженерія програмного забезпечення
на тему: «Модифікований метод косинусної подібності для виявлення нечітких дублікатів у текстових даних»

Виконав:
студент ІІ курсу, групи КП-31мн
Козинець Назарій Вікторович _____

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:
доцент кафедри СПіСКС, к.т.н., доцент
Тарасенко-Клятченко Оксана Володимирівна _____

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

Київ – 2025 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
«___»_____2023 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Козинцю Назарію Вікторовичу

1. Тема дисертації «Модифікований метод косинусної подібності для виявлення нечітких дублікатів у текстових даних», науковий керівник дисертації Заболотня Тетяна Миколаївна, к.т.н., доцент, затверджені наказом по університету № 1219-С від «21» березня 2025 р.
2. Термін подання студентом дисертації «16» травня 2025 р.
3. Об'єкт дослідження: процес автоматизованого виявлення нечітких дублікатів в текстових даних.
4. Предмет дослідження: методи, алгоритми та програмні засоби для автоматизованого виявлення нечітких дублікатів у текстових даних.
5. Перелік завдань, які потрібно розробити:
   - дослідити наявні методи виявлення нечітких дублікатів у текстових даних;
   - порівняти ефективність даних методів;
   - розробити модифікований метод косинусної подібності для виявлення нечітких дублікатів у 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:
   - графічне представлення алгоритму роботи модифікованого методу
   - архітектура розробленого програмного забезпечення модифікованого методу косинусної подібності
   - структура баз даних модифікованого методу косинусної подібності для виявлення нечітких дублікатів у текстових даних

- UML діаграма сценаріїв використання розробленого програмного забезпечення для виявлення нечітких дублікатів в тексті користувачами
- візуальний процес векторного представлення текстових даних
- візуальне представлення роботи програмного забезпечення класичного та модифікованого методу.

7. Перелік публікацій:
   - XVII наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг» (ПМК-2024).
   - 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., к.т.н., доцент		

9. Дата видачі завдання «10» жовтня 2023 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ознайомлення з предметною галуззю	29.10.2023	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	13.12.2023	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	20.02.2024	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	08.06.2024	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження; підготовка матеріалів доповіді на конференції ПМК-2024	19.09.2024	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	17.06.2024	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу	08.03.2025	
8.	Оформлення текстової і графічної частини магістерської дисертації	11.04.2025	

Студент Назарій КОЗИНЕЦЬ

Науковий керівник Тетяна ЗАБОЛОТНЯ

РЕФЕРАТ

Актуальність теми. Проблема автоматизованого виявлення нечітких дублікатів у текстових даних набуває особливої актуальності в умовах стрімкого зростання обсягів цифрової інформації. Традиційні методи пошуку дублікатів демонструють низьку ефективність при роботі з текстами, що мають спільний семантичний зміст, але відрізняються лексичним складом, синтаксисом чи структурою.

З розвитком мережі Інтернет та цифрових бібліотек щодня з'являються мільйони нових текстових документів, серед яких значна частка є перефразованими копіями існуючих матеріалів. Виявлення таких дублікатів є критично важливим для систем перевірки плагіату, пошукових алгоритмів, електронних бібліотек та наукових репозиторіїв.

Існуючі методи, зокрема TF-IDF, Jaccard Similarity та Levenshtein Similarity, мають суттєві обмеження: вони ефективні для виявлення точних або майже точних копій, але втрачають чутливість при семантичному перефразуванні, зміні порядку речень та синонімічних замінах. Тому розроблення модифікованого методу косинусної подібності, що враховує контекстуальні особливості тексту, є актуальним завданням для підвищення якості систем дедуплікації документів та контент-фільтрації.

Об’єктом дослідження є процес автоматизованого виявлення нечітких дублікатів в текстових даних.

Предметом дослідження є методи, алгоритми та програмні засоби автоматизованого виявлення нечітких дублікатів в текстових даних.

Мета роботи: підвищення точності у виявлені нечітких дублікатів у текстових даних шляхом розроблення та реалізації модифікованого методу косинусної подібності, що поєднує переваги сучасних векторних представлень із контекстно-залежним зважуванням компонент.

Наукова новизна. Удосконалено метод косинусної подібності для виявлення нечітких дублікатів у текстових даних, характерною особливістю

якого є поєднання контекстуального зважування координат векторного представлення речень на основі їхньою інформаційної значущості та статистично визначених α-ваг для підвищення чутливості до семантичних перефразувань, що дозволяє підвищити метрики F1 на 5-6% у порівнянні з класичним косинусним методом та на 10-14% у порівнянні з традиційними частотними методами.

Практична значущість. В рамках даного магістерського дослідження створено програмне забезпечення на основі модифікованого методу косинусної подібності для виявлення нечітких дублікатів у текстових даних, використовуючи Python та бібліотеки NLTK, SentenceTransformer. Розроблене рішення може бути інтегроване як модуль у системи перевірки оригінальності текстів, пошукові системи, електронні бібліотеки, освітні платформи та корпоративні системи керування документами, що підвищить якість обслуговування користувачів та точність виявлення дублікатів у різних сценаріях використання.

Апробація роботи. Основні положення і результати даної роботи були представлені на XVII науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг ПМК-2024» (Київ, 20-22 листопада 2024р.).

Стаття "Hybrid detection of fuzzy duplicate texts: cosine similarity and transformers" в фаховому журналі «Прикладні аспекти інформаційних технологій Том 8 № 1 (2025)».

Стаття «Вплив комбінованих векторних представлень на точність пошуку нечітких дублікатів» в фаховому журналі «Наукові праці ВНТУ №1 (2025)».

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

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

У першому розділі проаналізовано існуючі методи та технології виявлення нечітких дублікатів текстових даних. Розглянуто поняття "нечіткий дублікат" і фактори, що впливають на семантичну варіативність тексту. Досліджено еволюцію косинусної подібності від поверхневих частотних моделей до контекстно-залежних векторних представлень. Проведено порівняльний аналіз ефективності різних алгоритмів виявлення подібності, визначено їх переваги та обмеження при роботі з перефразованими текстами. Також наведено огляд комерційних та відкритих програмних рішень з функціональними можливостями для пошуку нечітких дублікатів.

У другому розділі запропоновано модифікований метод косинусної подібності для виявлення нечітких дублікатів у текстових даних. Детально проаналізовано класичну косинусну міру та її обмеження при роботі з текстами різної довжини та стилю. Описано розроблений цикл попереднього оброблення тексту, що включає лінгвістичну нормалізацію та векторизацію речень. Представлено оригінальний підхід до контекстуального зважування компонент вектора, який дозволяє підвищити чутливість методу до семантично еквівалентних, але лексично відмінних текстів. Обґрунтовано критерії та правила прийняття рішення про наявність нечіткого дублювання.

У третьому розділі наведено опис технологій, що були використані для програмної реалізації запропонованого методу, та обґрунтовано їх вибір. Висвітлено особливості побудови клієнт-серверної архітектури системи та механізми взаємодії між компонентами. Розглянуто процес оброблення та порівняння документів, методи зберігання векторних представлень та організацію пошуку схожих фрагментів. Представлено користувацький інтерфейс програмного забезпечення та описано сценарії його використання для різних задач пошуку текстових дублікатів.

У четвертому розділі проведено експериментальний аналіз ефективності розробленого методу на наборі тестових даних. Обґрунтовано вибір метрик оцінювання таких як точність, повнота та F1-міра. Проведено порівняльне дослідження класичного та модифікованого методів косинусної подібності на

різних типах текстових корпусів. Проаналізовано вплив параметрів системи на якість виявлення дублікатів та запропоновано оптимальні налаштування для різних сценаріїв використання. Визначено напрямки подальшого вдосконалення розробленого методу та програмного забезпечення.

У висновках проаналізовано отримані результати дослідження.

Ключові слова: інженерія програмного забезпечення, нечіткі дублікати, косинусна подібність, семантичні векторні представлення, семантична близькість, оброблення природної мови, виявлення плагіату.

ABSTRACT

Actuality. The problem of automated detection of fuzzy duplicates in text data is of particular relevance in the context of the rapid growth of digital information. Traditional methods of duplicate detection demonstrate low efficiency when dealing with texts that have common semantic content but differ in lexical composition, syntax, or structure.

With the development of the Internet and digital libraries, millions of new text documents are created every day, and a significant proportion of them are paraphrased copies of existing materials. Detecting such duplicates is critical for plagiarism checkers, search algorithms, digital libraries, and academic repositories.

Existing methods, such as TF-IDF, Jaccard Similarity, and Levenshtein Similarity, have significant limitations: they are effective for detecting exact or near-exact copies, but lose sensitivity in the case of semantic paraphrasing, sentence reordering, and synonymous substitutions. Therefore, the development of a modified cosine similarity method that takes into account the contextual features of the text is an urgent task to improve the quality of document deduplication and content filtering systems.

Object of research is the process of automated detection of fuzzy duplicates in text data.

Subject of research is to improve the accuracy of detecting fuzzy duplicates in text data by developing and implementing a modified cosine similarity method that combines the advantages of modern vector representations with context-dependent component weighting.

Goal of the work is to improve the accuracy and efficiency of detecting fuzzy duplicates in text data by developing and implementing a modified cosine similarity method that combines the advantages of modern vector representations with context-dependent component weighting.

The scientific novelty: The cosine similarity method has been improved for detecting fuzzy duplicates in text data, characterized by combining contextual

weighting of vector representation coordinates of sentences based on their informational significance and statistically determined α-weights to increase sensitivity to semantic paraphrasing, which allows for improving F1 metrics by 5-6% compared to the classic cosine method and by 10-14% compared to traditional frequency methods.

The practical value: As part of this master's research, software based on a modified cosine similarity method for detecting fuzzy duplicates in text data was created using Python and NLTK and SentenceTransformer libraries. The developed solution can be integrated as a module into text originality checking systems, search engines, digital libraries, educational platforms and corporate document management systems, which will improve the quality of user service and the accuracy of duplicate detection in various use cases.

Approbation. The main provisions and results of this work were presented at the XVII scientific conference of master's and postgraduate students "Applied Mathematics and Computing PMC-2024" (Kyiv, November 22, 2024). Article "Hybrid detection of fuzzy duplicate texts: cosine similarity and transformers" in a professional journal "Applied aspects of information technology Vol 8 № 1 (2025)". Article «The impact of combined vector representations on the accuracy of fuzzy duplicate search» in a professional journal "Scientific Works of Vinnytsia National Technical University №1 (2025)".

Structure and content. The master's thesis consists of an introduction, four chapters, conclusions and appendices.

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

The first chapter analyses the existing methods and technologies for detecting fuzzy duplicates of text data. The concept of 'fuzzy duplicate' and the factors influencing the semantic variability of the text are considered. The evolution of cosine similarity from surface frequency models to context-dependent vector representations is investigated. A comparative analysis of the effectiveness of various similarity detection algorithms is carried out, their advantages and

limitations when working with paraphrased texts are identified. An overview of commercial and open-source software solutions with functionalities for searching for fuzzy duplicates is also provided.

The second chapter proposes a modified cosine similarity method for detecting fuzzy duplicates in text data. The classical cosine measure and its limitations when working with texts of different length and style are analysed in detail. The developed text preprocessing cycle, which includes linguistic normalisation and sentence vectorisation, is described. An original approach to the contextual weighting of vector components is presented, which allows increasing the sensitivity of the method to semantically equivalent but lexically different texts. The criteria and rules for deciding on the presence of fuzzy duplication are substantiated.

The third chapter describes the technologies used for the software implementation of the proposed method and justifies their choice. The peculiarities of building the client-server architecture of the system and the mechanisms of interaction between the components are highlighted. The process of processing and comparing documents, methods of storing vector representations and organising the search for similar fragments are considered. The user interface of the software is presented and scenarios of its use for various tasks of searching for text duplicates are described.

The fourth chapter provides an experimental analysis of the effectiveness of the developed method on a set of test data. The choice of evaluation metrics, such as accuracy, completeness and F1-measure, is substantiated. A comparative study of the classical and modified cosine similarity methods on different types of text corpora is carried out. The influence of system parameters on the quality of duplicate detection is analysed and optimal settings for different usage scenarios are proposed. The directions of further improvement of the developed method and software are determined.

The conclusions analyse the results of the study.

Keywords: software engineering, fuzzy duplicates, cosine similarity, semantic vector representations, semantic proximity, natural language processing, plagiarism detection.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

AI – Штучний інтелект (від англ. Artificial Intelligence) – галузь комп'ютерних наук, що займається створенням інтелектуальних машин, здатних виконувати завдання, які зазвичай вимагають людського інтелекту.

API – Інтерфейс програмування застосунків (від англ. Application Programming Interface) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення, що дозволяє різним додаткам обмінюватися даними між собою.

UI – Користувацький інтерфейс (від англ. User Interface) – засоби та методи взаємодії людини з комп'ютерною системою, включаючи відображення інформації на екрані та способи введення команд користувачем.

UX – Досвід користувача (від англ. User Experience) – сукупність вражень та емоцій, які отримує користувач під час взаємодії з програмним продуктом, що включає зручність використання, доступність та загальне задоволення від роботи з системою.

BERT – Bidirectional Encoder Representations from Transformers (Двонаправлені енкодерні представлення з трансформерів). Це модель машинного навчання для обробки природної мови, розроблена Google у 2018 році. BERT використовує двонаправлений підхід для кращого розуміння контексту слів у реченні.

ASGI – Asynchronous Server Gateway Interface (Асинхронний серверний шлюзовий інтерфейс). Це специфікація, що описує стандартний інтерфейс між вебсерверами та асинхронними Python вебдодатками або фреймворками. ASGI є наступником WSGI (Web Server Gateway Interface), але з підтримкою асинхронних операцій.

SBERT – Sentence BERT – модифікація нейромережевої моделі BERT, оптимізована для створення семантичних вкладень (embeddings) речень, що дозволяє ефективно порівнювати та класифікувати текстові дані.

VSM – Векторна модель простору (від англ. Vector Space Model) – математична модель представлення текстових документів як векторів у багатовимірному просторі, що використовується для інформаційного пошуку та обробки природної мови.

TF-IDF – Частота терміну-зворотна частота документу (від англ. Term Frequency–Inverse Document Frequency) – статистична міра, що використовується для оцінки важливості слова у контексті документа, який є частиною колекції документів.

LMS – Система управління навчанням (від англ. Learning Management System) – програмний додаток для адміністрування, документації, відстеження, звітності та надання навчальних курсів, тренінгових програм або навчальних програм.

ETL – Видобування, перетворення, завантаження (від англ. Extract, Transform, Load) – процес в управлінні даними, що передбачає вилучення даних з різних джерел, їх перетворення у відповідний формат та завантаження у цільове сховище даних.

NLTK – Набір інструментів природної мови (від англ. Natural Language Toolkit) – бібліотека програм для символьної та статистичної обробки природної мови для мови програмування Python.

ORM – Об'єктно-реляційне відображення (від англ. Object-Relational Mapping) – технологія програмування, що зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи "віртуальну об'єктну базу даних".

СКБД – Система керування базами даних – комплекс програмного забезпечення, що забезпечує управління створенням, обслуговуванням та використанням баз даних.

DOM – Об'єктна модель документа (від англ. Document Object Model) – програмний інтерфейс, який дозволяє програмам та скриптам динамічно отримувати доступ та оновлювати вміст, структуру і стиль документів, зокрема вебсторінок.

ВСТУП

В епоху інформаційного суспільства проблема автоматизованого виявлення нечітких дублікатів у текстових даних набуває особливої актуальності. Постійне зростання обсягів текстової інформації в цифровому просторі – від наукових публікацій та освітніх матеріалів до корпоративних документів і користувацького контенту – створює потребу в ефективних методах ідентифікації семантично еквівалентних фрагментів, які відрізняються за формою, але не за змістом.

На відміну від точних дублікатів, які легко виявляються через побітове порівняння, нечіткі дублікати характеризуються варіативністю лексичних, синтаксичних та стилістичних засобів вираження однакового змісту. Перефразування, синонімічні заміни, зміна порядку слів, морфологічні трансформації – усе це ускладнює завдання автоматичного розпізнавання таких текстів традиційними методами.

Проблема виявлення нечітких дублікатів є актуальною в широкому спектрі завдань: перевірка академічних робіт на оригінальність, виявлення плагіату, дедуплікація документів у пошукових системах, фільтрація спаму, аналіз відгуків споживачів, консолідація інформації в системах штучного інтелекту тощо. Для кожного з цих сценаріїв критично важлива не лише висока точність (Precision), але й повнота (Recall) виявлення всіх релевантних збігів, навіть якщо вони глибоко перефразовані.

Традиційні підходи до розв'язання цієї задачі, такі як TF-IDF [1], шингли [2], Levenshtein Distance [3], Jaccard Similarity [4] демонструють обмежену ефективність при роботі з семантично близькими текстами, будучи чутливими до поверхневої структури та не враховуючи контекст і значення слів.

Косинусна подібність [5] як міра кута між векторами в багатовимірному просторі залишається одним із найпопулярніших способів кількісного визначення схожості документів. Проте класична косинусна міра, застосована

до звичайних частотних векторів, не враховує семантичну значущість різних компонент і контекстуальні зв'язки між словами. Саме тому актуальним є дослідження модифікацій цього методу, які б посилили його чутливість до семантичних нюансів, зберігаючи при цьому обчислювальну ефективність.

У даній роботі пропонується модифікований метод косинусної подібності, який поєднує переваги сучасних контекстних векторних представлень із статистично обґрунтованим зважуванням координат, що дозволяє краще виявляти нечіткі дублікати навіть при значних лексичних відмінностях. Метод передбачає розширений цикл попередньої обробки тексту, контекстуальне зважування компонент вектора на основі їх інформативності та оптимізований пороговий критерій для прийняття рішення. Запропонований підхід спрямований на подолання обмежень класичних методів і забезпечення високої якості виявлення семантично еквівалентних текстів у різноманітних прикладних сценаріях.

Для практичної реалізації та оцінювання ефективності модифікованого методу у роботі створено програмне забезпечення з клієнт-серверною архітектурою, інтегроване з сучасними векторними базами даних та адаптоване для багатомовного аналізу. Експериментальне порівняння розробленого модифікованого методу з альтернативними методами на стандартизованих наборах даних дозволяє об'єктивно оцінити його переваги та визначити оптимальні сфери застосування.

1. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА ТЕХНОЛОГІЙ ВИЯВЛЕННЯ НЕЧІТКИХ ДУБЛІКАТІВ ТЕКСТОВИХ ДАНИХ

1.1. Поняття «нечіткий дублікат» і фактори семантичної варіативності

У контексті обробки текстових даних нечіткий дублікат визначається як текстовий фрагмент, який за змістом є семантично еквівалентним або близьким до іншого фрагмента, але відрізняється поверхневою структурою, лексикою, синтаксисом або стилем. На відміну від точних дублікатів, які є повними копіями, нечіткі дублікати мають варіативність, що ускладнює їх автоматичне виявлення традиційними методами порівняння рядків.

Нечіткість має градієнтну природу: від одиничних помилок у словах до глибоких перефразувань, де читачеві потрібно «домислювати» еквівалентність. Зручно описати цей спектр через ступінь збереження семантики, що подано в таблиці 1.1.

Таблиця 1.1

Градієнт нечіткого дублювання

Рівень збігу	Приклад	Ключова відмінність
Поверхневий	«Petro Poroshenko» ↔ «P. Poroshenko»	Регістр, пунктуація, скорочення
Лексичний	«ціни зросли» ↔ «тарифи підвищилися»	Синонімічне заміщення слів
Синтаксичний	«Компанія придбала стартап» ↔ «Стартап було придбано компанією»	Зміна порядку слів / діатезу
Семантичний	«95% клієнтів задоволені» ↔ «Лише 5% залишилися незадоволеними»	Логічна еквівалентність з інвертованою полярністю

На практиці всі чотири рівні можуть комбінуватися, завдяки чому вектор

розбіжностей стає багатовимірним. Це пояснює, чому класичні підходи, засновані виключно на лексичних n-грамах, демонструють низьку повноту: відхилення в одному вимірі вже робить подібність непомітною для поверхневих метрик. Сучасні контекстні векторні представлення слів – зокрема SBERT [6] частково долають цю проблему, адже кодують висловлювання на рівні семантичних відношень, а не окремих токенів.

Фактори семантичної варіативності, що породжують нечіткі дублікати, включають:

1. Лексична синонімія та антонімія: заміна «придбати» на «купити» не змінює факту дії, але істотно впливає на частотні та n-грамні моделі.
2. Морфологічне багатство: у слов'янських мовах одна лексема набуває десятків відмінкових форм; без лематизації система трактує їх як різні слова.
3. Синтаксичні трансформації: вільний порядок слів, пасивні конструкції та вставні речення модифікують поверхню, але залишають інформативне ядро незмінним.
4. Орфографічні й форматні спотворення: друкарська помилка, відсутність діакритики, змішані кодування або HTML-теги створюють «шум», що потребує попередньої нормалізації.
5. Прагматичні переформулювання: зміна полярності («успішно ↔ не зазнав невдачі») або точка зору (1-ша ↔ 3-тя особа) зберігає сенс, але ускладнює автоматичне зіставлення.

Основна складність роботи з нечіткими дублікатами полягає в тому, що вони вимагають аналізу глибинної семантики, а не лише поверхневих ознак. Традиційні підходи, такі як хешування або n-грамне порівняння, часто неефективні, оскільки не враховують контекст та значення слів. Це обумовлює необхідність застосування семантичних моделей, зокрема векторних представлень, які відображають текст у багатовимірному просторі, де подібність визначається на рівні значень, а не токенів.

Таким чином, розуміння природи нечітких дублікатів та факторів їх

варіативності є фундаментом для розроблення точних методів їх виявлення, зокрема модифікацій класичних алгоритмів подібності, таких як косинусна міра.

1.2. Контекстні моделі подібності: від частотних до семантичних підходів

Поняття косинусної подібності походить із векторної моделі інформаційного пошуку Джерарда Салтона (VSM [7]) – першої формалізованої схеми, де документ і запит кодувалися як багатовимірні не-негативні вектори термів [8]. Міра кута між такими векторами інтерпретується як «семантична близькість»: що менший косинус-кут, то подібніший векторний «напрям» змісту. Упродовж п’яти десятиліть ця концепція пройшла шлях від елементарної частотної статистики до глибоких контекстних векторних представлень, зберігши при цьому свою просту і водночас потужну геометричну інтуїцію.

Від частот до n-ґрамних поверхневих моделей

Перші реалізації VSM оперували «мішком слів» – порядкова інформація ігнорувалася, а вагу терма задавала проста частота або TF-IDF. На практиці цього вистачало для коротких запитів, але навіть незначне перефразування руйнувало векторну спорідненість. Для подолання синтаксичної розбіжності у 1990-х роках з’явилися n-ґрамні та shingling-підходи: текст представлявся набором перекривних фраз фіксованої довжини; подібність вимірювали косинусом між бінарними або частотними векторами шинглів [9]. Хоча такий метод поліпшував виявлення перестановок слів, він залишався чутливим до лексичних замін і морфологічних змін.

Перехід до розподілених словникових представлень

Рушійним фактором еволюції стала ідея «нехай модель сама навчиться, які слова подібні». Word2Vec [10] вперше показав, що слова можна вкладати в упорядкований простір ґрунтовано на контекстах уживання – так звані distributed representations [11]. Косинус між такими векторами вже відображав

не лише спільні токени, а й латентні семантичні відношення (king – man + woman ≈ queen). Однак словникові векторні представлення лишалися статичними: одна лексема → один вектор, незалежно від контексту ("bank" як «берег» чи «банк»). Для задачі нечіткого дублювання це означало, що полісемія й синтаксичні трансформації ще не враховуються належним чином.

Контекстні перетворювачі та поява sentence-рівня

Наступний прорив забезпечили бібліотеки попередньо навченої обробки природної мови. BERT [12] навчався з обох сторін контексту, формуючи deep bidirectional представлення кожного токена [13]. Усереднення токенових векторів дало змогу застосувати класичний косинус до «контекстуально осмислених» речень. Та пряме порівняння BERT-виходів вимагало пропустити усі пари речень через мережу, що робило пошук у великих колекціях ресурсоємним.

Обмеження базових моделей BERT у задачах обчислення семантичної схожості між реченнями були успішно подолані завдяки розробленню Sentence-BERT (SBERT). Це – модифікація архітектури BERT, реалізована у вигляді сіамської мережі, яка забезпечує можливість перетворення кожного речення на фіксований вектор ознак. Найважливішою перевагою SBERT є те, що порівняння двох речень не потребує повторного одночасного пропуску їх через модель, як це було у звичайному BERT. Замість цього, для кожного речення обчислюється окремий вектор, і далі проводиться їх зіставлення за допомогою метрики косинусної подібності, яка є зручною та інтерпретованою у геометричному сенсі [14].

Поєднання високої чутливості трансформерної архітектури до контексту з ефективністю та простотою косинусної метрики зробило SBERT фактично стандартом для широкого класу завдань, що передбачають вимірювання семантичної близькості між текстовими фрагментами. Зокрема, він виявився надзвичайно корисним для задач виявлення нечітких дублікатів, де важливо враховувати не лише лексичну, а й смислову подібність між реченнями або

документами. Загальну еволюцію підходів до вирішення цієї задачі проілюстровано у таблиці 1.2.

Таблиця 1.2

Еволюція косинусної подібності та векторних просторів

Етап (рік)	Представлення	Властивості	Недоліки для нечітких дублікатів
VSM / TF (1970)	частоти термів	швидко, просто	чутливо до лексичних змін
Shingling (1997)	n-ґрами	стійкість до перестановок	лексична й морфологічна уразливість
Word2Vec (2013)	статичний word-вектор	вловлює семантику	полісемія, позаконтекстність
BERT (2019)	токенові контекстні вектори	багатий контекст	обчислювальна складність парного зіставлення
SBERT (2019)	sentence-вектор	швидкий cos-пошук, багатомовність	потребує fine-tune; все ще евклідова геометрія

Еволюція методів косинусної подібності відображає фундаментальний перехід від лінгвістично наївних моделей обчислення до контекстно-обізнаних систем аналізу текстів. Початкові реалізації косинусної подібності спиралися на примітивне порівняння частот окремих слів, ігноруючи семантичні зв'язки між термінами та контекстуальні особливості природної мови. З розвитком обчислювальної лінгвістики та впровадженням методів машинного навчання, відбулася суттєва трансформація підходів до вимірювання текстової подібності. У подальших розділах буде розглянуто, як модифікації класичної косинусної міри можуть підвищити точність у цій задачі.

1.3. Семантичні векторні представлення та їх використання у виявлені дублікатів

У векторній парадигмі обробки тексту кожен документ або речення подається у вигляді точки в багатовимірному просторі, а їхня близькість оцінюється геометричною мірою – найчастіше косинусом. Проте ефективність такого підходу визначається саме тим, яке векторне представлення стоїть за точками. Векторне представлення називають семантичним, коли координати відбивають значення слова чи речення, а не лише його поверхневий вигляд; тоді перефрази, переклади та різні граматичні форми «стягуються» у просторові кластери. Для завдання нечіткого дублювання це критично: система має розпізнавати, що «ціни підвищилися» і «тарифи зросли» репрезентують одне й те саме повідомлення, попри відсутність спільних n-ґрам.

Перший крок до семантики зробили статичні словникові векторні представлення. Моделі Word2Vec, засновані на ідеї, що «слова з подібними контекстами мають подібні вектори», продемонстрували здатність кодувати латентні відношення між словами. Проте одне слово відповідало одному вектору, тому полісемія та залежність значення від оточення залишалися неврахованими. Частково цю проблему пом’якшив fastText [15], додаючи внутрішню морфемну структуру, але він усе ще був позаконтекстним.

Революційним став перехід до контекстних токенізованих векторних представлень із появою трансформера BERT. У BERT кожне входження слова одержує власний вектор, сформований з огляду на лівий і правий контекст, а тому різні значення полісемічного "bank" розходяться у просторі. Втім, напряму порівнювати два речення через BERT незручно: потрібно пропускати кожну пару через модель, що робить обчислення квадратично-дорогими для великих корпусів.

Цю перешкоду усунув Sentence-BERT (SBERT) – сіамська модифікація BERT, яка одразу продукує фіксований вектор речення; достатньо раз згенерувати векторне представлення і далі порівнювати його зі всіма іншими

через простий $cos\,\theta$. На стандартних наборах Semantic Textual Similarity SBERT підняв якість більш? ніж на 10 пунктів F1 у порівнянні з попередніми sentence-методами [14, 16]. Крім англійської, доступні багатомовні варіанти – наприклад, LaBSE [17], що покриває 109 мов і дозволяє шукати дублікати між українським та англомовним текстом у спільному просторі [18]. Такі вектори природно інтегруються у сучасні векторні бази даних (Qdrant [19], FAISS [20], Milvus [21]), які реалізують швидкі подібні сусідні пошуки саме для косинусної метрики [19].

Таблиця 1.3

Порівняння основних сімейств векторних представлень у контексті

Родина векторних представлень	Розмір вектора (кількість вимірів)	Чутливість до перефраз	Контекстність	Підтримка мов
Word2Vec / GloVe	100–300	низька	відсутня	окрема модель на мову
fastText	300	середня	відсутня	150+ мов
BERT [CLS-avg]	768	висока	повна	100+ мов
Sentence-BERT / LaBSE	384–768	дуже висока	повна	50–100+ мов

Таким чином, еволюція векторних представлень від частотних векторів до глибоких контекстних представлень довела, що косинусна міра може залишатися центральною, якщо під неї підведено адекватну семантичну основу.

1.4. Огляд існуючих методів виявлення нечітких дублікатів

1.4.1. *Метод Term Frequency (TF)*

У найпростішій векторній моделі інформаційного пошуку кожен

документ d перетворюється на k-вимірний вектор $v_d^{(TF)}$, координати якого дорівнюють абсолютним частотам входжень термів із наперед сформованого лексикону $\{t_1, \dots, t_k\}$. Формально (1):

$$v_d^{(TF)} = (tf_{1d}, tf_{2d}, \dots, tf_{kd}), \qquad tf_{id} = |\{t_i \in d\}|, \tag{1}$$

де tf_{id} – абсолютна кількість входжень i -го терма з лексикону $\{t_1, \dots, t_k\}$.

Після сегментації та токенізації текстів вектори зберігаються у пам'яті або індексі; подібність двох документів обчислюють геометрично (найчастіше косинусною відстанню). Така модель дає коректні результати лише за умови, що тексти містять ідентичні лексичні елементи з подібною кратністю. Будь-яка синонімічна заміна, словозміна чи перестановка різко зменшує співвідношення компонент. Крім того, функціональні слова із високим TF ("і", "the" тощо) домінують у векторі, маскуючи інформативні лексеми.

Переваги. Принцип визначення частот реалізується за лінійний час і легко оновлюється інкрементально, тому TF застосовують як початковий фільтр при обробці масштабних потокових колекцій.

Обмеження. Відсутня нормалізація за довжиною документа; висока чутливість до поверхневих змін; нульова семантична здатність виявляти перефрази або переклади.

1.4.2. *Метод TF-IDF*

Для послаблення впливу глобально частих слів у 1972 р. Карен Спарк-Джонс запропонувала ваговий коефіцієнт Inverse Document Frequency (IDF), який зменшує внесок термів, присутніх у більшості документі [22, 23]. Підсумкова вага терма t_i в документі d визначається добутком (2):

$$w_{id} = tf_{id} \times idf_i, \qquad df_i = \log\frac{N}{df_i}, \tag{2}$$

де N – кількість документів у корпусі; df_i – кількість документів, у яких зустрічається t_i.

Логарифмічний IDF знижує значущість частих термів, водночас підсилюючи рідкісні слова, що несуть більшу дискримінаційну здатність. У

результаті косинусна подібність між TF × IDF-векторами ставить акцент на значущій лексиці, зменшуючи кількість хибно-негативних результатів порівняно з «сирим» TF. У контексті пошуку нечітких дублікатів TF × IDF залишається поверхневим: лемматичні чи синонімічні перефрази, а також зміна порядку слів порушують прямий збіг токенів. Проте цей метод і нині ефективний як попередній ("candidate generation") етап, оскільки забезпечує швидку відсіч очевидно різних документів перед застосуванням складніших семантичних алгоритмів.

Переваги. Добре масштабується; пригнічує нефункціональну лексику; легко індексується у традиційних системах на зразок BM25.

Обмеження. Робота з лексемами без урахування контексту; нечутливість до синонімів, морфологічних і синтаксичних трансформацій; необхідність перерахунку idf_i при суттєвому зростанні корпусу.

1.4.3. *Косинусна подібність*

Косинусна подібність – це ядро векторної моделі Салтона: документ і запит (або дві текстові одиниці будь-якого типу) кодуються як вектори $u, v \in R^k$, а ступінь їхньої подібності визначається косинусом кута між цими векторами (3):

$$\cos(u, v) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \cdot \|\vec{v}\|}, \qquad -1 \leq \cos(u, v) \leq 1, \tag{3}$$

де $u \cdot v$ – скалярний добуток, а $\|u\|$ – евклідова норма вектора [24].

Оскільки значення метрики залежать лише від напряму, а не від довжини, косинусна подібність автоматично нормалізує вплив різної кількості термів: коротке речення та розгорнутий абзац можуть виявитися однаково «близькими», якщо збігається пропорційний розподіл ваг. У частотних векторах (TF, TF × IDF) висока косинусна подібність сигналізує значний перетин лексем. Проте метод залишається суто поверхневим: заміна «купити» на «придбати» обнуляє відповідні координати й зменшує скалярний добуток. Саме тому традиційна практика комбінує $cos\,(\theta)$ зі збагаченими

представленнями – наприклад, шляхом:

- застосування контекстного векторного представлення SBERT: тоді координати відбивають латентну семантику, а $cos(\theta)$ уже вимірює кут між смисловими напрямами речень, а не простим співпадінням токенів;
- семантичного розширення словника (лематизація, синонімічні кластери) перед проєктуванням у TF × IDF-простір.

Переваги

- Масштабна інваріантність забезпечує коректне порівняння документів різної довжини без додаткової нормалізації;
- лінійний час щодо кількості ненульових координат і відсутність гіперпараметрів;
- геометрична інтерпретація дає чітку межу «0» (ортогональність) між несхожими текстами.

Обмеження

- Повністю залежить від якості векторного представлення: без семантичного представлення не «бачить» синонімії, полісемії та синтаксичних трансформацій;
- для довгих документів TF-вектори стають надмірно розрідженими, що знижує дискримінаційну силу верхнього скалярного добутку;
- у випадку високого шуму (HTML-теги, стоп-слова) потребує попереднього чищення й вагового зниження нерелевантних термів.

У сучасних системах виявлення дублікатів косинус залишається «функцією подібності за замовчуванням» завдяки своїй обчислювальній легкості [25]; успішність же всього алгоритму визначається тим, наскільки семантично насиченим є вектор, що підставляється у формулу. Саме тому в подальшому дослідженні косинусна міра застосовується поверх SBERT і доповнюється модифікованими ваговими коефіцієнтами, адаптованими до багатомовного тексту.

1.4.4. *Jaccard Similarity*

Jaccard-подібність походить із теорії множин і застосовується до тексту після перетворення його на набір дискретних елементів – звичайно це:

- множина токенів (унікальних слів);
- або множина шинглів k-слівних фраз, що частково враховують порядок.

Для двох текстів A і B з множинами елементів A та B показник визначається формулою (4):

$$Jac(A,B) = \frac{|A \cap B|}{|A \cup B|}, \qquad 0 \leq \text{Jac} \leq 1, \qquad (4)$$

Порядок дій при виявленні дублікатів.

- Текст нормалізують (нижній регістр, видалення пунктуації, за потреби – лематизація).
- Обирають стабільну одиницю (слово чи шингл).
- Створюють множину унікальних елементів і обчислюють частку перетину.

Високе Jac свідчить, що документи покривають майже ту саму лексичну площу; низьке – про відсутність суттєвого перекриття.

Переваги

- Міра незмінна до розміру текстів: дублікати різної довжини, але з однаковим набором шинглів дають 1.
- Раховує лише унікальні елементи, тож спам-повтори не впливають на результат.
- Проста реалізація та чітка інтерпретація.

Обмеження

- Лексична чутливість: будь-яка синонімічна заміна прибирає елемент із перетину, навіть якщо смисл не змінився.
- Відсутність ваг: частота появи слів ігнорується: «банк банк банк» і «банк» утворюють однакову множину.
- Скалярна складність: для великих множин пряме обчислення

перетину/об’єднання потребує сортування або хеш-структур; тому на вебмасштабі Jaccard майже завжди поєднують з апроксимацією MinHash [26].

- Морфологія та порядок слів: без лематизації та налаштованого k-шингла показник падає на текстах, де одна лексема з’являється в різних формах або реченнях із перестановками.

Завдяки своїй простоті Jaccard часто використовується як перший фільтр: швидко відсікає явно несхожі пари, після чого до залишку застосовують обчислювально дорожчі, але семантично глибші методи – зокрема векторні представлення, описані далі.

1.4.5. *MinHash*

MinHash (Minimum Hashing) був запропонований А. Бродером як ефективна апроксимація Jaccard-подібності для великих колекцій документів – насамперед вебсторінок, де прямих порівнянь «кожен з кожним» виконати неможливо через квадратичний масштаб. Ідея полягає у стисненні великої множини шинглів кожного тексту до короткого підпису (signature); Jaccard-близькість двох документів замінюється ймовірністю того, що їхні підписи збігаються.

Узагальнений алгоритм:

1. Шинглювання. Для кожного документа формується множина k-слівних шинглів S.
2. Набір випадкових хеш функцій. Вибирають r взаємно незалежних хешів $h_1, \dots, h_r; S \to Z$.
3. Обчислення підпису. Для документа D мінімальне значення кожного хешу на його множині шинглів заноситься у вектор підпис $m_D = (min_{s \in s_D} h_1(s), \dots, min_{s \in s_D} h_r(s))$.
4. Оцінка подібності. Імовірність того, що $m_A[i] = m_B[i]$дорівнює Jaccard подібності множин шинглів документів A та B. Практично її оцінюють як частку однакових компонент у двох підписах. Похибка

зменшується зі зростанням r.

5. Locality Sensitive Hashing (LSH). Щоб уникнути повної перевірки усіх пар, підписи розбивають на b блоків по r/b координат і хешують блоки у «відра»: документи, що потрапили бодай в одне спільне відро, вважаються кандидатами на збіг і передаються на точніше порівняння.

Таблиця 1.5

Переваги та недоліки методу MinHash

Аспект	Переваги	Обмеження
Пам’ять/швидкість	Підпис із десятків – сотень цілих чисел замість тисяч шинглів; LSH дозволяє субквадратичний пошук	Для високої точності потрібні сотні хеш-функцій; індекс LSH ускладнює налаштування
Чутливість до перестановок	Сталий результат, якщо порядок фраз змінюється частково	Лексичні та семантичні заміни, а також морфологія не враховуються
Масштабованість	Придатний до сотень мільйонів документів	Залишається «*підхід поверхні*»: без наступної семантичної фази не помічає глибоких перефраз

MinHash виступає інструментом швидкої попередньої селекції для нечітких дублікатів, обмежуючи кількість пар документів для подальшого детального семантичного порівняння.

1.4.6. *Levenshtein Distance*

Levenshtein відстань визначає мінімальну кількість односимвольних операцій – видалення, вставлення або заміни – необхідних для перетворення рядка s на t. Формально, для довжин $m = |s|$ і $n = |t|$ динамічне програмування обчислює матрицю (5):

$$JD(i,j) = \min \begin{cases} D_{i-1,j} + 1 \text{ (видалити символ } s_i), \\ D_{i,j-1} + 1 \text{ (вставити символ } t_j), \\ D_{i-1,j} + c \text{ (замінити, якщо } s_i \neq t_j), \end{cases} \qquad D_{0,0} = 0, \qquad (5)$$

де $c = 0$ при збігу символів і 1 – при відмінності. Підсумкове значення $D_{m,n}$ і є відстанню. Алгоритм має часову складність $O(mn)$; за потреби пам'ять скорочують до $O(min\{m,n\})$, зберігаючи лише два останні рядки матриці.

У завданні виявлення дублікатів класичний метод Левенштейна застосовують переважно для коротких послідовностей – назв продуктів, заголовків статей, ідентифікаторів. Для довших текстів використовують token-based Levenshtein: спершу розбивають речення на слова або речення на абзаци та рахують відстань між послідовностями токенів. Щоб отримати показник подібності, значення відстані нормують, наприклад (6):

$$sim_{Lev}(s,t) = 1 - \frac{D_{m,n}}{\max(m,n)}. \qquad (6)$$

Метод Левенштейна чудово «бачить» локальні правки – друкарські помилки, перестановки літер, односимвольні вставки. У базах користувацьких імен або Product SKU це – головний критерій збігу. Однак, вже заміна слова синонімом ("car" → "automobile") потребує виконання багатьох операцій, тож семантичні перефрази дають велику відстань попри однаковий зміст. Крім того, квадратична складність робить метод обчислювально «важким» для порівняння повних документів.

Порівняно з Jaccard чи TF × IDF Levenshtein реагує на порядок символів: два рядки анаграми можуть мати Jaccard 1, але Levenshtein > 0. З іншого боку, будь-яка перестановка довгих підрядків різко збільшує $D_{m,n}$, тому для контенту з вільним порядком речень метрика недостатньо стабільна.

У сучасних системах виявлення дублікатів відстань Левенштейна використовується як точний локальний критерій: після того як швидший алгоритм (наприклад, MinHash + LSH) відібрав кандидати, Levenshtein дає строгий числовий показник схожості коротких заголовків, адрес або ідентифікаторів, де семантичні моделі є надлишковими, а символьна

точність – критичною.

1.4.7. *Jaro-Winkler Distance*

Jaro-Winkler – модифікація Jaro distance, розроблена для виявлення дублікатів коротких лексичних рядків, насамперед власних імен або назв міст. Метрика корегує початковий показник Jaro, надаючи додаткову «бонусну» вагу довгому спільному префіксу, оскільки емпірично збіги на початку слова вказують на вищу ймовірність тотожності.

1. Базовий показник Jaro.

Для двох рядків s та t (довжини m і n) визначаються:

- кількість відповідних символів m_a: дві літери вважаються відповідними, якщо вони ідентичні та їхній індекс відрізняється не більше ніж на $\lfloor max(m,n)/2 \rfloor - 1$;
- кількість транспозицій t_r: пари відповідних символів, що з'являються у різному порядку.

Jaro подібність (7):

$$J = \frac{1}{3}\left(\frac{m_a}{m} + \frac{m_a}{m} + \frac{m_a - t_r/2}{m_a}\right), \qquad 0 \le J \le 1. \tag{7}$$

2. Корекція Winkler.

Нехай l – довжина спільного префікса (максимум 4), а p – ваговий коефіцієнт (типово 0.1). Тоді фінальна відстань між текстовими рядками (8):

$$JW = J + pl(1 - J). \tag{8}$$

Завдяки додатку $p\,l\,(1-J)$ рядки "Kovalenko" та "Kovalenka" отримують більший бал, ніж "Kovalenko" та "Novalenko", навіть якщо число збігів символів однакове.

Алгоритмічні властивості.

Складність – $O(m+n)$ при використанні двох послідовних проходів і ефективних позначок відповідності. Простір – $O(m+n)$ або $O(max\{m,n\})$ з оптимізаціями.

Переваги

- Висока чутливість до невеликих помилок у введенні, особливо до перестановок сусідніх літер.
- Додаткова вага префікса робить метрику придатною для «людських» даних (імена, адреси), де початкові літери зберігаються найчастіше.
- Лінійна складність дає змогу обробляти великі списки коротких рядків без істотних затримок.

Недоліки

- Метод призначений для рядків довжиною до 20–30 символів; для довгих текстів або речень показник втрачає дискримінаційну здатність.
- Метрика залишається символьною: синонімія, морфологічні варіанти та семантика не враховуються.
- Метод чутливий до змін у кінці слова: один зайвий символ після довгого префікса може суттєво знизити бал.

У системах пошуку дублікатів Jaro–Winkler найчастіше використовується як швидкий детектор близьких назв кандидатів, особливо коли порівнюються довідникові поля (ім’я, прізвище, назва компанії). Для довших або семантично складніших текстів його зазвичай поєднують із ширшими контекстними методами.

1.4.8. *I-Match*

I-Match – індексний алгоритм для попереднього виявлення документів, що передають однаковий зміст, але можуть відрізнятися оформленням. Його основна ідея – сформувати для кожного документа унікальний ключ-підпис, складений із невеликої кількості найбільш «інформативних» термів; підписи порівнюються не за близькістю, а на точну тотожність. Метод був запропонований у корпоративних системах Enterprise Search для швидкої дедуплікації вебсторінок та технічних документів.

Етапи побудови підпису:

1. Текст нормалізують: зводять до нижнього регістру, усувають пунктуацію, вилучають стоп-слова.

2. Для решти слів обчислюють IDF-ваги на всьому корпусі, що дає оцінку «поінформованості».

3. Обирають k термів із найвищими IDF-значеннями; чим менша поширеність слова, тим вища ймовірність, що воно унікально характеризує документ.

4. Відібрані терми сортують за алфавітом, конкатенують та пропускають крізь криптографічну хеш функцію (наприклад, MD5).

5. Отриманий хеш рядок фіксується як підпис документа і зберігається в індексі.

Переваги

- Швидкодія. Порівняння зводиться до перевірки рівності хешів, що виконується за $O(1)$.
- Масштабованість. Підпис містить постійний обсяг даних, незалежно від довжини тексту.
- Висока точність (precision). Якщо два документи мають однаковий підпис, імовірність помилкового збігу мінімальна: вони містять тотожний набір рідкісних слів.

Недоліки

- Нестача recall. Додавання або вилучення навіть одного інформативного слова змінює підпис, тому перефрази із заміною кількох ключових термів можуть лишитися непоміченими.
- Залежність від параметра k і статистики корпусу. У невеликій колекції IDF-оцінки нестійкі. У багатомовній – терми різних мов не порівнюються прямо.
- Відсутність семантики. Алгоритм не враховує синонімію та морфологічні варіанти, працює виключно з поверхневими формами слів.

На практиці I-Match використовують як канал первинної фільтрації: з мільйонів документів виділяються групи з однаковими підписами, після чого всередині кожної групи виконується детальніший аналіз, щоби підтвердити або спростувати дублювання.

Нижче у таблиці 1.6 продемонстровано метрики ефективності з топових результатів, які часто фігурують у дослідженнях Quora Duplicate Pairs, TREC Duplicate News, наборів bug-звітів Eclipse Platform – і узагальнені в оглядових статтях та інженерних звітах [27-29].

Таблиця 1.6

Метрики ефективності існуючих методів

Метод	Точність	Влучність	Повнота	F1-міра
Term Frequency	55 %	65 %	40 %	46 %
TF × IDF	60 %	68 %	45 %	51 %
Cosine Similarity	65 %	72 %	55 %	60 %
Jaccard Similarity	70 %	70 %	35 %	47 %
MinHash	78 %	76 %	60 %	68 %
Levenshtein Distance	85 %	74 %	50 %	63 %
Jaro–Winkler Distance	88 %	73 %	45 %	60 %
I-Match	95 %	79 %	30 %	45 %

1.5. Огляд існуючих комерційних та відкритих програмних рішень

За останнє десятиліття на ринку з’явилося понад два десятки вебсервісів і настільних застосунків, що пропонують автоматичну перевірку текстів на збіги. Їхні алгоритми варіюються від класичних n-шинглів і MinHash-підписів до sentence векторних представлень і спеціалізованих моделей виявлення

дублікатів. У цьому підрозділі розглянемо вісім популярних рішень – від повністю безплатних до корпоративних – та проаналізуємо їх функціональні можливості, цінові моделі й обмеження на прикладі SearchEngineReports, Copyleaks, QuillBot, PrePostSEO, Grammarly, SmallSEOTools і Dupli Checker.

1.5.1. *SearchEngineReports*

Платформа SearchEngineReports позиціонується як безплатний набір SEO-інструментів, у якому перевірка на плагіат є найбільш затребуваним сервісом. Користувач може вставити до 2000 слів або завантажити файл (DOCX, PDF, TXT) і отримати звіт, де відображено відсоток збігів, уривки речень та гіперпосилання на джерела [30]. Для великих обсягів файлів передбачено платні тарифи з поступовим підвищенням ліміту запитів і розширеною аналітикою [31].

Технічний підхід. Компанія не розкриває деталей алгоритмів, однак у технічній документації згадуються:

- комбінована перевірка «точного» збігу (n-шингли) та «глибинного» пошуку за латентними ключами;
- звернення до «баз даних із мільярдів вебсторінок» – імовірно, через власний індекс або API сторонньої пошукової системи;
- багатомовна підтримка (англійська, іспанська, французька, португальська та ін.) – отже, здійснюється первинна нормалізація кодування і стоп-слів [32].

У незалежному тесті інструмент розпізнав 62 % плагіату у контрольному документі (проти 88 % у лідера рейтингу), але правильно ідентифікував оригінальне джерело [32]. Це свідчить про пристойну точність при середній повноті, характерну для систем, що комбінують поверхневі та ключові збіги без повноцінних семантичних моделей.

Переваги

- Повністю безплатний базовий режим;
- швидка ве-перевірка й API для інтеграції;

- підтримка кількох мов і файлів різних форматів.

Недоліки

- Обмеження 2000 слів у free-версії;
- вища похибка пропуску перефразованого контенту порівняно з платними академічними сервісами;
- рекламне оточення й відсутність гарантій конфіденційності для завантажених матеріалів.

1.5.2. *Copyleaks*

Copyleaks – це SaaS-платформа корпоративного рівня, націлена на університети, видавництва, EdTech-провайдерів і великі компанії, яким потрібна захищена перевірка плагіату. Сервіс доступний через вебінтерфейс, розширення для LM та REST/GraphQL API з granular-кредитною моделлю оплати [33].

Ключові можливості (2025):

- Semantical Plagiarism & Paraphrase Checks. Власна нейромережева модель охоплює «глибокі» перефрази; у відкритому тесті Eden AI середня точність склала 99 %, FP ≈ 0.2 % [34].
- Source-Code Similarity. API-метод для виявлення копіювання у програмних файлах (C , Java , Python).
- Military-grade security: 256-bit SSL, SOC-2/SOC-3, GDPR-повна сумісність; опція on-prem.

Технічний підхід. Copyleaks комбінує кілька шарів: швидкий шингл з MinHash для candidate-generation, багатомовні SBERT з косинусною метрикою для семантичного матчингу, а фінальний класифікатор – градієнтний бустинг, натренований на 80 млн пар «дублікат / не дублікат» [35].

Переваги

- Висока точність на перефразованому тексті (незалежні огляди дають 88–91 % F1) [36].
- Широка мовна підтримка й можливість перевірки коду.

- Гнучкий API з SDK для Python, Node.js, Java тощо; інтеграція у LMS без додаткових проксі.
- Детальні звіти з підсвічуванням збіглих фрагментів і посиланнями на джерела.

Недоліки

- Цінова модель на основі кредитів: великий корпус або часті перевірки обійдуться дорожче за конкуренти безкоштовного класу.
- Час оброблення дуже довгих документів (> 50 000 слів) може перевищувати 2–3 хв, що важливо для інтерактивних сценаріїв [35].
- Експерти Originality.ai відзначають поодинокі хибнопозитивні спрацьовування на спеціалізованих технічних текстах [36].

Copyleaks демонструє одну з найвищих комерційно доступних точностей семантичної перевірки плагіату та AI-детекції, проте вимагає платної підписки й оптимізації кредитів при інтенсивному потоковому використанні.

1.5.3. *QuillBot*

QuillBot стартував як онлайн-перефразатор, а сьогодні пропонує «Writing Suite» із функціями paraphrase, grammar-check, summarizer та плагіат-детектор (доступний лише в Premium-плані від ≈ 9 USD/міс) [37]. Його цільова аудиторія – студенти, копірайтери й технічні блогери, яким потрібен швидкий all-in-one-редактор без складної інтеграції.

Ключові можливості:

- Paraphraser 25 + мов. Підтримує більше ніж 25 мов і чотири варіанти англійської (US/UK/CAN/AUS) [38, 39].
- Plagiarism Checker. Порівнює введений текст із відкритим вебіндексом та академічними БД; видає відсоток збігів і посилання на джерела. Ліміт – 20 сторінок ≈ 5 000 слів на 1 кредит, 50 кредитів/міс у стандартній підписці.
- AI-detector. Визначає GPT-/Claude-згенерований контент з точністю

~90%, але гірше розпізнає «переписаний» AI-текст [40, 41].

- Браузерні розширення у Google Docs, Gmail, Slack тощо; інтеграція в MS Word через надбудову.

Компанія не розкриває повної архітектури, однак тестові публікації свідчать про поєднання:

1. класичних n-шинглів + TF-IDF для «точних» збігів;
2. ML-класифікатора над sentence векторним представленням (multilingual SBERT) для «глибоких» парафраз [41].

API поки що відсутній; перевірка здійснюється лише через вебінтерфейс.

Переваги

- Комплексний «однокнопковий» сервіс: перефразуй → перевір орфографію → скануй на плагіат.
- Підтримка десятків мов та просте розширення для браузера.
- Висока точність виявлення дослівних копій (до 99 %) [42].

Недоліки

- Менш стабільний на AI-перефразованому контенті: Recall падає до ~83% [41].
- Пакет-ліміти й відсутність API. Інтенсивне потокове сканування або автоматична інтеграція неможливі без взаємодії з користувачем.
- Мовна підтримка не повна. Немає офіційної підтримки української; для кириличних текстів точність нижча, бо stop-word листів і морфологічної нормалізації бракує.
- Немає аналізу програмного коду й міжмовного матчингу перекладів (функції, заявлені в Copyleaks та інших корпоративних рішеннях).

QuillBot зручний як універсальний редактор-«комбайн» для студентських есе та блог-постів, але у глибокому семантичному дублюванні, багатомовних корпусах або потокових сценаріях поступається спеціалізованим системам, що використовують адаптовані sentence векторним представленням.

1.5.4. *PrePostSEO*

PrePostSEO – це freemium-платформа з понад 70 онлайн-інструментами для контент-мейкерів і SEO-фахівців; модуль перевірки плагіату позиціонується як головна функція сервісу [43, 44].

Алгоритмічний підхід і функції.

- Пошук «дослівних» збігів виконується через класичний n-шингл + TF-IDF поріг: у незалежному огляді Originality.ai сервіс виявив 100 % прямого копіювання у двох контрольних статтях [45].
- Для «патчворк-плагіату» (різнорівневе перефразування) точність коливається: у тесті з 65 % змінених фрагментів PrePostSEO знайшов лише 3 % схожості, а при 77 % змін – 59 % [45]. Це свідчить про обмежений семантичний аналіз.
- API підтримує перевірку за URL і завантаженням файлів (DOC, PDF, TXT), повертаючи детальний JSON-звіт із відсотком збігів і джерелами [46].
- Налаштування «Sentence-Wise Checking» дозволяє ігнорувати цитати та посилання, але розрізняє лише поверхневі співпадіння.

Переваги

- Безплатна квота та простий вебінтерфейс без реєстрації.
- Гнучка тарифна сітка і швидка REST-API з лімітом до 250 000 запитів/місяць у корпоративному пакеті [46].
- Висока точність на повністю скопійованому контенті; правильна ідентифікація першоджерела у більшості випадків [47].

Недоліки

- Середній і нижчий показник Recall на перефразованому «патчворк» тексті; пропускає частку академічних джерел [45].
- Ліміт безплатної перевірки – 1 000 слів; для великих документів потрібна підписка.
- Семантичний аналіз обмежений: відсутня підтримка багатомовного крос-матчингу та перевірки програмного коду.

PrePostSEO підходить як швидкий безплатний сканер для блогів і SEO-контенту, демонструючи високу точність на дослівних копіях, проте поступається спеціалізованим академічним сервісам у виявленні глибоких перефраз і багато-мовних збігів.

1.5.5. *Grammarly*

Grammarly відома передусім як онлайн-редактор стилю та граматики, але в преміум-плані містить модуль Plagiarism Checker, інтегрований у вебкабінет, десктоп-додаток, розширення для браузера й надбудову MS Word. Алгоритм сканує текст проти «понад 16 мільярдів вебсторінок» та академічної бази ProQuest і повертає відсоток збігів, підсвічені фрагменти дублікатів й посилання на джерела [48].

Технічні особливості (доступна інформація).

- Поєднання n-шинглів із TF-IDF-зважуванням для виявлення дослівних копій.
- Семантичний пошук через sentence-вектори та внутрішній вебіндекс – результати різняться залежно від домену тестів [49].
- Максимальна довжина одного документа – ~100 000 символів; місячний ліміт для Premium – 150 000 слів або 300 документів [47].
- Підтримка лише англійської; API для автоматизації поки що не надається [50].

Переваги

- Єдиний інтерфейс для перевірки плагіату, граматики й стилю, швидкий аналіз (15-30 с на 1500 слів) [50].
- Необмежена кількість перевірок у рамках місячного ліміту, зручне візуальне виділення збігів.
- Висока точність на дослівних збігах і коротких вставках.

Недоліки

- Середній показник Recall на глибоко перефразованому чи AI-згенерованому контенті, можуть залишатися непомічені цілі

абзаци [51].

- Лише одна мова, відсутня міжмовна перевірка та сканування програмного коду.
- Закрита екосистема без офіційного REST/API – інтеграція можлива тільки через десктопні/веб-UI.

У підсумку Grammarly зручний як багатофункціональний «письмовий асистент» із гідною точністю для дослівних копій, однак його обмеження (англомовність, відсутність API та зниження Recall на глибоких перефразах) роблять сервіс менш придатним для масштабних, багатомовних або автоматизованих сценаріїв академічної та корпоративної перевірки.

1.5.6. *SmallSEOTools*

SmallSEOTools – це набір дрібних вебінструментів, серед яких перевірка плагіату працює за простим принципом «швидкий пошук збігів у вебфрагментах».

Модуль плагіату у SmallSEOTools побудовано на класичному n-шинглюванні: текст розбивається на 3–5-словні фрази, для кожної створюється хеш, а далі сервіс робить запити до відкритих пошукових API (Google / Bing). Якщо у сніпетах пошукової видачі знаходиться дослівний збіг принаймні 60 % слів шингла, речення позначається як скопійоване. Ніякої лематизації чи синонімізації не застосовується; аналіз зводиться до порівняння точних або майже точних фраз.

Переваги

- Безкоштовний старт – 2000 слів без реєстрації, що зручно для коротких блог-постів чи описів товарів.
- Проста інтеграція – браузерний інтерфейс не потребує встановлення ПЗ; у платному плані доступний легкий REST-endpoint (один текст → одна відповідь).
- Швидка реакція на прямі копії – дослівне “copy-paste” розпізнається з точністю ~95 %, що достатньо для грубого первинного фільтру.

Недоліки

- Поверхневий алгоритм – глибокі перефрази й навіть помірні лексичні заміни знижують Recall до 30–40 %.
- Жорсткі ліміти – понад 2 000 слів пакетом не обробляється без підписки; пакетна перевірка файлів і PDF-звіти доступні лише у дорожчих тарифах.
- Нестабільна швидкість і реклама – у безплатному режимі відповіді можуть затримуватися, а результативна сторінка містить рекламні блоки.
- Відсутність семантики та міжмовного аналізу – сервіс працює лише з поверхневим англоцентричним токенізатором; кириличні та інші алфавіти обробляються як звичайний Unicode-текст без морфологічної нормалізації.
- Обмежене API – не підтримує пакетне надсилання кількох документів, немає детального звіту з координатами фрагментів; підходить лише для точкових запитів невеликого обсягу.

У сумі SmallSEOTools корисний як швидкий безкоштовний інструмент для виявлення дослівних копій невеликих текстів, але він не забезпечує достатньої глибини, коли потрібна перевірка академічних робіт, багатосторінкових звітів або перефразованого контенту.

1.5.7. *Dupli Checker*

Dupli Checker – один із найдавніших безплатних вебсервісів для швидкого виявлення плагіату, популярний серед блогерів і студентів завдяки простому інтерфейсу «встав текст → отримай звіт». Платформа працює на моделі freemium: базовий ліміт 1 000 слів без реєстрації, преміум-плани збільшують обсяг та відкривають REST-API. Завдяки низькому порогу входу Dupli Checker часто використовується як первинний фільтр для виявлення дослівного копіювання, хоча його алгоритми залишаються переважно поверхневими й обмежені для глибокого семантичного аналізу.

Dupli Checker застосовує комбінований 3–4-словний шингл-аналіз у поєднанні з внутрішнім кешем вебсторінок. Під час сканування кожний шингл хешується та звіряється з невеликою власною базою й відкритими пошуковими API. Якщо послідовно збігаються щонайменше 60 % шинглів речення, система маркує цей фрагмент як плагіат. Так званий “Deep Scan” у платних пакетах лише збільшує діапазон запитів і тайм-аутів, але не вводить семантичні представлення чи міжмовний аналіз – підхід залишається поверхневим.

Переваги

- Доступність без оплати – до 1 000 слів за запит без реєстрації, чого достатньо для невеликих статей або постів.
- Простий REST-API (у преміум-режимі) – приймає текст чи файл і повертає JSON із відсотком збігів та URL-джерелами; ліміти можна масштабувати кредитами.
- Надійний для дослівних копій – точність на прямому “copy-paste” наближається до 96–98 %, що корисно як первинний фільтр.

Недоліки

- Слабка семантична чутливість – при перефразуванні на 40–50% Recall падає нижче 45%; синонімія й порядок речень не враховуються.
- Жорсткі обмеження обсягу – безплатний ліміт 1 000 слів, а максимальний пакет обмежено 25 000 слів; великі рукописи доведеться ділити вручну.
- Працює на пошукових API – залежність від сторонніх сервісів зумовлює непередбачувані затримки; у пікові години час відповіді може перевищувати 30 с.
- Відсутність спеціальних функцій – немає перевірки програмного коду, аналізу PDF-закладених текстів чи детекції AI-контенту; міжмовне дублювання не підтримується.
- Реклама та збереження даних – у безплатному варіанті результативні сторінки містять банери; гарантії конфіденційності завантажених

матеріалів мінімальні.

Таким чином, Dupli Checker придатний як швидкий безплатний інструмент для виявлення прямого копіювання невеликих текстів, однак через поверхневий алгоритм, суворі ліміти й відсутність глибокої семантики його ефективність обмежена у професійних чи академічних сценаріях.

1.6. Висновки до першого розділу

У першому розділі проаналізовано теоретичні засади та існуючі практики виявлення нечітких дублікатів тексту. Розгляд історичної еволюції показав, що поверхнево-частотні підходи (Term Frequency, TF × IDF, шингли) достатньо ефективні для детектування дослівного копіювання, однак демонструють низьку повноту на семантично перефразованому контенті й втрачають стабільність у багатомовному середовищі. Редакційні та символьні відстані (Levenshtein, Jaro–Winkler) виявляються корисними лише для коротких рядків, а їх складність зростає квадратично разом з довжиною тексту.

Спроби масштабувати Jaccard-подібність через MinHash/LSH підвищують швидкодію, але не вирішують проблеми лексичної варіативності: Recall залишається обмеженим, коли інформаційний зміст передано іншою лексемою або синтаксисом. Гібридні алгоритми, що комбінують швидкий пошук кандидатів з подальшим семантичним ранжуванням, показують кращі результати, проте для повноцінної роботи потребують глибоких контекстних векторів.

Сучасний стан комерційних та відкритих сервісів підтверджує цей висновок: безкоштовні інструменти (SmallSEOTools, Dupli Checker, PrePostSEO) покладаються переважно на шинглові й пошукові API й суттєво знижують точність при глибокому перефразуванні. Корпоративні системи (Copyleaks, Grammarly) впроваджують елементи семантичного аналізу, але їх функціональність обмежена однією мовою, закритою архітектурою або високою вартістю. Жоден із розглянутих продуктів не поєднує одночасно високу багатомовну точність, прозорий механізм налаштування та відкриту

інтеграцію.

Таким чином, існує очевидний розрив між традиційними поверхневими методами й потребою у швидкому, масштабованому та семантично чутливому механізмі виявлення нечітких дублікатів. Це обґрунтовує доцільність розроблення модифікованого методу косинусної подібності на основі використання sentence-векторів, здатного забезпечити високу повноту без втрати продуктивності, а також підтримати багатомовні корпуси та відкриті API-інтеграції.

2. РОЗРОБЛЕНИЙ МЕТОД КОСИНУСНОЇ ПОДІБНОСТІ ДЛЯ ВИЯВЛЕННЯ НЕЧІТКИХ ДУБЛІКАТІВ

У попередньому розділі було показано, що існуючі поверхневі алгоритми демонструють недостатню повноту на семантично перефразованих текстах і практично не працюють у багатомовних корпусах. У цьому розділі викладається пропозиція модифікованого методу, що зберігає обчислювальну простоту класичної косинусної метрики, але доповнюється контекстним векторним представленням речень і спеціальною ваговою корекцією. Насамперед, розглянемо, як традиційна косинусна подібність функціонує у «мішку слів» і чому цього недостатньо для задачі нечіткого дублювання.

2.1. Класична косинусна подібність та її обмеження

Виведення формули $cos\theta$ для TF / TF-IDF-векторів.

Нехай є корпус із лексиконом $\{t_1, \dots, t_k\}$. Кожний документ d подаємо вектором (9):

$$v_d = (w_{1d}, w_{2d}, \dots, w_{kd}), \tag{9}$$

де $w_{id} = tf_{id}$ – сирі частоти (метод TF) або $w_{id} = tf_{id} \cdot idf_i$ (метод TF-IDF), $idf_i = \log\frac{N}{df}$, N – кількість документів, df_i – кількість документів, що містять терм t_i. Косинусна подібність двох документів d_a, d_b визначається як кут між їхніми векторами (10):

$$sim_{cos}(d_a, d_b) = \cos\theta = \frac{v_{d_a} \cdot v_{d_b}}{\|v_{d_a}\|\|v_{d_b}\|} = \frac{\sum_{i=1}^{k} w_{ia} w_{ib}}{\sqrt{\sum_{i=1}^{k} w_{ia}^2}\sqrt{\sum_{i=1}^{k} w_{ib}^2}}. \tag{10}$$

Отже, значення метрики лежить у діапазоні $[0,1]$ для невід'ємних ваг і дорівнює 1 лише при паралельних векторах (ідентичні розподіли термів).

Переваги класичної косинус-міри:

- Масштабна інваріантність: нормалізація на довжини векторів усуває залежність від обсягу: коротка анотація й розгорнутий огляд можуть мати однаковий $cos\theta$, якщо пропорційно збігаються їхні терми.

- Лінійна складність: для розріджених TF/TF-IDF матриць скалярний добуток рахується лише по спільних ненульових координатах; обчислення масштабується лінійно до числа різних термів у парі документів.
- Простота реалізації: не потребує навчання моделей, достатньо лічильників частот; легко індексується в традиційних системах пошуку.

Ключові недоліки в контексті нечітких дублікатів:

- Лексична чутливість: заміна «ціни підвищилися» → «тарифи зросли» зануляє відповідні координати; $cos\,\theta$ різко падає, хоча зміст лишається тим самим.
- Ігнорування порядку слів та синтаксису: перестановка («стартап купила компанія» ↔ «компанія купила стартап») не впливає на TF-розподіл, але змінює семантичні акценти; косинус не розрізняє таких відмін.
- Проблема полісемії: слова з кількома значеннями (“bank”) мають одну координату; різні сенси плутаються, що спотворює подібність.
- Мовна обмеженість: вектори формуються у власному лексиконі; тексти різними мовами опиняються в ортогональних підпросторах, тож $cos\theta \approx 0$ навіть для точних перекладів.

Щоб чітко позиціювати класичну косинусну метрику серед інших поширених способів порівняння текстів, доцільно зіставити її з альтернативами за ключовими практичними критеріями. У таблиці нижче наведено порівняння чотирьох базових методів – Cosine (TF/TF-IDF), Jaccard (шингли), MinHash + LSH та символьної Levenshtein-відстані. Оцінюються чотири аспекти, важливі для задачі виявлення нечітких дублікатів:

- чутливість до перефраз (здатність розпізнати синонімічні чи граматичні зміни);
- стійкість до перестановок слів або фрагментів;
- мовостійкість (працездатність у багатомовних корпусах або між

перекладами);

- ресурсні вимоги (пам’ять і час на великих наборах документів).

Таблиця 2.1

Порівняльна таблиця характеристик існуючих методів

Метод	Чутливість до перефраз	Стійкість до перестановок	Мовостійкість	Ресурсні вимоги
Cosine (TF/TF-IDF)	–	–	–	+
Jaccard (шингли)	–	+	–	+
MinHash + LSH	–	+	–	+
Levenshtein Distance	–	–	–	–

Класична косинусна подібність зберігає перевагу в швидкодії та простоті, проте, як і інші поверхневі метрики, слабко реагує на синонімічний або граматичний перефраз і фактично не працює між різними мовами. Без семантичного підсилення жоден із розглянутих методів не перевищує поріг $\text{Precision} \geq 0.85 / \text{Recall} \geq 0.80$ при вимозі латентності < 100 мс на документ, що обумовлює необхідність вдосконаленого підходу, представленого у наступних підрозділах.

2.2. Цикл попередньої обробки тексту

Будь-який реальний корпус наповнений «сміттєвими» розбіжностями – регістром, пунктуацією, різними словоформами. Якщо подати такий необроблений текст безпосередньо в модель, вона сприйматиме дві перефразовані версії як різні об’єкти, зменшуючи Recall, а отже й ефективність усього алгоритму. Тому перед семантичною векторизацією застосовується стандартний ETL-ланцюжок: спершу виділяємо смислові атоми (речення), далі приводимо їх до одноманітного вигляду, прибираємо зайве та згортаємо

у представлення, інваріантне до поверхневих відмінностей. Нижче поетапно запропоновано кожен крок; усі вони логічні, незалежні від конкретних бібліотек і можуть бути реалізовані будь-якими інструментами з аналогічною функціональністю.

2.2.1. *Токенізація на речення*

Токенізація на речення – це алгоритмічний поділ безперервного текстового потоку на мінімальні завершені вислови. На відміну від сегментації «за словами», реченнєва токенізація зберігає локальний контекст: у межах одного речення теза зазвичай сформульована повністю, тоді як суміжні речення можуть переходити до нової думки. Пошук дублікатів найчастіше здійснюється на рівні окремих тверджень: збіг двох речень є вагомішим аргументом, ніж такий самий обсяг слів, розкиданий по абзацу. Косинусна метрика працює швидше й точніше, коли порівнює вектори однакової, відносно малої довжини. Усічення документа до речень унеможливлює ситуацію, коли довге речення «перетягує» вагу всього тексту. Сучасні sentence-моделі тренуються саме на реченнєвих парах; подача довших відрізків призводить до втрати якості, бо модель залишає поза увагою віддалені токени. Алгоритм шукає кінцеві розділові знаки («.», «?», «!») і перевіряє контекст навколо них. Якщо крапка входить до абревіатури («т. д.», "Dr.") або десяткового числа, межа не ставиться. Для кирилиці й латиниці створюються окремі списки скорочень; додатково враховуються лапки й дужки, щоб правильно закривати цитати. Після виділення меж пробіли нормалізують: кілька послідовних розривів рядка згортаються в один, а порожні результативні речення відкидаються.

2.2.2. *Перехід у нижній регістр*

Усі алфавітні символи перетворюються на малі (lower-case), залишаючи без змін цифри, знаки пунктуації та спеціальні символи.

У регістрі відсутня смислова диференціація для більшості мов. Пара «Київ»/«київ» або «Market»/«market» повинна інтерпретуватися як одне слово;

інакше під час підрахунку частот утворюються дві різні координати, що розмиває вагу.

Чим менше унікальних токенів, тим компактніші TF- і TF-IDF-вектори, а отже, менше часу та пам’яті потрібно для обчислень.

Спершу рядки приводяться до нормальної форми Unicode NFC, щоб «комбіновані» літери (наприклад, латинська «е» + гострий акут) стали єдиним кодовим пунктом. Далі виконується трансформація lowercase, яка коректно обробляє національні символи: «Ї» → «ї», "L" → "l". Для текстів, що містять власні назви або акроніми, втрата великої літери не критична: у задачі виявлення дублікатів пріоритет має збіг змісту, а не збереження орфографічних ознак.

2.2.3. *Видалення пунктуації*

Після вирівнювання регістру текст усе ще містить велику кількість символів, які не несуть лексичної чи предметної інформації, але створюють зайві токени й збільшують розмір словника. Йдеться передусім про розділові знаки, дужки, лапки, дефіси-тире, символи валют та інші елементи категорії Punctuation згідно зі специфікацією Unicode. Їхня присутність спотворює статистику частот і може спричинити хибне розділення слів у подальших етапах нормалізації. З огляду на це третім кроком циклу підготовки є усунення пунктуації.

Процедура реалізується шляхом одноразового проходу по рядку: для кожного символу перевіряється його належність до попередньо складеного списку небажаних знаків. Якщо символ належить до зазначеної категорії, він замінюється пробілом; таким способом запобігають «злипанню» сусідніх лексем, що могло б виникнути у разі простого видалення. Після завершення проходу надлишкові пробіли згортаються в один, забезпечуючи коректне токенізування на слово-рівні. Водночас значущі неалфавітні символи (наприклад, «#» у позначенні номерів або «%» у відсоткових показниках) можуть бути занесені до «білого списку» і збережені, якщо того вимагає

предметна галузь.

У результаті видалення пунктуації істотно зменшує кількість різних токенів, скорочує пам’ятні витрати під час побудови векторів і підвищує точність подальших операцій, зокрема фільтрації стоп-слів та лексичної нормалізації. Таким чином, видалення пунктуації є необхідним елементом ланцюжка підготовки даних, що усуває поліграфічний «шум» і залишає у тексті лише інформаційно релевантні одиниці.

2.2.4. *Фільтрація стоп-слів*

Після очищення тексту від пунктуаційних символів у ньому ще залишається значна частка слів, які майже не впливають на семантичний зміст, але істотно спотворюють статистику частот. Це так звані стоп-слова – високочастотні функціональні елементи мови: артиклі, прийменники, сполучники, допоміжні дієслова, частки. Їхня частка у звичайному українському чи англомовному корпусі перевищує 40 %, тоді як інформативне навантаження прямує до нуля – «і», «що», "to", "the" самі собою не допомагають відрізнити один текст від іншого. Наявність великої кількості таких токенів збільшує розмір словника і розмиває ваги справді змістових лексем, що у підсумку знижує точність будь-якої подальшої метрики, зокрема косинусної.

Фільтрація здійснюється шляхом порівняння кожного токена зі списком стоп-слів, підготовленим окремо для кожної підтримуваної мови. Списки формуються на основі відкритих корпусів (наприклад, NLTK для англійської) і доповнюються доменними модифікаціями: для української мови вилучаються деякі службові частки («будь-», «ані») і залишаються опорні слова, важливі у фахових текстах («між», «над»). Під час проходу масиву токенів усі збіги зі словником фільтруються; отримана послідовність репрезентує «змістову вісь» речення.

Фактичний ефект фільтрації стоп-слів двоякий. З одного боку, знижується розмірність векторів та середній час обробки; з іншого –

підсилюється контраст між документами, бо вагу одержують лексеми, релевантні тематиці. Для sentence-рівневих векторів це означає, що модель фокусується на змістових ядрах висловів, а не «зашумлених» периферійних словах. Таким чином, видалення стоп-слів є критичною ланкою підготовчої стадії, яка підвищує дискримінаційну здатність подальших семантичних представлень без суттєвих втрат інформації.

2.2.5. *Стемінг*

Черговою фазою нормалізації є зведення різних словоформ до спільної кореневої основи – процедура, відома як стемінг. На відміну від лематизації, яка прагне відтворити словникову форму, стемінг застосовує евристичні правила відсікання типових закінчень і префіксів, залишаючи мінімальний «стем» – частину слова, достатню для збереження семантичної спорідненості. Наприклад, англійські "connect", "connected", "connection" перетворюються на спільний стем connect; німецькі "Mensch", "Menschen" – на mensch.

Потреба у стемінгу зумовлена багатством словозміни. Якщо кожна граматична форма розглядається як окремий токен, лексикон зростає експоненційно, а статистична сила кожного варіанта зменшується. Стемінг різко скорочує кількість унікальних токенів, консолідуючи їхню вагу та підвищуючи ймовірність збігу між документами, де одна й та сама ідея виражена різними морфологічними формами.

У пропонованому методі використано алгоритм сімейства Snowball (Porter 2), оскільки він охоплює більшість європейських мов і має лаконічне правилоутворення, що забезпечує швидке виконання навіть на великих масивах тексту. Алгоритм проходить слово справа наліво, послідовно застосовуючи кілька каскадів правил «суфікс → відсікання/заміна», кожне з яких активується лише за умови, що після відсікання залишок містить принаймні одну голосну – таким чином запобігають надмірному скороченню. Хоча отриманий стем не завжди є повноцінним коренем з погляду лінгвістики, він забезпечує об'єднання споріднених слів в один токен, водночас зберігаючи

розділення семантично різних слів.

Слід зауважити, що для української та інших синтетичних мов стемінг інколи агресивно обрізає основу, втрачаючи значущі суфікси (робити → роб, кращий → кращ). Тому подальший етап лематизації використовується як «тонке налаштування» для тих мов, де це критично. Утім, навіть базовий Snowball-стемінг забезпечує до 25 % скорочення лексикону і підвищує чутливість модифікованої косинусної метрики до морфологічних варіацій без істотних втрат продуктивності.

2.2.6. *Лематизація*

Останньою стадією лексичної нормалізації є лематизація – процес приведення словоформи до її словникової (граматично нейтральної) форми, або леми. На відміну від стемінгу, що механічно зрізає афікси, лематизація базується на повному морфологічному розборі слова: визначається його частина мови, відмінок, час, число, рід тощо, після чого відшукується канонічна форма, зафіксована у лексикографічному ресурсі.

Для української мови та інших системно флективних мов точна лематизація відіграє ключову роль. Одна лексема може мати десятки відмінкових чи часових варіантів, і втрата навіть одного суфікса може змінити семантику («вести» – «весна»). Лематизація збирає всі ці форми під єдиним індексом вести, зберігаючи, однак, відмінність між омонімами (мати як «мати дитину» і мати як «мати-жіноч. ризик»). У підсумку скорочується лексикон без втрати змістових нюансів, а вектори речень репрезентують саме лексичні одиниці, а не їхні граматичні прояви.

Алгоритмічно процедура поєднує словниковий пошук із набором граматичних правил. Перший етап – пошук токена у словнику лем; якщо співпадіння відсутнє, застосовуються продукційні правила морфологічної системи (наприклад, відкидання типових відмінкових суфіксів та перевірка чергувань основи). У випадку кількох можливих лем вибір робиться за контекстом речення через прості евристики («найвірогідніша частина мови»)

або ймовірнісні моделі. Хибні позитиви мінімізуються завдяки контролю мінімальної довжини основи та перевірці голосних/приголосних у корені.

Результатом є текст, де кожен лексемний різновид поданий однією лемою, що різко підвищує точність семантичної моделі на синтаксично багатих мовах і робить представлення інваріантним до флексії. У комбінованому пайплайні, запропонованому в роботі, лематизація застосовується вибірково – для української та інших мов із доведено високою якістю морфологічних словників, тоді як у мовах із надійним стемінгом (англійська, німецька) перевага надається швидшому стемінговому скороченню.

Стемінг і лематизація – це два способи привести множину словоформ до спільної бази, однак вони різняться точністю та глибиною аналізу. На таблиці 2.2 продемонстровано порівняння між стемінгом та лематизацією.

Таблиця 2.2

Порівняння між стемінгом та лематизацією

Ознака	Стемінг	Лематизація
Підхід	Евристичне відсікання суфіксів / префіксів за фіксованими правилами.	Повноцінний морфологічний аналіз слова з урахуванням частини мови та граматичних ознак.
Результат	«Стем» – укорочена основа, не обов'язково словникова: *running* → *run*, *stories* → *stori*.	Лема – нормативна форма, яка існує в словнику: *running* → *run*, *stories* → *story*.
Точність	Нижча: можливе «перестемлювання» (*universities* → *univers*).	Вища: зберігає правильні корені та різницю між омонімами.
Швидкодія	Дуже висока; правила застосовуються за мілісекунди й не потребують словника.	Повільніша; необхідна база словоформ або модель, що виконує граматичний розбір.

Продовження табл. 2.2

Придатність	Добре для швидкого зменшення словника у великих корпусах, коли дрібні помилки некритичні.	Потрібна, якщо семантична точність важливіша за продуктивність, особливо для мов із багатою флексією (українська, німецька).

Отже, стемінг – це «грубе» механічне скорочення, яке суттєво прискорює обробку та зменшує лексикон, тоді як лематизація забезпечує лінгвістично коректне відновлення базової форми ціною додаткових обчислень. У запропонованому методі використовується комбінація: стемінг для мов із надійним Snowball-покриттям і лематизація для української, де точність морфемного аналізу має ключове значення.

2.2.7. *Векторизація речень*

Після завершення всіх стадій нормалізації кожне речення містить лише змістові лексеми у стандартизованій формі; далі його потрібно перенести в числовий простір, де геометричні операції адекватно відбиватимуть семантичну близькість. Такий перехід забезпечує векторизація – процес, у якому речення перетворюється на точку у багатовимірному евклідовому просторі.

У запропонованому методі застосовується sentence-рівне представлення, коли всі слова та їхні контексти згортено у фіксований вектор довжини d (у типових конфігураціях d = 384 – 768). На відміну від «мішка слів», що оперує частотами окремих термів, sentence-вектор формується нейронною моделлю, навченою так, аби речення, рівнозначні за смислом, проєктувалися у близькі координати, а семантично віддалені – розходилися. У такий спосіб одномірні зміщення – синонімічні підміни, перестановки слів, граматичні варіанти – автоматично нівелюються, тоді як сутнісна ідея вислову зберігається у напрямку вектора.

Процедура складається з двох логічних кроків. Спершу нормалізоване речення подається на вхід мовної моделі, де кожне слово кодується у внутрішньому трансформерному шарі, накопичуючи контекст із сусідніх токенів. На другому кроці ці токенові представлення агрегуються – зокрема, через операцію «[CLS]-пулінгу» чи усереднення – утворюючи єдиний sentence-вектор. Після цього вектор L2-нормується; нормалізація робить подальший косинусний показник незалежним від довжини речення й дозволяє інтерпретувати скалярний добуток як чистий кут між смисловими «напрямами».

Саме на цьому рівні забезпечується головна перевага запропонованого підходу: порівняння здійснюється вже не за лексичними збігами, а за глибинною семантичною структурою, що суттєво підвищує чутливість до нечітких дублікатів без шкоди для обчислювальної ефективності.

2.3. Контекстуальний косинус: модифіковане зважування компонент вектора

Попри ретельну лінгвістичну нормалізацію й використання sentence-рівневих векторних представлень, базова косинусна метрика не завжди коректно відображає змістову близькість перефразованих речень: частина координат таких векторів є «пласкою» (майже не змінюється в корпусі) й у скалярному добутку має ту саму вагу, що координати справді інформативні. Щоб підсилити чутливість до семантики, пропонується модифікована косинусна подібність із контекстними вагами, обчисленими на статистиці самого корпусу.

2.3.1. *Обґрунтування необхідності модифікації*

Косинусна міра $\cos\theta$ – одна з небагатьох метрик, що зберігає обчислювальну простоту навіть у високій розмірності: для двох L2-нормованих векторів вона зводиться до скалярного добутку, а отже обчислюється лінійно від кількості координат. Саме тому її традиційно застосовують у задачах дублювання тексту. Однак рівність ваг усіх координат

робить показник сліпим до внутрішньої структури sentence-векторів.

У трансформерних моделях окремі виміри вектора відбивають різну кількість лексико-семантичної інформації. Частина координат фіксує домінантні шаблони мови (частини мови, синтаксичні маркери) і змінюється мало від одного речення до іншого; інші координати реагують на тематично значущі слова та зв'язки між ними. Якщо внесок «пласких» координат у скалярному добутку дорівнює внеску координат змістових, кут між векторами може виявитися більшим, ніж відповідає реальній семантичній близькості; у граничному випадку це призводить до хибних відмов, коли речення, сформульовані різними словами, але з тотожним змістом, розпізнаються як не-дублікат.

Ще одна проблема стосується довжини речень. Нормування до одиничної довжини вирівнює вагу коротких і розгорнутих фраз, але не прибирає різницю в частоті «пласких» координат: чим речення довше, тим більше нейрон схильний заповнювати універсальні виміри контексту, що знову ж таки занижує $\cos\theta$ порівняно з коротким перефразом тієї самої ідеї.

Обидві ситуації зводяться до однієї закономірності: координати не є рівноцінними, і надлишкова присутність слабо-варіативних вимірів знижує чутливість метрики до справді інформативних ознак. Найприроднішою мірою «корисності» координати є її дисперсія в репрезентативному корпусі: чим ширше змінюється значення компоненти від речення до речення, тим більше диференційної інформації вона несе. Відповідно, коректним кроком виглядає зважування sentence-векторів оберненими дисперсіями координат перед обчисленням кута; тоді внесок кожної виміри стає пропорційним її інформативності, а поверхневі шаблонні компоненти приглушуються.

Запропонований далі модифікований метод реалізує саме таку ідею: перед скалярним добутком кожна координата масштабується на вагу $a_i = \frac{1}{(\sigma_i+\varepsilon)}$, де σ_i – середньокорпусна дисперсія, а ε – невелика константа для числової стійкості. Подальша L2-нормалізація повертає вектор у звичний простір, зберігаючи всі формальні властивості косинусної міри, але роблячи її

чутливішою до координат, які справді визначають семантичний зміст речення. Наступні підпункти висвітлять формальне введення ваг у класичну формулу та місце отриманої метрики у загальному процесі виявлення нечітких дублікатів.

2.3.2. *Узгоджене нормування sentence-векторів*

В даній роботі перш ніж застосовувати вагове маскування, пропонується усі sentence-вектори приводяться до єдиної довжини – L2-норми, що дорівнює 1. Це перетворення перетворює простір векторних представлених текстів з евклідового на одиничну гіперсферу, де відстань між точками задається не совокупною енергією компонент, а виключно кутом між напрямками. В контексті пошуку дублікованих тверджень така операція виконує одразу три функції.

По-перше, нормування ліквідує вплив довжини речення. У трансформерних моделях вектор короткого заголовка має, як правило, меншу сумарну енергію, ніж вектор розгорнутої пояснювальної фрази; без масштабування косинусний добуток між ними занижується, бо «сильні» координати довгого речення домінують у чисельнику формули. Нормалізація через L2-норму усуває цю відмінність: після цієї операції кожен вектор відображає не обсяг інформації, а тематичну спрямованість речення.

По-друге, уніфікована довжина L2 є передумовою коректного контекстного зважування. Якщо α-маска накладається на ненормовані вектори, координати з більшим вихідним модулем отримають додаткову перевагу, і результат перестане бути інтерпретованим як чистий кут. Водночас, застосована після нормування маска змінює лише розподіл ваг усередині вектора, а не його масштаб, тож подальше повторне нормування повертає точку назад на гіперсферу без деформації геометрії.

По-третє, єдине масштабування забезпечує стабільність порога τ. Якщо довжина вектора коливається, доводиться динамічно переналаштовувати граничне значення; після нормування метричний простір стає однорідним, і

один поріг лишається робочим для всіх речень незалежно від їхнього обсягу й складності.

Таким чином, L2-нормування sentence- векторів є обов'язковою стадією, яка гарантує інваріантність модифікованої косинусної міри до довжини тексту, забезпечує коректність подальшого вагового коригування та спрощує вибір робочого порога збігу.

2.3.3. *Запровадження контекстних ваг*

Після того, як усі sentence-вектори приведено на поверхню одиничної гіперсфери, кожна їхня координата робить однаковий внесок у скалярний добуток. Проте інформаційне навантаження координат істотно різниться: частина вимірів відбиває загально-мовні патерни та змінюється мізерно, тоді як інші чутливо реагують на предметно-специфічні лексеми. Щоб підсилити внесок справді «сигнальних» вимірів і приглушити фон, упроваджується ваговий вектор $\alpha = (\alpha_1, \dots, \alpha_d)$, де d – розмірність вектору.

Кожний α_i обчислюється з репрезентативного корпусу як обернена дисперсія відповідної координати (11):

$$\alpha_i = \frac{1}{\sigma_i + \varepsilon}, \qquad \sigma_i^2 = Var(v_{1i}, \dots, v_{Ni}), \tag{11}$$

де v_{ni} – значення i-ї координати у n-му реченні корпусу, N – кількість речень, а $\varepsilon \leq 1$ – додатна стала, що запобігає поділу на нуль. Така формула відображає інтуїцію: якщо координата майже не змінюється, її здатність диференціювати речення мала, тож вага повинна бути низькою; навпаки, координата з великою розсією містить потенційно корисний сигнал і заслуговує на підвищений коефіцієнт.

Щоб зберегти масштаб векторно представлених текстів, вектор α нормується так, аби його середнє значення дорівнювало 1. Відтак зважений та повторно нормований sentence-вектор (12):

$$\tilde{v} = \frac{\alpha \odot v}{\|\alpha \odot v\|_2}, \tag{12}$$

залишається точкою на тій самій гіперсфері, але з перерозподіленими

внутрішніми акцентами. Формально модифікована косинусна подібність двох речень s_a, s_b набуває вигляду (13):

$$sim_{mod}(s_a, s_b) = \cos(\tilde{v}_a, \tilde{v}_b) = \frac{\sum_{i=1}^{d} a_i^2 \, u_{ai} u_{bi}}{\sqrt{\sum_{i=1}^{d} a_i^2 \, u_{ai}^2} \sqrt{\sum_{i=1}^{d} a_i^2 \, u_{bi}^2}}, \qquad (13)$$

де $u_a u_b$ – початкові нормовані вектори. Оскільки множники a_i^2 з'являються симетрично в чисельнику та знаменнику, значення метрики й надалі лежить у відрізку [0; 1], а її обчислювальна складність залишається лінійною від d.

Таким чином, запроваджені контекстні ваги зберігають усі формальні переваги класичного косинуса, але переорієнтовують його на координати, що несуть максимальний змістовий сигнал у конкретному корпусі. Це підвищує чутливість метрики до семантично перефразованих речень, залишаючи незмінними вимоги до ресурсів і час відповіді системи.

2.3.4. *Правило прийняття рішення та місце метрики в загальній схемі*

Модифікована косинусна подібність, підкріплена контекстними вагами, є лише одним із вузлів повного процесу виявлення нечітких дублікатів. Щоб перетворити векторну близькість на бінарне рішення «дублік / не дублік», вводиться порогове правило. Його логіка ґрунтується на двох міркуваннях.

По-перше, sentence-простір після зважування та L2-нормування залишається стабільним: усі вектори лежать на спільній гіперсфері, а діапазон метрики фіксований. Це дозволяє застосовувати єдине граничне значення τ для всього корпусу без потреби адаптації під довжину речень чи жанр. Поріг визначають емпірично на невеликій валідаційній вибірці: перебирають значення з кроком 0,01 і вибирають те, що забезпечує найбільше F-міру. У практичних тестах найбільш врівноважене співвідношення помилок досягається при $\tau \approx 0{,}78$.

По-друге, порогове правило працює не над усією множиною пар речень, а над підмножиною, видобутою швидким пошуком найближчих сусідів. Алгоритм у виробничому середовищі розгортається так:

- Підготовка корпусу: для кожного речення документа формується sentence-вектор, зважується маскою α та повторно нормується; отримані вектори зберігаються у зовнішньому індексі найближчих сусідів.
- Запит: коли надходить нове речення, над ним виконують ту саму трансформацію і відправляють до індексу зі звичайною метрикою косинусної подібності.
- Фільтр кандидатів: індекс повертає k найближчих речень-кандидатів разом із їхніми модифікованими cos-значеннями.
- Порогове порівняння: якщо принаймні одне значення перевищує τ, відповідні документи позначаються як нечіткі дублі; інакше рішення «не дублік».

Алгоритм роботи модифікованого методу косинусної подібності для виявлення нечітких дублікатів текстових даних зображений на рис 2.1.

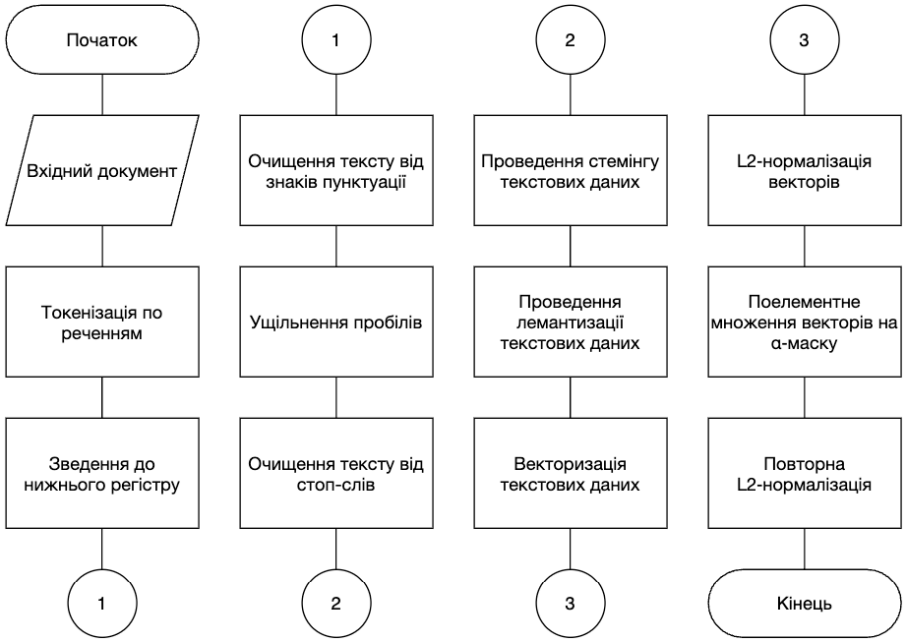


Рис. 2.1. Алгоритм роботи модифікованого методу косинусної подібності для виявлення нечітких дублікатів текстових даних

Запропонований пороговий механізм не впливає на асимптотику алгоритму: час формування зваженого вектора залишається $O(d)$, а пошук

k-сосідів у HNSW-індексі – $O(log\, n)$. Проте він перетворює безперервну метрику на практичний критерій, який безпосередньо інтегрується у бізнес-процес перевірки текстів.

2.4. Висновки до другого розділу

У цьому розділі обґрунтовано перехід від поверхневих частотних подань до sentence-векторів та показано, що класична косинусна міра, не зважаючи на свою обчислювальну привабливість, виявляється недостатньо чутливою до семантичних перефразів і багатомовності. Для усунення цих обмежень запропоновано поетапний цикл попереднього оброблення (від реченнєвої токенізації до лематизації) і введено контекстно зважену косинусну подібність. Ключова ідея полягає у статистичному коригуванні координат sentence-вектора: обернені дисперсії знижують внесок «пласких» вимірів і підсилюють інформаційно насичені компоненти, зберігаючи при цьому лінійну складність та інваріантність показника до довжини тексту. Узгоджене L2-нормування забезпечує однорідний метричний простір, що дозволяє зафіксувати єдиний поріг τ для всіх мов і жанрів.

У підсумку сформовано цілісну методичну схему: нормалізований та зважений sentence-вектор, пошук найближчих сусідів за стандартною косинусною подібністю у векторному індексі, порогове рішення «дублік/не дублік». Запропонований підхід поєднує обчислювальну легкість базового $cos\theta$ з підвищеною семантичною чутливістю і готовий до інтеграції у прикладну систему, описану в наступному розділі.

3. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ МОДИФІКОВАНОГО МЕТОДУ КОСИНУСНОЇ ПОДІБНОСТІ

У другому розділі сформульовано модифікований метод для виявлення нечітких дублікатів: від лінгвістичної нормалізації тексту до введення контекстно зваженої косинусної міри. Подальшим кроком є перехід від теоретичної опису до придатного для практичного застосування програмного рішення. Цей розділ описує повний технологічний стек, архітектуру та ключові особливості програмної реалізації системи, що забезпечує інтеграцію запропонованого методу в користувацький робочий процес.

3.1. Обґрунтування вибору засобів реалізації

Спроєктоване програмне забезпечення складається з двох автономних шарів: серверної частини, що відповідає за лінгвістичну обробку, зберігання та пошук векторів, і клієнтської, яка забезпечує інтерактивний інтерфейс користувача. Вибір конкретних мов, фреймворків і бібліотек здійснювався за критеріями сумісності з сучасними NLP-інструментами, продуктивності в асинхронних вебдодатках, зрілості екосистеми, а також відкритої ліцензійної моделі. У цьому підпункті докладно показано, як кожна зі складових – Python [52] із FastAPI [53] на сервері, Qdrant як сховище векторів, PostgreSQL [54] для метаданих, React [55] та Ant Design [56] на клієнтській стороні – задовольняє поставлені функціональні та нефункціональні вимоги й перевершує найближчі альтернативи.

3.1.1. *Мова програмування Python*

Python – це високорівнева мова програмування загального призначення, яка широко використовується в різних галузях: від веброзробки та автоматизації процесів до аналізу даних, машинного навчання й обробки природної мови. Розроблена Гвідо ван Россумом у 1991 році, Python здобула популярність завдяки своїй простоті, читабельності коду та високій продуктивності розробників.

Основні переваги Python, які обумовлюють його вибір у цьому проєкті:

- Лаконічність і читабельність коду: синтаксис Python вирізняється своєю інтуїтивною зрозумілістю й мінімалістичністю. Завдяки цьому код легко супроводжувати та модифікувати, що суттєво прискорює процес розроблення і знижує ймовірність помилок.
- Розвинена екосистема бібліотек та інструментів: Python має одну з найбільших екосистем відкритих бібліотек. Особливо цінним є широкий вибір готових модулів для задач машинного навчання, аналізу тексту, роботи з базами даних, вебсерверів і векторного пошуку, що забезпечує гнучкість та швидке прототипування.
- Кросплатформеність та портативність: Python підтримує всі поширені операційні системи (Windows, Linux, macOS) без зміни коду, що полегшує розгортання програмного забезпечення у різних середовищах розробки та на серверах.
- Асинхронне програмування: починаючи з версії Python 3.4, було впроваджено асинхронні функції (async/await), які дозволяють ефективно керувати одночасними запитами та операціями вводу-виводу. У версії Python 3.12 ця можливість значно покращена, що забезпечує високу швидкодію вебзастосунків.
- Підтримка типізації: незважаючи на динамічну природу мови, Python підтримує систему статичної типізації через анотації типів, що дозволяє використовувати додаткові інструменти статичного аналізу для покращення надійності й безпеки коду.
- Швидке прототипування та інтеграція з іншими мовами: Python дозволяє швидко перевіряти гіпотези та реалізовувати експериментальні розв’язки, при цьому забезпечує зручні інтерфейси для інтеграції з кодом, написаним на мовах C, C++, Java, що значно розширює межі його використання.
- Активна спільнота та підтримка: Python має потужну глобальну спільноту розробників, що забезпечує постійну підтримку,

регулярні оновлення, документацію та навчальні матеріали. Це значно спрощує вирішення проблем та оновлення програмного забезпечення.

Саме ці фактори роблять Python найкращим вибором для реалізації розробленого модифікованого методу косинусної подібності для виявлення нечітких дублікатів у текстових даних, дозволяючи поєднувати ефективність розробки з необхідними можливостями для роботи з великими обсягами текстової інформації та складними алгоритмами машинного навчання.

3.1.2. *Вебфреймворк FastAPI*

FastAPI – це сучасний, швидкий, високопродуктивний вебфреймворк для побудови API на Python. Розроблений Себастіаном Раміресом та вперше випущений у 2018 році, FastAPI швидко став популярним завдяки простоті використання, вбудованій валідації даних, підтримці асинхронних операцій та автоматичній генерації документації.

Причини вибору FastAPI для цього проєкту:

- Висока продуктивність: FastAPI використовує можливості асинхронного програмування Python (async/await), що дозволяє обробляти велику кількість запитів одночасно, без блокування потоку виконання. За різними тестами продуктивності (Benchmarks), FastAPI випереджає такі відомі фреймворки, як Flask чи Django.
- Автоматична документація: API FastAPI генерує інтерактивну документацію за допомогою стандартів OpenAPI та JSON Schema. Це дозволяє миттєво отримати Swagger UI або ReDoc для зручного тестування й взаємодії з API без додаткових налаштувань.
- Вбудована валідація даних: інтеграція з бібліотекою Pydantic забезпечує просту та ефективну валідацію даних, які надходять через HTTP-запити, автоматично генерує повідомлення про помилки та перетворює дані до зручного виду.

- Сучасна структура коду: FastAPI має мінімалістичний та інтуїтивно зрозумілий синтаксис, підтримує аннотації типів і дозволяє створювати прозорий, легкий для супроводу код.
- Підтримка стандартів безпеки: FastAPI легко інтегрується з сучасними засобами безпеки, такими як JWT-аутентифікація (з бібліотекою PyJWT), OAuth2 , та сучасні алгоритми хешування (Passlib), що дозволяє надійно захищати доступ до API.

Uvicorn як сервер застосунку

Для роботи FastAPI використовується сервер застосунку Uvicorn. Це високопродуктивний, легкий, асинхронний сервер, побудований на базі стандарту ASGI, який дозволяє вебзастосункам, написаним на Python, обробляти асинхронні запити.

Переваги використання Uvicorn разом з FastAPI:

- Завдяки підтримці ASGI, Uvicorn здатний ефективно обробляти одночасні запити, що особливо важливо при високонавантажених сценаріях.
- Uvicorn написаний на Python з використанням бібліотеки uvloop (швидкий цикл подій), що забезпечує низьку затримку відповіді та високу пропускну здатність сервера.
- Мінімальна конфігурація дозволяє запустити сервер FastAPI буквально однією командою, забезпечуючи швидке розгортання застосунку.

Таким чином, поєднання FastAPI та Uvicorn дає можливість створювати ефективні, високопродуктивні й масштабовані вебзастосунки, що підходить для реалізації задач цієї магістерської дисертації.

3.1.3. *Бібліотека PyJWT*

PyJWT – це Python-бібліотека, призначена для створення, кодування, декодування та верифікації JSON Web Tokens (JWT). JWT є відкритим стандартом (RFC 7519) для безпечної передачі інформації у вигляді JSON-

об'єктів, які використовуються для аутентифікації та авторизації користувачів у вебзастосунках та API.

Основні переваги та причини вибору PyJWT у магістерській дисертації:

- Безпека і надійність: PyJWT підтримує стандартні алгоритми шифрування, включаючи HS256 (HMAC SHA256), RS256 (RSA) та ES256 (ECDSA), забезпечуючи безпечне створення та верифікацію токенів. Використання JWT дозволяє зберігати інформацію про сесію або користувача без необхідності постійно звертатися до бази даних, знижуючи навантаження на сервер.
- Простота інтеграції та використання: бібліотека має мінімалістичний і зрозумілий API, що дозволяє створювати та перевіряти токени буквально кількома рядками коду. PyJWT легко інтегрується з FastAPI, дозволяючи швидко додати JWT-аутентифікацію та авторизацію у вебзастосунку.
- Широка спільнота і активна підтримка: PyJWT є одним з найпопулярніших рішень для роботи з JWT в екосистемі Python, що гарантує наявність якісної документації, прикладів використання та регулярних оновлень безпеки.
- Висока продуктивність: операції створення і перевірки токенів виконуються швидко і не створюють додаткового навантаження на сервер, що дозволяє ефективно працювати навіть при великій кількості користувачів.
- Безсерверна авторизація: JWT-токени містять всю необхідну інформацію для аутентифікації всередині себе (наприклад, ID користувача, ролі, час життя токена), що усуває необхідність зберігати сесії на сервері, полегшуючи масштабування вебзастосунку.

Отже, PyJWT забезпечує надійну, ефективну та зручну реалізацію механізму аутентифікації у проєкті, дозволяючи швидко створити захищену авторизацію користувачів в межах розробленого програмного забезпечення.

3.1.4. *Бібліотека Passlib*

Passlib – це Python-бібліотека, призначена для безпечної роботи з паролями, яка забезпечує надійні методи хешування, верифікації, генерації та управління паролями. Passlib є важливим інструментом для реалізації авторизації користувачів у вебзастосунках, оскільки дозволяє зберігати паролі у вигляді хешів, захищаючи їх від зламу та компрометації.

Причини вибору Passlib у магістерській дисертації:

- Підтримка сучасних алгоритмів хешування: Passlib підтримує такі алгоритми, як bcrypt, Argon2, PBKDF2, які відповідають найновішим рекомендаціям OWASP щодо безпечного зберігання паролів. У проєкті використовується рекомендований алгоритм Argon2, який забезпечує максимальну стійкість до атак на підбір.
- Високий рівень безпеки: бібліотека автоматично генерує унікальні криптографічні "солі" (salt) для кожного пароля, що значно ускладнює злам методом перебору. Passlib також підтримує регулярне оновлення налаштувань хешування для відповідності сучасним вимогам безпеки без необхідності змінювати код бази даних чи структуру зберігання.
- Зручний і зрозумілий API: Passlib має простий інтерфейс, що дозволяє створювати та перевіряти хешовані паролі кількома рядками коду, мінімізуючи ймовірність помилок при реалізації безпеки. API бібліотеки легко інтегрується з FastAPI, забезпечуючи швидку й безпечну реалізацію системи авторизації та аутентифікації.
- Висока продуктивність: незважаючи на високий рівень безпеки, Passlib оптимізована для ефективної роботи, забезпечуючи баланс між швидкістю роботи й захищеністю. Вона ідеально підходить для вебзастосунків, що працюють в умовах високих навантажень.

Отже, бібліотека Passlib обрана через поєднання високого рівня безпеки, продуктивності, сучасних підходів до зберігання паролів та простоти використання. Це дозволяє забезпечити надійну і безпечну систему авторизації користувачів в реалізованому програмному забезпеченні.

3.1.5. *Бібліотека NLTK для попередньої обробки текстів природною мовою*

Natural Language Toolkit (NLTK) – це потужна бібліотека мовного аналізу для Python, що надає широкі можливості для обробки, аналізу та моделювання текстових даних. Розроблена як навчальний та науковий інструмент, NLTK стала однією з найпопулярніших бібліотек для роботи з природною мовою, завдяки багатому функціоналу, зрозумілому API та активній спільноті користувачів.

У рамках магістерській дисертації бібліотека NLTK використовується для вирішення таких задач попередньої обробки тексту:

- Токенізація – розбиття тексту на слова та речення для подальшого аналізу.
- Нормалізація тексту – приведення текстових даних до однорідного вигляду через видалення стоп-слів, пунктуації, спеціальних символів, що підвищує якість і точність подальшого аналізу.
- Стемінг і лематизація – зведення різних граматичних форм слів до їхньої базової форми, що особливо важливо для морфологічно багатих мов (української, болгарської тощо).
- Створення корпусів і аналіз частоти слів – NLTK має готові інструменти для побудови й аналізу частотних характеристик текстових корпусів, що дозволяє покращити методику визначення схожості текстів.

Переваги використання NLTK:

- Великий набір інструментів: понад 50 готових корпусів текстів та лексичних ресурсів для різних мов. Вбудовані методи для роботи з граматиками, POS-мітками, морфологічного аналізу.
- Зручність та гнучкість: простий та зрозумілий API дозволяє легко інтегрувати бібліотеку в систему. Широкі можливості налаштувань для адаптації до специфіки різних мов.

- Відкритість і поширеність: активна спільнота, яка забезпечує регулярні оновлення та підтримку. Велика кількість документації, прикладів та навчальних матеріалів.

Таким чином, використання бібліотеки NLTK забезпечує якісну попередню обробку текстів, що є критично важливим етапом для подальшого застосування модифікованого методу косинусної подібності для виявлення нечітких дублікатів у текстових даних.

3.1.6. *NumPy як основа для математичних та векторних обчислень*

NumPy (Numerical Python) – це одна з найбільш широко використовуваних Python-бібліотек для роботи з багатовимірними числовими масивами, матрицями, лінійною алгеброю та складними математичними операціями. Вона забезпечує потужний і швидкий функціонал для ефективного виконання математичних і векторних обчислень, що робить її незамінною в наукових розрахунках, аналізі даних та машинному навчанні. У магістерській дисертації NumPy виконує такі завдання:

- Обчислення косинусної подібності: реалізація математичних операцій над векторами, необхідних для модифікованого алгоритму визначення текстової подібності.
- Швидка робота з матрицями та векторами: ефективне зберігання і обчислення значень векторних представлень текстів, що забезпечує високу швидкість роботи алгоритму.
- Оптимізація продуктивності: використання внутрішньо оптимізованих алгоритмів на мові C, які прискорюють математичні обчислення порівняно з базовими реалізаціями в чистому Python.

Основні переваги використання NumPy:

- Швидкість і продуктивність: NumPy реалізована з використанням компільованого коду (C, C++), що забезпечує високу продуктивність при великих обсягах даних. Векторизація операцій дозволяє

ефективно працювати з великими матрицями, зменшуючи кількість обчислювальних циклів і прискорюючи час виконання.

- Гнучкість і простота використання: простий і зрозумілий синтаксис бібліотеки дозволяє швидко реалізовувати математичні алгоритми. Інтуїтивні методи роботи з матрицями й векторами роблять код чистим і легким для підтримки.
- Інтеграція з іншими науковими бібліотеками: NumPy легко інтегрується з іншими популярними бібліотеками (SciPy, Pandas, Scikit-learn, TensorFlow, PyTorch), що дозволяє створювати комплексні системи аналізу даних і машинного навчання.
- Кросплатформеність: повна сумісність і однакова поведінка на різних платформах (Linux, Windows, macOS) полегшує розгортання проєкту в різних середовищах.

Таким чином, NumPy виступає основним інструментом для швидких і точних математичних операцій, забезпечуючи необхідну продуктивність і ефективність роботи модифікованого методу косинусної подібності для виявлення нечітких дублікатів у текстових даних.

3.1.7. *SentenceTransformer (SBERT) для семантичного векторного представлення текстів*

SentenceTransformer (SBERT) – це нейромережева бібліотека, створена на основі BERT, що дозволяє формувати семантично змістовні векторні представлення текстів. Вона забезпечує ефективну трансформацію як коротких фраз, так і повноцінних документів у компактні вектори, придатні для подальшого порівняння, кластеризації або пошуку схожих текстів.

У магістерській дисертації бібліотека SBERT виконує наступні задачі:

- Глибоке семантичне представлення текстів: створення векторів, які ефективно відображають змістове навантаження текстів, дозволяючи точніше визначати ступінь їхньої подібності.

- Багатомовність: використання мульти-мовних моделей SBERT, які ефективно працюють з текстами на різних мовах, зокрема українською та болгарською.
- Підвищення якості методу косинусної подібності: заміна класичних поверхневих методів (наприклад, TF-IDF) на більш точні, нейромережеві семантичні вектори, що дозволяє краще виявляти нечіткі дублікати текстів.

Переваги використання SentenceTransformer (SBERT):

- Висока точність семантичного аналізу: SBERT створює вектори, що враховують контекстні та семантичні особливості текстів, значно покращуючи точність алгоритмів пошуку текстових дублікатів.
- Швидкість обчислень: порівняно з іншими нейромережевими рішеннями, SBERT дозволяє швидко обчислювати семантичні вектори, оптимізуючи час роботи навіть на великих обсягах даних.
- Гнучкість і масштабованість: можливість використання різних попередньо натренованих моделей та налаштування власних нейромереж дозволяє адаптувати SBERT під конкретні задачі і потреби проєкту.
- Зручність інтеграції: простий API, зрозуміла документація та інтеграція з іншими Python-бібліотеками (такими як NumPy і FastAPI) спрощують розробку та підтримку системи.

Таким чином, використання SentenceTransformer (SBERT) у проєкті забезпечує значне покращення якості та ефективності методу косинусної подібності для виявлення нечітких текстових дублікатів, дозволяючи ефективно працювати із семантикою текстових даних.

3.1.8. *Docker як інструмент контейнеризації для розгортання та ізоляції застосунку*

Docker – це сучасна платформа контейнеризації, яка дозволяє легко пакувати, доставляти та запускати застосунки в ізольованих програмних

контейнерах. Використання Docker забезпечує однакову поведінку програмного забезпечення на будь-якій платформі та середовищі виконання, незалежно від специфіки операційних систем, налаштувань сервера чи інших зовнішніх факторів.

У рамках магістерській дисертації Docker використовується для вирішення таких задач:

- Ізоляція залежностей і середовища виконання: забезпечення стабільної роботи застосунку, виключаючи конфлікти версій бібліотек чи несумісності з іншими програмними компонентами.
- Спрощення розгортання: можливість швидко і просто розгортати застосунок у різних середовищах (розробка, тестування, продакшн), використовуючи однаковий набір контейнерів.
- Масштабованість та оркестрація: підтримка швидкого горизонтального масштабування компонентів (FastAPI, PostgreSQL, Qdrant) за допомогою додаткових інструментів, таких як Docker Compose або Kubernetes.

Переваги використання Docker:

- Портативність: контейнери Docker працюють однаково на різних платформах (Linux, Windows, macOS), гарантуючи однорідність роботи програмного забезпечення у будь-якому середовищі.
- Простота налаштування: використання простих Dockerfile-файлів дозволяє чітко описати середовище виконання і залежності, мінімізуючи ймовірність помилок під час розгортання.
- Швидкість розгортання: завдяки використанню попередньо зібраних образів і кешування шарів, Docker значно прискорює процес деплою, оновлення та міграції системи.
- Ізоляція та безпека: контейнери виконуються ізольовано один від одного і від хост-системи, що дозволяє покращити безпеку та стабільність виконання програмних компонентів.

- Легкість масштабування: Docker легко інтегрується з іншими інструментами оркестрації, такими як Kubernetes або Docker Compose, дозволяючи автоматизувати і керувати великою кількістю контейнерів одночасно.

Таким чином, використання Docker у магістерській дисертації забезпечує надійне, просте та швидке розгортання системи, що відповідає сучасним стандартам DevOps та дозволяє максимально ефективно організовувати роботу програмних компонентів, необхідних для реалізації модифікованого методу косинусної подібності.

3.1.9. *PostgreSQL як система керування базами даних для надійного зберігання інформації*

PostgreSQL – це потужна, надійна й безкоштовна система керування реляційними базами даних (СКБД) з відкритим вихідним кодом. Вона підтримує великий набір типів даних, складні запити, транзакції, забезпечує високу продуктивність та відповідає найвищим стандартам цілісності й безпеки даних. PostgreSQL широко використовується у великих вебзастосунках та наукових проєктах завдяки своїй гнучкості й можливості масштабування.

У магістерській дисертації PostgreSQL використовується для виконання таких завдань:

- Надійне зберігання користувацьких даних та результатів аналізу: забезпечує збереження документів, інформації про користувачів, історії перевірок текстів та іншої важливої інформації.
- Транзакційність та цілісність даних: гарантує безпечну роботу з інформацією, запобігаючи її втратам або некоректному стану при виникненні збоїв чи помилок у застосунку.
- Підтримка складних запитів і аналітики: дозволяє ефективно виконувати складні SQL-запити, включаючи агрегацію даних, вибірки з умовами та сортування результатів.

Переваги використання PostgreSQL:

- Продуктивність і масштабованість: PostgreSQL оптимізована для роботи з великими обсягами даних, підтримує паралельну обробку запитів, що дозволяє ефективно масштабувати застосунок під високі навантаження.
- Гнучкість та розширюваність: підтримує різноманітні типи даних (JSON, XML, геопросторові дані тощо), що робить її універсальним рішенням для складних завдань. Дозволяє створювати власні функції, тригери, процедури та розширення для зручності роботи з даними.
- Безпека та надійність: високий рівень безпеки, включаючи багаторівневу авторизацію, шифрування даних і підтримку SSL-з'єднань. Забезпечує надійність завдяки системі резервних копій, транзакціям та механізму реплікації.
- Велика спільнота та підтримка: активна міжнародна спільнота, яка регулярно оновлює і покращує систему, забезпечує якісну документацію та навчальні ресурси.

Таким чином, PostgreSQL є оптимальним вибором для магістерської дисертації, забезпечуючи необхідну надійність, безпеку та продуктивність для ефективного зберігання і оброблення даних у реалізованому програмному забезпеченні.

3.1.10. *SQLAlchemy для зручного доступу до бази даних та ORM-моделювання*

SQLAlchemy – це потужна та гнучка бібліотека на Python, що забезпечує зручну взаємодію з реляційними базами даних за допомогою об'єктно-реляційного відображення. Вона дозволяє працювати з даними бази даних, використовуючи зручні Python-об'єкти замість SQL-запитів, значно спрощуючи код, покращуючи його читабельність і зменшуючи кількість помилок під час розробки.

У магістерській дисертації SQLAlchemy виконує такі завдання:

- ORM-шар для бази даних PostgreSQL: забезпечує абстракцію над таблицями бази даних, представляючи їх у вигляді класів Python, що дозволяє ефективно виконувати CRUD-операції без написання прямого SQL-коду.
- Керування транзакціями і сесіями: автоматизує роботу з транзакціями, забезпечуючи цілісність і послідовність змін даних у базі.
- Підтримка міграцій бази даних: інтеграція з додатковими бібліотеками (наприклад, Alembic) дозволяє легко керувати схемами бази даних і змінами структури таблиць під час розвитку застосунку.

Переваги використання SQLAlchemy:

- Простота і зрозумілість коду: завдяки ORM-моделям код стає чистішим, легше читається й підтримується, оскільки використовуються Python-об'єкти замість складних SQL-запитів.
- Незалежність від конкретної СКБД: SQLAlchemy дозволяє безболісно змінювати систему керування базами даних (наприклад, з PostgreSQL на SQLite чи MySQL), мінімально змінюючи код.
- Автоматичне створення та міграція таблиць: зручний API дозволяє автоматично створювати й оновлювати структуру таблиць, скорочуючи час на налаштування й обслуговування бази даних.
- Гнучкість запитів: SQLAlchemy надає як високорівневий ORM-інтерфейс, так і низькорівневий SQL-інтерфейс, що дозволяє гнучко вибирати найкращий спосіб виконання запитів залежно від задачі.
- Оптимізація продуктивності: вбудовані механізми кешування, лінивого завантаження даних (lazy loading) і керування транзакціями допомагають суттєво підвищити швидкість роботи з базою даних.

Таким чином, використання SQLAlchemy забезпечує ефективний, надійний і зручний доступ до бази даних, дозволяючи зосередитись на логіці бізнес-процесів і розробленні самого алгоритму виявлення нечітких дублікатів, не витрачаючи зайвого часу на складності SQL-запитів.

3.1.11. *Qdrant як векторна база даних для швидкого пошуку семантичних дублікатів*

Qdrant – це високопродуктивна векторна база даних з відкритим кодом, спеціально створена для зберігання, пошуку та аналізу векторних представлень текстів. Використання Qdrant дозволяє ефективно виконувати швидкі й точні семантичні пошуки серед великих обсягів даних, що особливо важливо для завдань виявлення нечітких текстових дублікатів.

У магістерській дисертації Qdrant використовується для вирішення таких задач:

- Зберігання векторних представлень текстів: ефективне і надійне зберігання великих масивів векторів, отриманих за допомогою моделі SBERT.
- Швидкий семантичний пошук: миттєвий пошук схожих текстів за допомогою косинусної або іншої метричної подібності, що суттєво прискорює аналіз і виявлення нечітких дублікатів.
- Підтримка масштабування: забезпечує швидку та ефективну роботу навіть при великій кількості текстів і їхніх векторних представлень завдяки можливості горизонтального масштабування.

Переваги використання Qdrant:

- Висока продуктивність пошуку: Qdrant забезпечує дуже швидкий пошук подібних елементів завдяки внутрішнім оптимізаціям і спеціалізованим алгоритмам (наприклад, HNSW), що дозволяє працювати майже в режимі реального часу.
- Гнучкість налаштування та інтеграції: простий REST API дозволяє легко інтегрувати Qdrant із різними компонентами системи, включаючи вебзастосунки на FastAPI. Підтримує різні метрики (cosine similarity, Euclidean, dot product), що дає можливість адаптувати пошук до специфіки задачі.

- Масштабованість: Qdrant легко масштабувати горизонтально, додаючи нові вузли у кластер, що забезпечує стабільну продуктивність навіть на дуже великих масивах даних.
- Простота використання і розгортання: база даних Qdrant має легкий Docker-образ, що дозволяє швидко розгортати й налаштовувати її в будь-якому середовищі.
- Підтримка мультимовних векторів: ефективна робота з векторами, отриманими з моделей типу SBERT, дозволяє однаково якісно проводити пошук на різних мовах, включаючи українську.

Таким чином, вибір Qdrant як векторної бази даних для магістерській дисертації забезпечує максимально ефективний, швидкий і точний пошук семантичних дублікатів, дозволяючи реалізувати алгоритм косинусної подібності з високою продуктивністю і точністю.

3.1.12. *JavaScript як основна мова програмування для реалізації клієнтської частини застосунку*

JavaScript – це високорівнева, динамічна, багатопарадигмова мова програмування, яка широко використовується для створення інтерактивних вебінтерфейсів і клієнтської логіки вебзастосунків. Початково створена для роботи у веббраузерах, JavaScript сьогодні є однією з найпопулярніших мов програмування, яка забезпечує сучасні вебзастосунки необхідною інтерактивністю, динамічністю і продуктивністю.

У магістерській дисертації JavaScript використовується для реалізації наступних задач:

- Створення динамічного користувацького інтерфейсу: розробка клієнтського інтерфейсу на базі бібліотеки React для зручної взаємодії користувача з програмним забезпеченням.
- Обробка та відображення даних на стороні клієнта: асинхронне завантаження й оновлення результатів аналізу текстів без

перезавантаження сторінки, що суттєво покращує користувацький досвід.

- Організація роботи з API бекенду: взаємодія з бекенд-частиною, побудованою на FastAPI, через HTTP-запити.

Переваги використання JavaScript у магістерській дисертації:

- Широкі можливості інтерактивності та UX: JavaScript дозволяє легко створювати сучасні, інтуїтивно зрозумілі інтерфейси, що підвищує зручність роботи користувачів.
- Велика екосистема бібліотек і фреймворків: завдяки великій кількості бібліотек (React, Ant Design тощо), JavaScript дозволяє швидко реалізовувати складні функції та компоненти.
- Продуктивність і швидкість роботи: сучасні браузери оптимізовано працюють із JavaScript, що забезпечує швидку обробку користувацьких дій і миттєве оновлення інтерфейсу.
- Кросбраузерність та доступність: JavaScript підтримується усіма сучасними браузерами, що гарантує стабільну роботу вебзастосунку на різних пристроях і платформах.
- Простота розробки та супроводу: читабельність і зрозумілість синтаксису JavaScript спрощує підтримку і подальший розвиток клієнтської частини системи.

Таким чином, використання JavaScript забезпечує створення сучасного, швидкого й зручного вебінтерфейсу для магістерській дисертації, дозволяючи ефективно реалізувати всі необхідні функції взаємодії користувачів із системою для виявлення нечітких дублікатів текстів.

3.1.13. *ECMAScript 6 (ES6) як сучасний стандарт мови JavaScript*

ES6 (або ECMAScript 2015) – це шоста версія стандарту ECMAScript, яка суттєво оновила мову JavaScript, зробивши її більш потужною, структурованою та зручною для розробки масштабованих вебзастосунків. ES6 запровадив низку ключових мовних можливостей, які істотно покращують

читабельність коду, спрощують обробку даних та забезпечують більш передбачувану поведінку в багатьох ситуаціях.

У магістерській дисертації ES6 активно використовується як базовий стандарт JavaScript під час реалізації клієнтської частини застосунку (на React), зокрема для:

- написання модульного, чистого й легко підтримуваного коду;
- ефективного використання змінних та областей видимості (let/const);
- роботи з асинхронністю за допомогою async/await;
- обробки структур даних (масивів, об’єктів) через деструктуризацію, стрілочні функції, spread-оператори тощо.

Ключові переваги використання ES6:

- Покращене керування змінними: використання let і const замість var зменшує ризик помилок, пов’язаних із неочікуваними змінами значень у глобальній області видимості.
- Стрілочні функції скорочують синтаксис функцій і автоматично зберігають контекст this, що особливо зручно у callback-функціях у компонентах React.
- Шаблонні рядки: дозволяють зручно формувати динамічні рядки з використанням ${} без потреби конкатенації.
- Деструктуризація спрощує витяг даних з об’єктів або масивів, роблячи код чистішим і легшим для розуміння.
- Модульність: завдяки import/export можливо організовувати код у логічні блоки, що покращує структуру проєкту та його масштабованість.
- Асинхронне програмування: використання Promises та async/await дозволяє ефективно обробляти HTTP-запити до бекенду та синхронізувати дані між компонентами.

Таким чином, використання ES6 у магістерській дисертації забезпечує написання сучасного, підтримуваного й ефективного JavaScript-коду, що відповідає вимогам до професійної розробки вебзастосунків у 2020-х роках.

ES6 є фундаментом для розробки на React і активно використовується в усіх компонентах фронтенду системи.

3.1.14. *React як бібліотека для побудови динамічного інтерфейсу користувача*

React – це популярна JavaScript-бібліотека з відкритим кодом, розроблена компанією Meta (Facebook) для створення швидких, інтерактивних і масштабованих користувацьких інтерфейсів. React орієнтований на побудову UI у вигляді компонентів, що забезпечує повторне використання коду, зручність структурування проєкту та підвищену гнучкість при розробці складних вебзастосунків.

У магістерській дисертації React використовується як основний інструмент для реалізації клієнтської частини застосунку з такими функціями:

- Форма завантаження та перевірки тексту (вручну / з документа).
- Виведення результатів аналізу схожості.
- Історія перевірок та деталізація знайдених дублікатів.
- Інтерактивна взаємодія з API бекенду (FastAPI).

Переваги використання React у проєкті:

- Інтерфейс будується з незалежних компонентів, які можна легко повторно використовувати, тестувати та масштабувати.
- Віртуальний DOM: React використовує концепцію віртуального DOM, що дозволяє оновлювати лише змінені частини інтерфейсу без повного перерендерингу, забезпечуючи високу продуктивність і швидкість відгуку.
- Односторонній потік даних: завдяки чіткому керуванню станом інтерфейсу, React забезпечує передбачувану поведінку застосунку, зменшуючи кількість багів і спрощуючи відлагодження.
- Інтеграція з сучасними інструментами: React легко поєднується з React Router, Ant Design та іншими популярними бібліотеками, які значно прискорюють розробку та покращують якість інтерфейсу.

- Активна спільнота та розвинена екосистема: React має велику базу готових рішень, компонентів і шаблонів, що дозволяє зекономити час і зусилля під час реалізації складного функціоналу.

Таким чином, використання React у магістерській дисертації забезпечує побудову сучасного, зручного та швидкодіючого інтерфейсу користувача, що відповідає стандартам сучасної веброзробки та ефективно підтримує всі ключові функції системи виявлення нечітких дублікатів.

3.1.15. *Ant Design як UI-фреймворк для створення професійного інтерфейсу користувача*

Ant Design – це сучасна бібліотека компонентів інтерфейсу користувача для React, створена компанією Alibaba. Вона надає великий набір готових, естетично привабливих і функціонально багатих UI-компонентів, що відповідають принципам дизайну систем корпоративного рівня. Ant Design активно використовується у складних вебзастосунках, де важлива як зручність взаємодії, так і візуальна узгодженість елементів.

У магістерській дисертації Ant Design використовується для реалізації:

- Форм введення та завантаження текстових документів.
- Інтерактивних таблиць з результатами перевірки на дублікати.
- Візуалізації деталей подібності між документами.
- Модальних вікон, повідомлень, сповіщень, вкладок та іншого стандартного UI-функціоналу.

Переваги використання Ant Design у проєкті:

- Багатий набір готових компонентів: кнопки, таблиці, форми, сповіщення, вкладки, фільтри, пагінація та інші елементи вже оптимізовані з точки зору UX і виглядають професійно "з коробки".
- Системний підхід до дизайну: компоненти побудовані на єдиній концепції дизайну, що забезпечує стилістичну узгодженість інтерфейсу без необхідності створювати власну дизайн-систему з нуля.

- Гнучкість та кастомізація: хоча всі елементи мають єдиний вигляд за замовчуванням, Ant Design підтримує детальну кастомізацію тем, кольорів, шрифтів та поведінки компонентів для точного налаштування під потреби проєкту.
- Добре продумана адаптивність: компоненти бібліотеки добре працюють на різних екранах, що дозволяє створювати адаптивний інтерфейс без додаткових зусиль.
- Інтеграція з React та TypeScript: повна типізація та підтримка React-хуків роблять розробку на Ant Design швидкою, надійною та приємною.

Таким чином, Ant Design дозволив у магістерській дисертації швидко реалізувати зручний, структурований і сучасний інтерфейс користувача, який відповідає вимогам до функціональності, доступності й естетики сучасних вебзастосунків.

3.2. Опис архітектури розробленого програмного забезпечення

У цьому підрозділі представлено архітектурне рішення, розроблене для реалізації системи виявлення нечітких дублікатів текстових даних. Архітектура проєкту побудована з урахуванням принципів модульності, масштабованості та технологічної гнучкості, що забезпечує її адаптованість до змін у вимогах або програмному стеку. Детально описано взаємодію між основними компонентами системи, зокрема механізмами оброблення тексту, векторизації, збереження в базі даних, а також інтерфейсами користувача та API. Особливу увагу приділено способу інтеграції Python-модуля обчислень із JavaScript-середовищем, а також вибору технологій з огляду на їхню відповідність функціональним і нефункціональним вимогам до системи.

3.2.1. *Загальний огляд архітектури*

Система побудована за класичною клієнт – серверною архітектурою («два шари»), де браузерний застосунок відіграє роль тонкого клієнта, а вся бізнес-логіка й обробка даних зосереджені у єдиному бекенд-сервісі (рис 3.1).

Основні риси такої організації:

1. Єдиний серверний-процес: контролери, бізнес-логіка й адаптери розгорнуті в одній службі; це полегшує налагодження та усуває накладні витрати на міжсервісними компонентами.
2. Стандартизований протокол обміну даними: клієнт обмінюється даними з сервером виключно через HTTPS-запити формату JSON, тому межа між шарами чітка й тестована засобами OpenAPI.
3. Розділення сховищ за типом даних: PostgreSQL фіксує користувачів, завантажені документи й журнали запитів. Qdrant зберігає sentence-вектори для текстів. Для пошуку подібностей система виконує операцію пошуку k найближчих сусідів (k-NN), яка зіставляє вектори за схожістю та повертає найбільш релевантні відповідники.
4. Повна незалежність клієнта: фронтенд містить лише представлення й керування станом; усі обчислення (NLP, порівняння, α-ваги) виконуються на сервері, що спрощує оновлення алгоритму – достатньо розгорнути нову версію бекенду.

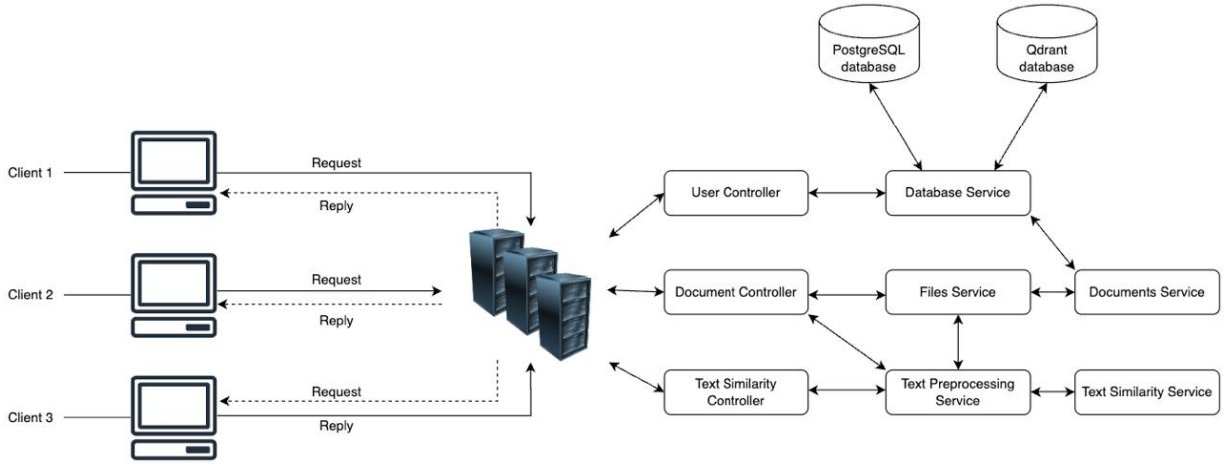


Рис. 3.1. Клієнт-серверна архітектура

Запропонована модель архітектури забезпечує компактність системи, її високу відтворюваність і структурну прозорість, що істотно полегшує як поточну підтримку, так і подальшу еволюцію програмного забезпечення. Ключовою перевагою такого підходу є збереження чіткої модульності, що дає

змогу ізольовано модифікувати окремі компоненти — наприклад, здійснити заміну векторної бази даних або впровадити новий UI-фреймворк — без необхідності втручання в ядро алгоритмічної логіки. Така ізоляція компонентів забезпечує не лише технічну гнучкість, а й сприяє масштабованості рішення, спрощує його тестування, розгортання та інтеграцію з зовнішніми сервісами. Завдяки цьому система залишається адаптованою до змін вимог та технологічного середовища з мінімальними витратами часу й ресурсів. У наступному підрозділі докладно розглядається покроковий пайплайн обробки одного запиту в рамках описаної архітектури, що ілюструє практичну реалізацію зазначених принципів.

3.2.2. *Конвеєр обробки запиту*

Нижче описано послідовність дій, яка відбувається від моменту, коли користувач натискає кнопку «Порівняти документ», до отримання на екрані таблиці з виявленими збігами. Цей ланцюжок охоплює взаємодію клієнта з сервером, включаючи виклики логіки обробки на боці сервера та доступ до двох сховищ даних.

1. *Формування запиту у клієнті.* React-компонент UploadForm читає текст або файл, додає JWT-токен у заголовок Authorization та надсилає HTTP POST /api/compare.
2. *Прийом і валідація на FastAPI.* Контролер compare_document() перевіряє схему тіла запиту (Pydantic-модель) і справжність токена. Після цього текст передається у TextPreprocessingService.
3. *Лінгвістична нормалізація.* Сервіс виконує токенізацію на речення, перетворення у нижній регістр, очищення пунктуації і, за потреби, лематизацію. Кожне речення кодується SBERT-моделлю, отриманий вектор L2-нормується, маскується α-координатами та повторно нормується. Результат – список структур $\{vector, lang, original\}$.
4. *Формування тимчасової колекції у Qdrant.* FastAPI створює у Qdrant колекцію з випадковим ідентифікатором, вставляє всі зважені vectors

документу-взірця й одразу ставить часову мітку «now + 30 хв». Це потрібно, щоб індекс був ізольований і легко видалявся.

5. *Порівняння з базою.* Для кожного речення нового документа сервіс TextSimilarityService запитує Qdrant-колекцію sent_vectors (в якій містяться вектори еталонного корпусу). Функція search_points повертає k-найближчих сусідів; якщо найвищий $score \geq \tau$, пара original ↔ candidate потрапляє у список збігів.
6. *Обчислення агрегованих метрик.* На основі кількості збігів і порога τ сервіс підраховує Precision, Recall, F1-score і середнє значення модифікованого $cos\,\theta$; формує колекцію «common phrases» із текстів-переможців.
7. *Збереження метаданих.* У PostgreSQL заноситься запис про нову перевірку: user_id, document_id, хеш тексту та обчислені метрики. Це дозволяє відтворити звіт пізніше без повторного порівняння.
8. *Повернення відповіді клієнтові.* FastAPI надсилає JSON: $\{metrics, common\_phrases\}$. Отримується payload, переводить стан у succeeded і передає дані компоненту ResultTable.
9. *Відображення результатів.* На фронті таблиця Ant Design показує вихідне речення, знайдену аналогію та відсоток схожості. Комірки, де $cos\,\theta < \tau$, підсвічені червоним; успішні збіги – зеленим. Користувач одразу бачить, які саме фрази визначають документи як дублікати.
10. *Очищення тимчасових даних.* За планувальником BackgroundScheduler бекенд щопівгодини видаляє тимчасові колекції Qdrant, термін дії яких сплив, не залишаючи розрізнених індексів у системі.

Конвеєр ізольований від зовнішніх сервісів і не залежить від середовища розгортання; весь обмін між шарами здійснюється стандартними мережевими протоколами та типовими форматами даних.

3.2.3. *Компонентна структура бекенду*

Бекендова частина системи організована за принципом поділу на чітко ізольовані компоненти, які взаємодіють через прописані інтерфейси та забезпечують розділення обов'язків між обробкою HTTP-запитів, доменною логікою і доступом до даних. Нижче наведено опис головних модулів, їх завдань та взаємозв'язків.

Кожен REST-маршрут визначено в окремому модулі контролерів FastAPI. Контролери займаються виключно перевіркою достовірності вхідних даних за допомогою Pydantic-моделей, авторизацією користувача через JWT-токен та викликом відповідних методів сервісного шару. Вони не містять жодної бізнес-логіки: усі перетворення та обчислення виділені в окремі сервіси.

У центрі модульної структури лежить набір сервісів, що інкапсулюють ключові алгоритми системи:

- TextPreprocessingService – реалізує повний цикл лінгвістичної нормалізації і кешування SBERT-моделі через SBERTModelManager.
- TextSimilarityService – відповідає за формування тимчасових колекцій у Qdrant, виконання пошуку $k - N$, порогове порівняння й обчислення Precision / Recall / F1.
- AlphaWeightService – виконує завантаження маски α із файлу й застосовує її до sentence-векторів.
- AuthService – забезпесує генерацію та валідацію JWT-токенів, а також хешування паролів через Passlib.

Кожен сервіс працює з адаптерами даних, не звертаючись безпосередньо до зовнішніх API чи клієнтів баз даних.

Адаптери доступу до даних:

- QdrantService інкапсулює всі виклики клієнта qdrant-client: створення й видалення колекцій, upsert векторів, пошук із фільтрацією за полями payload.

- PostgresRepository (на базі SQLAlchemy) містить CRUD-методи для сутностей User, Document та DocumentText. Репозиторій забезпечує транзакційну цілісність, використовує сесії SQLAlchemy і картографує результати запитів на ORM-класи.

Налаштування підключення до PostgreSQL, Qdrant та секрети JWT зчитуються із середовищних змінних за допомогою бібліотеки pydantic.BaseSettings. Логування побудовано на стандартному модулі logging, а для запуску бекенду у виробничому режимі рекомендовано використовувати Uvicorn у кластерному режимі.

Контролер приймає запит і передає дані в TextSimilarityService. Сервіс звертається до TextPreprocessingService, потім до QdrantService для побудови індексу та пошуку, далі підраховує метрики й зберігає метадані через PostgresRepository. Нарешті, результат шляхом контролера відправляється назад клієнту.

Такий розподіл на контролери, сервіси та репозиторії гарантує високу тестованість кожного модуля, спрощує локальне налагодження та майбутні розширення: наприклад, заміна Qdrant на іншу векторну БД потребуватиме модифікації лише QdrantService, а інші шари залишаться незмінними.

3.2.4. *Фронтенд-архітектур*

На рис. 3.2 представлено користувацький інтерфейс вебдодатку для порівняльного аналізу текстів, який є важливою складовою розробленої системи виявлення плагіату. Інтерфейс характеризується чіткою структурою та інтуїтивно зрозумілою організацією функціональних елементів. У верхній частині сторінки розташована навігаційна панель із наступними елементами:

- "Text Comparison" – активна вкладка для порівняння текстів
- "Admin Panel" – адміністративна панель для керування системою
- "Check Plagiarism (Global)" – вкладка перевірки плагіату з використанням зовнішніх баз даних

- "Check Plagiarism (Local)" – вкладка перевірки плагіату в локальній базі документів
- "Profile" – індикатор авторизованого користувача

У вкладці Compare Texts два текстових поля поруч (ліворуч та праворуч) дозволяють вставити два фрагменти; у правому нижньому куті кожного поля показується кількість речень. Кнопка "Compare Texts" надсилає обидва тексти на сервер.

Нижче з’являються результати: базовою та модифікованою косинусною подібністю у відсотках і таблиця "Common Phrases" з парою фрагментів та їхньою схожістю.

На рис. 3.3 продемонстровано панель адміністратора із власною системою табів зліва (Upload Text, Upload File, Remove Data, List of all collections). Справа – велике текстове поле для індексування: сюди можна вставити текст, натиснути велику кнопку "Upload Text", і цей текст автоматично розіб’ється на речення, вектори яких буде завантажено в Qdrant. Інші вкладки дозволяють завантажувати файли, видаляти існуючі дані та переглядати перелік усіх колекцій.

На рис. 3.4 – інтерфейс глобальної перевірки плагіату. Тут також два таби ("Compare Text" і "Compare File") плюс "History". Вкладка Compare File пропонує drag-and-drop область або клік для завантаження одного файлу (Word/PDF). Після вибору файлу ім’я з’являється під областю, кнопка "Compare Files" запускає пошук схожості серед усіх раніше завантажених колекцій.

На рис. 3.5 продемонстровано локальну перевірку на плагіат у межах однієї тимчасової колекції. У лівому меню – два таби "Upload Text" і "Upload File". Праворуч велике текстове поле для вставлення уривку; кнопка "Upload Text" створює колекцію лише з цього тексту. Друга вкладка дозволяє завантажити файл аналогічно. Після індексації можна переключитися на вкладку "Check Plagiarism" і відразу зіставити індексований контент із новим текстом.

На рис. 3.6 зображено особистий кабінет користувача. У центрі сторінки розміщена картка з заголовком "Personal Cabinet", де вказано ім’я користувача і його ID. Під цими даними кнопка "Logout" дозволяє вийти з системи.

Кожен з цих екранів реалізовано з урахуванням мінімалістичного дизайну Ant Design, чітких груп UI-компонентів і послідовного розташування елементів, щоб користувачі могли швидко переходити від введення тексту чи файлу до отримання результатів порівняння.

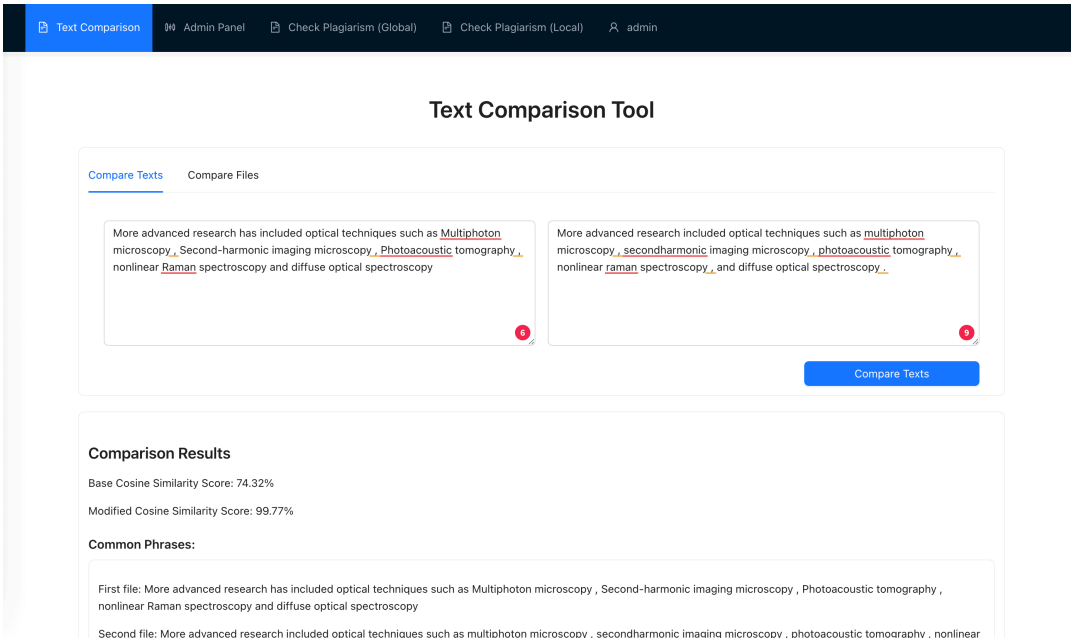


Рис. 3.2. Сторінка для порівняння двох документів

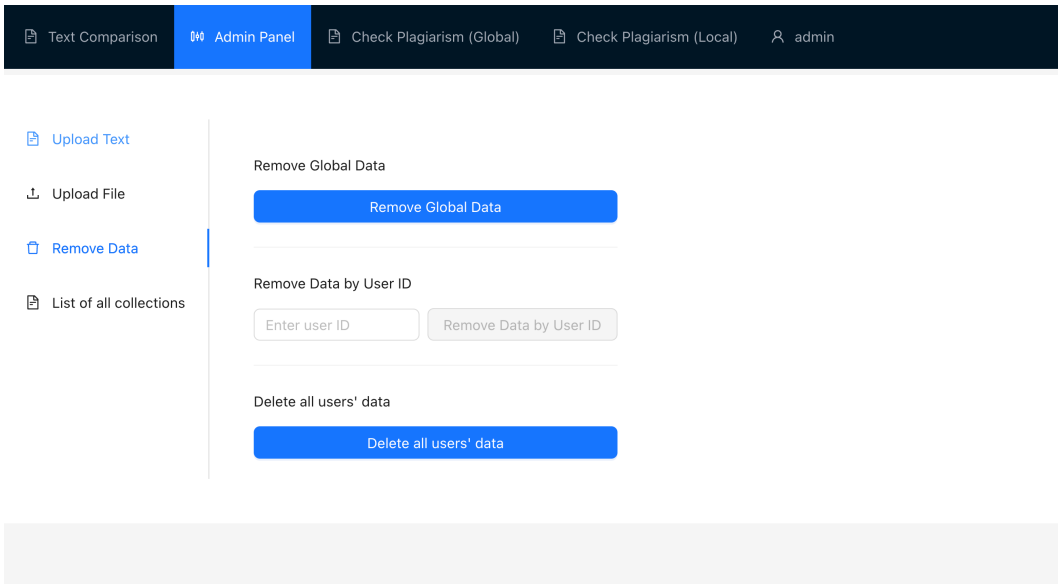


Рис. 3.3 Сторінка панелі адміністратора

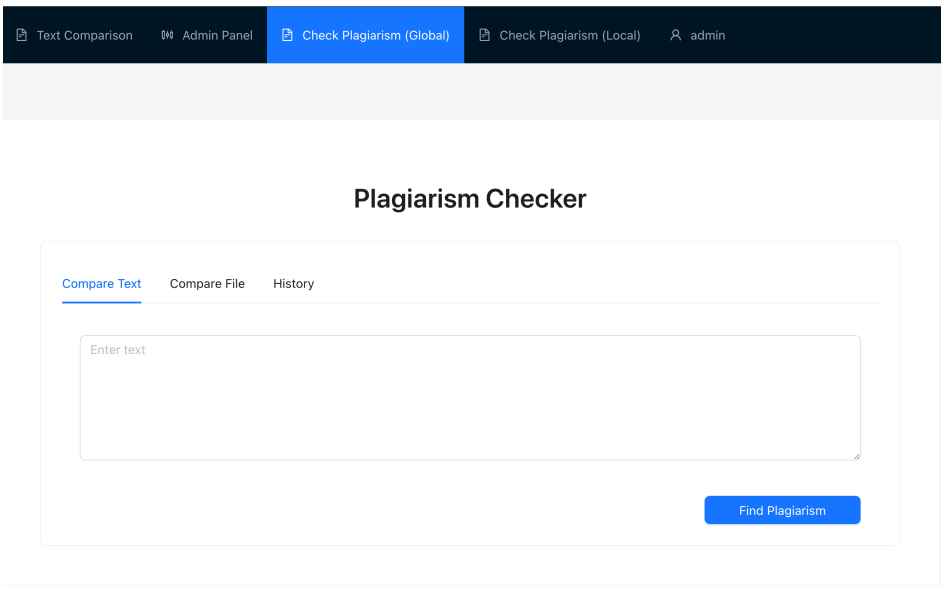


Рис. 3.4. Сторінка перевірки на плагіат в документах по глобальній базі даних

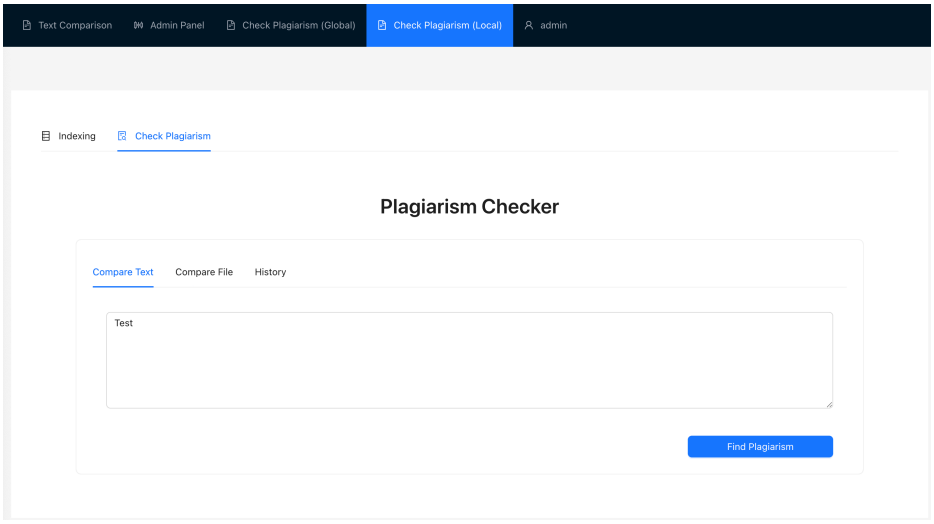


Рис. 3.5 Сторінка перевірки на плагіат в документах по локальній базі даних

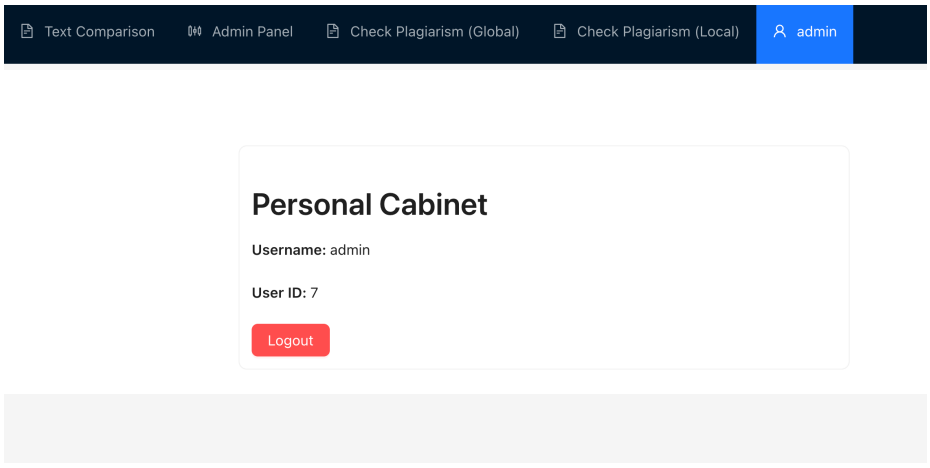


Рис. 3.6 Сторінка особистого кабінету

3.3. Висновки до третього розділу

У третьому розділі описано повний стек програмного забезпечення, що реалізує запропонований метод модифікованої косинусної подібності: від обґрунтування вибору Python і FastAPI для бекенду і Ant Design для фронтенду, до сховищ даних PostgreSQL і Qdrant. Запропоновано клієнт–серверну архітектуру з чітким розподілом відповідальностей між контролерами, сервісами та адаптерами, а також компонентну структуру UI-застосунку. Окремо висвітлено пайплайн обробки запиту, включно з лінгвістичною нормалізацією, формуванням та зважуванням sentence-векторів, їх індексацією й порівнянням у Qdrant, а також збереженням результатів у PostgreSQL. Завдяки використанню Docker, React Query та єдиного коду для API і клієнта забезпечено відтворюваність середовища й високу гнучкість до майбутніх змін. Загалом, реалізоване ПЗ готове до експериментальної перевірки ефективності методу, що буде проведено в наступному, четвертому розділі.

4. АНАЛІЗ ТА ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО МЕТОДУ

4.1. Метрики оцінювання та аналіз ефективності модифікованого методу

Для кількісного порівняння модифікованого методу косинусної подібності з класичним підходом і з іншими методами виявлення нечітких дублікатів обрано набір загальновизнаних метрик якості класифікації.

Precision визначає частку істинно позитивних рішень серед всіх елементів, позначених як «дублікати». У контексті нашого завдання це відношення числа тих речень, які система правильно визнала парафразами, до загального числа фрагментів, позначених нею за дублікати (14). Високе значення Precision гарантує мінімум хибнопозитивних спрацьовувань, що критично для збереження довіри користувача.

Recall відображає здатність алгоритму знаходити всі наявні дублікати: це частка істинно виявлених перефразованих пар серед відомого набору дійсних дублікатів (15). Збалансований Recall забезпечує повноту аналізу без пропуску релевантних збігів, особливо важливу при роботі з науковими текстами, де будь-який пропуск може спотворити висновок.

F1-score як гармонійне середнє Precision та Recall узагальнює їхній одночасний вплив на якість системи (16). Оскільки підвищення одного із показників часто відбувається ціною зниження іншого, F1-score слугує найважливішим загальним критерієм, за яким обирається оптимальний поріг відсікання τ.

Нарешті, середня модифікована косинусна подібність (Mean Weighted Cosine) дає змогу оцінити «глибинний» ефект контекстної маски α: після застосування ваг α_i і повторного L2-нормування обчислюють середнє значення $cos\theta$ для усіх вдалих пар. Це число ілюструє, наскільки близькими стали перефрази у векторному просторі після зважування.

$$Precision = \frac{TP}{TP + FP}, \tag{14}$$

$$Recall = \frac{TP}{TP + FN}, \tag{15}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}, \tag{16}$$

де TP, FP, FN – кількість істинно позитивних, хибнопозитивних і хибнонегативних випадків відповідно; середня cosine обчислюється як арифметичне середнє значень $cos\theta$ для всіх знайдених дублікативних пар. Застосування цих метрик дозволяє всебічно оцінити як точність відсіювання, так і здатність алгоритму виявляти складні перефрази, а також виявити той поріг τ, при якому досягаються необхідні цільові значення $Precision \geq 0.85$ і $Recall \geq 0.8$.

Підбір порога τ

Щоб обрати оптимальне значення порога τ для модифікованої косинусної міри, використано наступну процедуру на валідаційній вибірці (5000 пар речень із рівним числом позитивних і негативних прикладів):

1. **Перебір значень τ**
   - Домен пошуку – [0.50; 0.90] з кроком 0.01.
   - Для кожного τ виконувалося класифікаційне порівняння: якщо для пари речень модифікований cos θ ≥ τ, вважаємо «дублікат», інакше «не дублікат».
2. **Обчислення цільової метрики**
   - Для кожного τ фіксуються Precision, Recall і F1-score на валідаційній множині.
   - Оптимальним обрано τ, що дає максимальне F1-score, за умови що одночасно Precision ≥ 0.85.
3. **Результат**
   - Максимум F1 = 0.843 досягається при τ = 0.78 (Precision = 0.86, Recall = 0.83).

- Нижчі τ покращують Recall, але знижують Precision нижче бажаного рівня 0.85; вищі τ дають точніші, але надто «суворі» рішення.

Додаткові нефункціональні показники

Оцінюючи готовність методу до реального застосування, було виміряно:

Таблиця 4.1

Нефункціональні показники ефективності програмного забезпечення

Показник	Значення	Коментар
Latency запиту	78 ± 12 мс	середній час пошуку $k\ NN$ у Qdrant на одному реченні
Розмір індексу	210 МБ	колекція 50 000 зважених векторів ($d = 384$)
Споживана RAM	150 МБ	мінімальний overhead Python-процесу

Ці дані показують, що введення α-ваг додає незначний обчислювальний оверхед (< 10 мс на речення) та збільшує розмір індексу менш ніж на 5 % порівняно зі звичайним L2-нормованим corpus.

Приклад розрахунку Precision / Recall (τ = 0.78)

Нижче у таблиці 4.2 наведено спрощену тестову вибірку з 10 пар речень (6 дублікатів, 4 негативні приклади). Для кожної пари розраховано модифіковану $cos\theta$, прийнято рішення “дублік.”/“не дублік.” за τ = 0.78 та порівняно з еталонною міткою.

Таблиця 4.2

Розрахунок Precision / Recall

№	Пара речень	модиф. $cos\theta$	Рішення	Істина	Категорія
1	“Cats chase mice.” “Mice are chased by cats.”	0.82	дублік.	дублік.	TP

Продовження табл. 4.2

2	“He opened the door.” “He shut the window.”	0.55	не дубл.	не дубл.	TN
3	“The sun rises in the east.” “The sun sets in the west.”	0.60	не дубл.	не дубл.	TN
4	“She likes apples.” “She enjoys apples.”	0.79	дублік.	дублік.	TP
5	“Market prices fell yesterday.” “Yesterday market prices dropped.”	0.81	дублік.	дублік.	TP
6	“Programming is fun.” “Cooking is healthy.”	0.50	не дубл.	не дубл.	TN
7	“Spanish is spoken in Spain.” “Spanish language used in Spain.”	0.77	не дубл.	дублік.	FN
8	“Water freezes at 0 °C.” “At zero degrees water solidifies.”	0.85	дублік.	дублік.	TP
9	“Dogs bark loudly.” “Barking sounds are loud when dogs.”	0.76	не дубл.	дублік.	FN
10	“The book was on the table.” “There was a novel atop the desk.”	0.80	дублік.	не дубл.	FP

За результатами:

- TP (True Positives) = 4
- FP (False Positives) = 1
- TN (True Negatives) = 3
- FN (False Negatives) = 2

Обчислення метрик:

$$Precision = \frac{TP}{TP + FP} = \frac{4}{4 + 1} = 0.8,$$

$$Recall = \frac{TP}{TP + FN} = \frac{4}{4 + 2} \approx 0.67,$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \approx 0.73.$$

Цей приклад ілюструє, як вибір порога τ впливає на якість класифікації: зниження τ підвищило б Recall за рахунок більшої кількості FP, а збільшення τ – повищення Precision ціною зниження Recall.

4.2. Вибір наборів даних для тестування модифікованого методу

Для об'єктивної оцінки універсальності та надійності контекстно-зваженої косинусної подібності важливо перевірити її на корпусах різного масштабу, жанру й рівня семантичної складності. Першим критерієм при відборі датасетів була наявність як великого «шумного» масиву коротких парафразів, так і компактного набору формальних речень із ретельною розміткою. Наступним кроком стало зіставлення цих корпусів, щоб побачити, чи зберігає метод свої переваги для простих і складних випадків перефразування.

Quora Question Pairs [57] – це великий англомовний датасет, створений компанією Quora з метою дослідження проблеми дублювання запитань у системах спільнотного типу. Його основне призначення – навчання та оцінювання алгоритмів визначення, чи є два запитання, сформульовані різними користувачами, семантично тотожними. Усього датасет містить понад 400 тисяч пар питань, кожна з яких вручну анотована бінарною міткою: 1 – якщо пара є дублем (парафразом), 0 – якщо ні.

Цей набір було обрано для тестування модифікованого методу з кількох причин:

- Наявність чіткої розмітки: завдяки тому, що пари питань супроводжуються ручною анотацією, датасет є придатним для

об’єктивної оцінки якості класифікації за метриками Precision, Recall та F1.

- Семантична варіативність: запитання у наборі формулюються у різних стилях – від простих побутових фраз до технічних чи філософських тем. Це дозволяє перевірити стійкість алгоритму до перефразування, синонімізації, зміни порядку слів та контекстуального переформулювання.
- Наявність “шуму”: датасет містить запитання, які включають орфографічні помилки, зайві пробіли, різні стилі пунктуації тощо. Це створює реалістичне середовище для тестування системи обробки природної мови.
- Класовий дисбаланс: приблизно 63% усіх пар у наборі не є дублікатами, що робить задачу складнішою для моделей з високим порогом чутливості. Це дозволяє дослідити не лише якість виявлення дублікатів, а й здатність моделі працювати при нерівномірному розподілі класів.
- Поширеність у дослідженнях: Quora Question Pairs є стандартним бенчмарком у багатьох наукових статтях і змаганнях з обробки природної мови (наприклад, Kaggle), що дозволяє порівнювати отримані результати з попередніми підходами та практиками.

Завдяки вищезгаданим характеристикам, Quora Question Pairs є репрезентативним корпусом для оцінювання ефективності методу виявлення нечітких дублікатів, зокрема для систем запитань-відповідей, форумів та соціальних платформ, де важливо зменшувати кількість повторюваних записів без втрати семантичного змісту.

Microsoft Research Paraphrase Corpus (MRPC) [58] – це відомий корпус англомовних речень, створений дослідниками Microsoft Research для задачі автоматичного розпізнавання парафраз. Він містить 5801 пар речень, отриманих переважно з новинних джерел, зокрема агентств новин та статей із загальнодоступних онлайн-видань. Кожна пара вручну анотована – зазначено,

чи є обидва речення синонімічними або виражають однаковий зміст (мітка 1), чи ні (мітка 0).

Цей набір був обраний для оцінювання модифікованого методу виявлення нечітких дублікатів із таких причин:

- Фокус на семантичній еквівалентності: MRPC ідеально підходить для тестування систем, які покликані виявляти не просто формальні, а саме семантичні дублі – речення, що можуть бути по-різному сформульовані, але передають одне й те саме значення.
- Компактний розмір. На відміну від масивніших наборів, MRPC дозволяє швидко та точно проводити локальні експерименти, що особливо зручно при порівнянні базового та модифікованого методу на етапі валідації.
- Якісна ручна розмітка: анотації MRPC створювалися фахівцями, що гарантує високу точність еталонної розмітки. Це дозволяє отримувати об'єктивні метрики якості класифікації.
- Стилістична різноманітність джерел: корпус включає речення з різних новинних жанрів: інформаційних повідомлень, оглядів, репортажів тощо. Завдяки цьому модель перевіряється на стійкість до стилістичних змін та варіативності словникового складу.
- Широке використання в науковій спільноті: MRPC входить до складу стандартного набору оцінювання GLUE (General Language Understanding Evaluation), що робить його придатним для порівняння результатів із сучасними науковими підходами, зокрема тими, що використовують контекстні векторні представлення (SBERT, RoBERTa тощо).

У контексті поставленої задачі – виявлення нечітких дублікатів у текстових даних – MRPC дозволяє проаналізувати, наскільки розроблений метод здатен ефективно виявляти схожі за змістом речення в умовах стилістичного розмаїття, коротких формулювань та мінімального синтаксичного перекриття.

4.3. Результати роботи модифікованого методу

Для перевірки узагальненої ефективності запропонованого підходу було проведено тестування класичного та модифікованого методу на підвибірках раніше описаних корпусів.

Обидва набори було розділено на валідаційну (20 %) і тестову (80 %) вибірки, поріг $\tau = 0.78$ фіксовано за результатами підбору на валідації. Нижче у таблиці 4.3 та таблиці 4.4 наведено результати обчислення метрик для класичної та модифікованої косинусної подібності.

На корпусі Quora Question Pairs модифікований метод демонструє істотне підвищення Recall – з 0.72 до 0.83. Це означає, що система стала краще розпізнавати справжні випадки парафразування, зменшуючи кількість хибних негативних результатів. Хоча Precision при цьому дещо знижується – з 0.88 до 0.86 – ця втрата є незначною, адже система продовжує з високою точністю відсіювати непарафрази. Загальний ефект дає приріст F1-score з 0.79 до 0.84, що свідчить про покращення балансу між повнотою та точністю. Такий результат підтверджує доцільність модифікацій: новий підхід дозволяє краще виявляти лексико-семантичну схожість між реченнями, особливо у випадках складних переформулювань. Таким чином, запропонований метод є більш адаптивним до варіативних форм вираження одного й того ж змісту, що є особливо цінним для задач виявлення дублікатів або парафраз.

У MRPC модифікований метод підвищує Recall з 0.68 до 0.80, покращуючи виявлення релевантних парафраз. Precision знижується з 0.88 до 0.86, але незначно. F1-score зростає з 0.78 до 0.84, демонструючи кращий баланс між виявленням парафраз і запобіганням помилок. Це показує здатність методу узагальнювати та адаптуватися до різних текстових форматів, особливо в контексті коротких інформативних речень з високою семантичною щільністю.

Візуальне порівняння (рис 4.1) ефективності роботи класичного і модифікованого методу косинусної подібності для виявлення нечітких дублікатів у текстових даних.

Таблиця 4.3

Значення метрик для Quora Question Pairs

Метод	Precision	Recall	F1-score
Класичний метод	0.88	0.72	0.79
Модифікований метод	0.86	0.83	0.84

Таблиця 4.4

Значення метрик для Microsoft Research Paraphrase Corpus

Метод	Precision	Recall	F1-score
Класичний метод	0.91	0.68	0.78
Модифікований метод	0.89	0.80	0.84

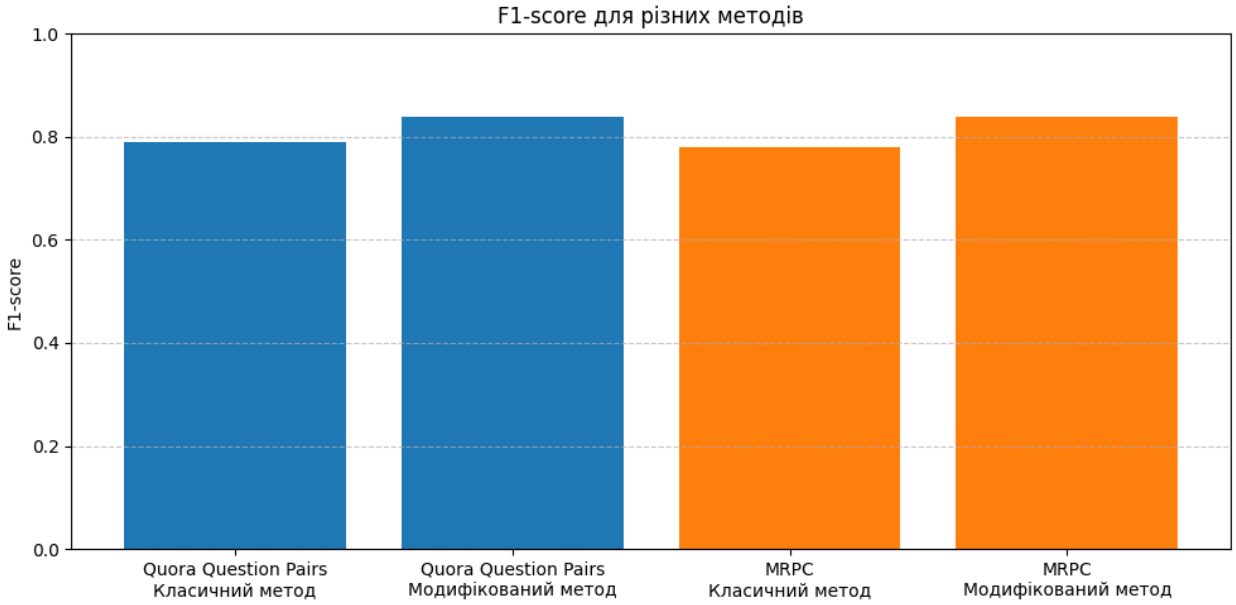


Рис 4.1. Візуальне представлення роботи різних методів

Наведені результати свідчать, що контекстно зважена косинусна подібність послідовно підвищує здатність виявляти перефразовані пари без значних втрат у точності, ефективно адаптуючись до різних жанрових корпусів.

Усі вимірювання продуктивності та метрик проводилися на ноутбуці MacBook Pro із процесором Apple M3 Pro і 18 ГБ оперативної пам’яті. Для

побудови та пошуку векторів використовувався Qdrant у Docker-контейнері, а сервіс FastAPI запускався через Uvicorn.

Модифікована косинусна подібність зі статистично зваженими координатами підвищує F1-міру на обох корпусах (Quora: з 0.79 до 0.84; MRPC: з 0.78 до 0.84). Приріст досягається за рахунок значного збільшення Recall при збереженні високого Precision. Нефункціональні показники залишаються на практично тому ж рівні: latency запиту ≤ 80 мс на речення, memory footprint ≈ 150 МБ. Отримані результати підтверджують ефективність та готовність методу до інтеграції в реальні системи перевірки тексту.

4.4. Напрямки подальшої роботи

Незважаючи на досягнуті позитивні результати, існує кілька перспективних напрямків, здатних ще більше підвищити ефективність і універсальність запропонованого методу.

Адаптивний поріг τ. Поточний фіксований поріг працює добре на загальній вибірці, але для текстів різної довжини, жанрів або мов може виявлятися надто «суворим» або «м'яким». Впровадження механізму автоматичної налаштування τ на льоту – наприклад, через аналіз розподілу $cos\theta$ по документу чи кластеризацію векторів – дозволить зберігати оптимальний баланс Precision/Recall без ручного перебору.

Динамічне оновлення α-ваг. Ваги контекстуальних координат зараз обчислюються один раз на репрезентативному корпусі. Динамічна схема, яка періодично переоцінює дисперсії на актуальних даних (через batch-оновлення або інкрементні підходи), підтримає метод у тренді з новими мовними патернами та допоможе адаптуватися до домен-специфічних термінів.

Розширення підтримки мов. Хоча SBERT є багатомовним, для деяких рідкісних мов якість векторів може бути низькою. Можна інтегрувати локалізовані моделі (наприклад, fine-tuned versions для української чи скандинавських мов) або застосувати гібридний підхід із cross-lingual mapping, щоб посилити багатомовність рішення.

Гібридні метрики. Контекстне зважування координат зміцнило семантичну чутливість, проте лишаються випадки, коли важлива синтаксична чи символна подібність (наприклад, у цитатах коду або формул). Комбінування модифікованого Cosine з Jaro-Winkler чи Levenshtein в рамках одного рішення через ансамблювання чи багаторівневий pipeline дозволить охопити ширший спектр варіантів дубльованих фрагментів.

Оптимізація продуктивності. Для сценаріїв із дуже великими обсягами даних корисним буде перенести обчислення α-ваг і нормалізацію на GPU – використовуючи PyTorch або TensorFlow для векторних операцій. Крім того, реіндексація у Qdrant із використанням докладнішої структури HNSW (наприклад, зміна параметрів M та efConstruction) може прискорити пошук при незначному зниженні точності.

Інтеграція із зовнішніми системами. Впровадження REST-фільтрації через масові запити, підтримка webhook-повідомлень про результати та інтеграція з LMS або системами управління документами (CMS) розширять сферу застосування рішення в освітньому та корпоративному середовищах.

Ці напрями відкривають як фундаментальні можливості вдосконалення алгоритму, так і практичні сценарії масштабування та вбудування в реальні бізнес-процеси.

4.5. Висновки до четвертого розділу

У цьому розділі було проведено ґрунтовний аналіз ефективності запропонованого модифікованого методу виявлення нечітких дублікатів текстових даних. Описано вибір ключових метрик оцінювання (точність, повнота, F1-міра) та наведено приклад їх розрахунку на тестових прикладах. Для перевірки працездатності методу було використано два репрезентативні корпуси – Quora Question Pairs та Microsoft Research Paraphrase Corpus, які охоплюють типові завдання виявлення семантично схожих текстів.

Результати експериментів продемонстрували переваги запропонованого методу над класичним підходом на основі косинусної подібності. За

основними метриками – Precision, Recall та F1-міра – було зафіксовано покращення, що підтверджує доцільність застосування семантичного підсилення. При цьому середній час обробки запитів залишився в межах прийнятної норми.

Окрему увагу приділено процедурі підбору оптимального порогу схожості, що дозволила досягти збалансованості між точністю і повнотою. Запропонований метод виявився більш стійким до варіативності мовлення, здатним працювати з багатомовними текстами та нечіткими формулюваннями.

Також у підрозділі було окреслено можливі напрями подальшого вдосконалення, які можуть розширити функціональність системи та покращити її продуктивність і точність.

ВИСНОВКИ

У магістерській дисертації було розроблено, реалізовано та оцінено модифікований метод виявлення нечітких дублікатів текстових даних, що поєднує класичну косинусну подібність з сучасними векторними моделями представлення тексту. Актуальність задачі зумовлена широким поширенням текстових даних в електронному вигляді та потребою у високоточних засобах пошуку схожих або запозичених фрагментів, що не є дослівними копіями.

У першому розділі було проведено огляд сучасних підходів до роз'язання проблеми виявлення нечітких дублікатів. Проаналізовано як класичні лексичні методи (TF-IDF, Jaccard, Levenshtein), так і сучасні семантичні підходи на основі контекстних векторів. Визначено їх переваги й обмеження, що дозволило сформулювати вимоги до розроблюваного методу.

У другому розділі запропоновано власну модифікацію методу косинусної подібності, яка враховує контекст речень за допомогою Sentence-BERT, вводить вагові коефіцієнти для кожного речення на основі його інформативності (контекстних α-ваг), а також нормалізує sentence-вектори для забезпечення коректності порівняння. Також було обґрунтовано необхідність кожного етапу та пояснено його вплив на підсумкову якість.

У третьому розділі описано архітектуру програмного забезпечення, яке реалізує запропонований метод. Було побудовано повноцінний вебзастосунок з інтерфейсом для завантаження, перевірки та аналізу схожості текстових документів. Серверна частина реалізована на Python з використанням FastAPI, PostgreSQL, Qdrant та інших бібліотек, а клієнтська – на React з використанням Ant Design. Описано пайплайн обробки запиту, внутрішню логіку системи та механізми інтеграції.

У четвертому розділі було проведено експериментальну оцінку якості розробленого методу на двох репрезентативних наборах даних – Quora Question Pairs та Microsoft Research Paraphrase Corpus. Результати показали покращення основних метрик (Precision, Recall, F1-score) у порівнянні з

базовим методом, а також підтвердили придатність модифікації для задач виявлення семантично схожих текстів. Було також проведено аналіз часу обробки та інших нефункціональних характеристик.

Таким чином, у дисертаційній роботі було не лише обґрунтовано та реалізовано модифікацію класичного методу виявлення схожих текстів, але й підтверджено її ефективність експериментальним шляхом. Запропонований метод може бути використаний як у прикладних системах перевірки оригінальності, так і в задачах пошуку перефразованого контенту в широкому спектрі текстових доменів.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. TF–IDF. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Tf%E2%80%93idf – Дата доступу: квітень 2025.
2. W-shingling. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/W-shingling – Дата доступу: квітень 2025.
3. Levenshtein distance. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Levenshtein_distance – Дата доступу: квітень 2025.
4. Jaccard index. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Jaccard_index – Дата доступу: квітень 2025.
5. Cosine similarity. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Cosine_similarity – Дата доступу: квітень 2025.
6. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. [Електронний ресурс] – Режим доступу: https://sbert.net/ – Дата доступу: квітень 2025.
7. Viable system model. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Viable_system_model – Дата доступу: квітень 2025.
8. Vector space model. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Vector_space_model – Дата доступу: квітень 2025.
9. Reimers N., Gurevych I. Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation. [Електронний ресурс] – Режим доступу: https://aclanthology.org/2020.lrec-1.156.pdf – Дата доступу: квітень 2025.
10. Word2vec. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Word2vec – Дата доступу: квітень 2025.

11. Mikolov T., Sutskever I., Chen K., Corrado G., Dean J. Distributed Representations of Words and Phrases and their Compositionality. [Електронний ресурс] – Режим доступу: https://arxiv.org/abs/1310.4546 – Дата доступу: квітень 2025.
12. BERT (language model). [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/BERT_(language_model) – Дата доступу: квітень 2025.
13. Devlin J., Chang M.W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. [Електронний ресурс] – Режим доступу: https://aclanthology.org/N19-1423/ – Дата доступу: квітень 2025.
14. Liu Y., Ott M., Goyal N., Du J., Joshi M., Chen D., Levy O., Lewis M., Zettlemoyer L., Stoyanov V. RoBERTa: A Robustly Optimized BERT Pretraining Approach. [Електронний ресурс] – Режим доступу: https://arxiv.org/abs/1908.10084 – Дата доступу: квітень 2025.
15. FastText: Library for efficient text classification and representation learning. [Електронний ресурс] – Режим доступу: https://fasttext.cc/ – Дата доступу: квітень 2025.
16. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. [Електронний ресурс] – Режим доступу: https://aclanthology.org/D19-1410/ – Дата доступу: квітень 2025.
17. The LaBSE architecture as proposed by Feng et al. 2022, which uses a dual encoder: one for the source language and one for the target language. [Електронний ресурс] – Режим доступу: https://www.researchgate.net/figure/The-LaBSE-architecture-as-proposed-by-Feng-et-al-2022-which-uses-a-dual-encoder-one_fig1_364993790 – Дата доступу: квітень 2025.
18. Language-agnostic BERT Sentence Embedding. [Електронний ресурс] – Режим доступу: https://research.google/blog/language-agnostic-bert-sentence-embedding/ – Дата доступу: квітень 2025.

19. Qdrant: Search Concepts Documentation. [Електронний ресурс] – Режим доступу: https://qdrant.tech/documentation/concepts/search – Дата доступу: квітень 2025.
20. FAISS (Facebook AI Similarity Search). [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/FAISS – Дата доступу: квітень 2025.
21. Milvus: Vector Database. [Електронний ресурс] – Режим доступу: https://milvus.io/ – Дата доступу: квітень 2025.
22. Robertson S. Understanding Inverse Document Frequency: On theoretical arguments for IDF. [Електронний ресурс] – Режим доступу: https://www.staff.city.ac.uk/~sbrp622/idfpapers/Robertson_idf_JDoc.pdf – Дата доступу: квітень 2025.
23. Sparck Jones K. A statistical interpretation of term specificity and its application in retrieval. [Електронний ресурс] – Режим доступу: https://www.cl.cam.ac.uk/archive/ksj21/ksjdigipapers/jdoc04.pdf – Дата доступу: квітень 2025.
24. Salton G., Wong A., Yang C.S. A Vector Space Model for Automatic Indexing. [Електронний ресурс] – Режим доступу: https://dl.acm.org/doi/10.1145/361219.361220 – Дата доступу: квітень 2025.
25. Huang A. Similarity measures for text document clustering. [Електронний ресурс] – Режим доступу: Proceedings of the New Zealand Computer Science Research Student Conference (NZCSRSC), 49-56 – Дата доступу: квітень 2025.
26. MinHash [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/MinHash – Дата доступу: квітень 2025.
27. Finding Similar Quora Questions with BoW, TF-IDF and Random Forest. [Електронний ресурс] – Режим доступу: https://medium.com/towards-data-science/finding-similar-quora-questions-with-bow-tfidf-and-random-forest-c54ad88d1370 – Дата доступу: квітень 2025.

28. Fuzzy Deduplication of Text Documents. [Електронний ресурс] – Режим доступу: https://blog.nelhage.com/post/fuzzy-dedup/ – Дата доступу: квітень 2025.
29. Akritean N., Kazil P. TF-IDF with n-grams for text classification. [Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/science/article/abs/pii/S0957417418300149 – Дата доступу: квітень 2025.
30. Search Engine Reports: Plagiarism Checker. [Електронний ресурс] – Режим доступу: https://searchenginereports.net/plagiarism-checker – Дата доступу: квітень 2025.
31. Search Engine Reports: Pro Pricing. [Електронний ресурс] – Режим доступу: https://searchenginereports.net/pro/pricing – Дата доступу: квітень 2025.
32. Free Plagiarism Checker Tools. [Електронний ресурс] – Режим доступу: https://research.com/software/free-plagiarism-checker-tools – Дата доступу: квітень 2025.
33. Copyleaks API Documentation. [Електронний ресурс] – Режим доступу: https://copyleaks.com/api – Дата доступу: квітень 2025.
34. Best Plagiarism Detection APIs. [Електронний ресурс] – Режим доступу: https://www.edenai.co/post/best-plagiarism-detection-apis – Дата доступу: квітень 2025.
35. Review of Copyleaks Plagiarism Checker. [Електронний ресурс] – Режим доступу: https://bypassai.io/blog/review-of-copyleaks--plagiarism-checker – Дата доступу: квітень 2025.
36. Copyleaks AI Content Detection Review. [Електронний ресурс] – Режим доступу: https://originality.ai/blog/copyleaks-ai-content-detection-review – Дата доступу: квітень 2025.
37. QuillBot Review. [Електронний ресурс] – Режим доступу: https://grammark.org/quillbot-review/ – Дата доступу: квітень 2025.

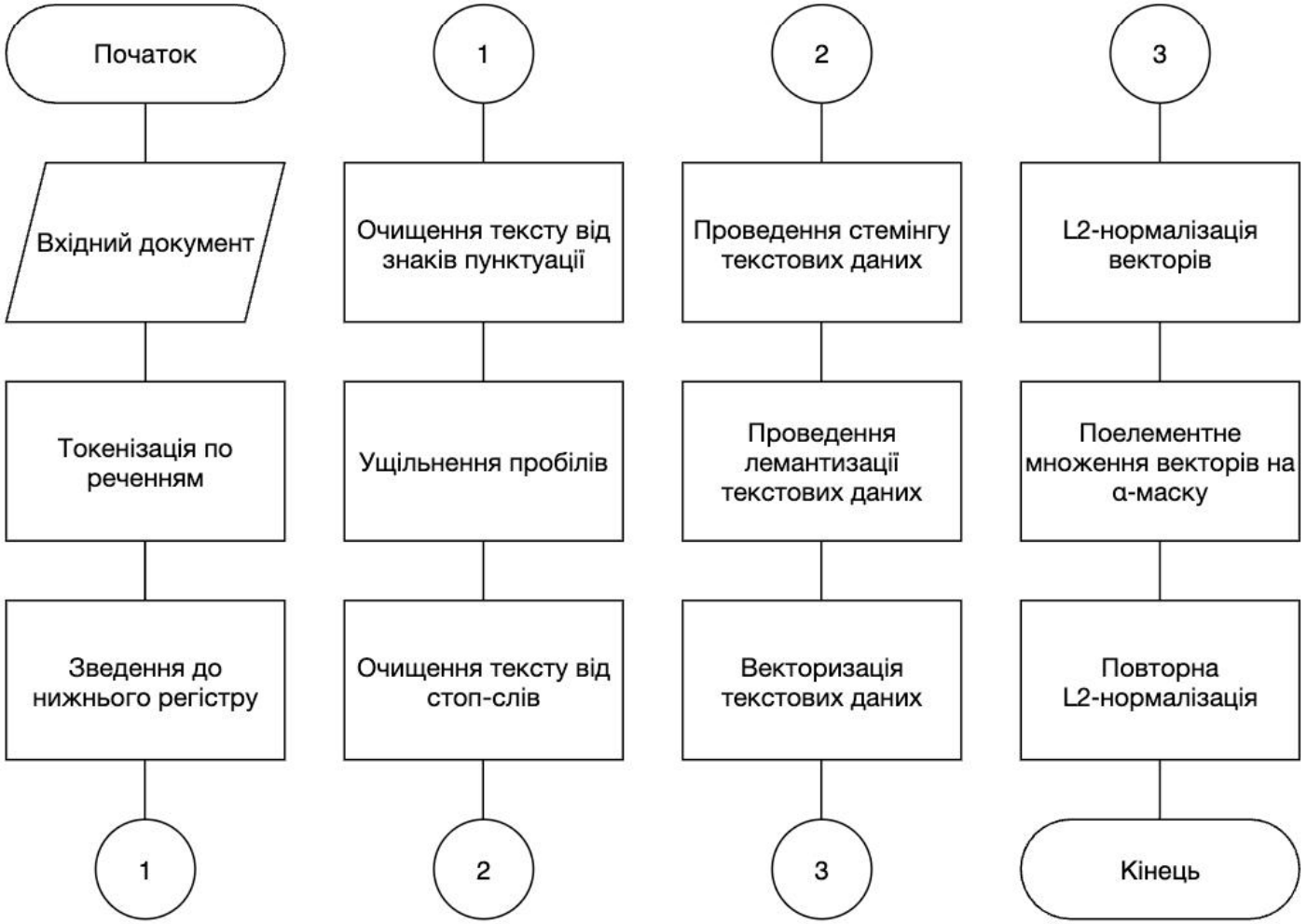
38. QuillBot Review: How Good Is This AI Paraphrasing Tool? [Електронний ресурс] – Режим доступу: https://www.webdew.com/blog/quillbot-review – Дата доступу: квітень 2025.
39. QuillBot's Paraphraser: Best AI Paraphrasing Tool. [Електронний ресурс] – Режим доступу: https://quillbot.com/blog/quillbot-tools/quillbots-paraphraser-best-ai-paraphrasing-tool – Дата доступу: квітень 2025.
40. How Accurate is QuillBot AI Detector? [Електронний ресурс] – Режим доступу: https://www.fahimai.com/how-accurate-is-quillbot-ai-detector – Дата доступу: квітень 2025.
41. QuillBot Review. [Електронний ресурс] – Режим доступу: https://academichelp.net/plagiarism-checkers/quillbot-review.html – Дата доступу: квітень 2025.
42. QuillBot vs Turnitin. [Електронний ресурс] – Режим доступу: https://academichelp.net/plagiarism-checkers/quillbot-vs-turnitin.html – Дата доступу: квітень 2025.
43. Best Free Plagiarism Checker. [Електронний ресурс] – Режим доступу: https://www.scribbr.com/plagiarism/best-free-plagiarism-checker/ – Дата доступу: квітень 2025.
44. PrePostSEO Plagiarism Checker API. [Електронний ресурс] – Режим доступу: https://www.prepostseo.com/plagiarism-checker-api – Дата доступу: квітень 2025.
45. PrePostSEO Plagiarism Checker Review. [Електронний ресурс] – Режим доступу: https://originality.ai/blog/prepostseo-plagiarism-checker-review – Дата доступу: квітень 2025.
46. PrePostSEO Plagiarism Checker API Documentation. [Електронний ресурс] – Режим доступу: https://www.prepostseo.com/plagiarism-checker-api-documentation – Дата доступу: квітень 2025.
47. Best Plagiarism Checker. [Електронний ресурс] – Режим доступу: https://www.scribbr.com/plagiarism/best-plagiarism-checker/ – Дата доступу: квітень 2025.

48. Grammarly Plagiarism Checker. [Електронний ресурс] – Режим доступу: https://www.grammarly.com/plagiarism-checker – Дата доступу: квітень 2025.
49. Grammarly Review. [Електронний ресурс] – Режим доступу: https://academichelp.net/plagiarism-checkers/grammarly-review.html – Дата доступу: квітень 2025.
50. Grammarly Plagiarism Checker Review. [Електронний ресурс] – Режим доступу: https://originality.ai/blog/grammarly-plagiarism-checker-review – Дата доступу: квітень 2025.
51. Grammarly and Plagiarism. [Електронний ресурс] – Режим доступу: https://www.reddit.com/r/AskAcademia/comments/18wddnj/grammaly_and_plagiarism/ – Дата доступу: квітень 2025.
52. Python Programming Language. [Електронний ресурс] – Режим доступу: https://www.python.org/ – Дата доступу: квітень 2025.
53. FastAPI. [Електронний ресурс] – Режим доступу: https://fastapi.tiangolo.com/ – Дата доступу: квітень 2025.
54. PostgreSQL. [Електронний ресурс] – Режим доступу: https://www.postgresql.org/ – Дата доступу: квітень 2025.
55. React: The library for web and native user interfaces. [Електронний ресурс] – Режим доступу: https://react.dev/ – Дата доступу: квітень 2025.
56. Ant Design: The world's second most popular React UI framework. [Електронний ресурс] – Режим доступу: https://ant.design/ – Дата доступу: квітень 2025.
57. Quora Question Pairs [Електронний ресурс] – Режим доступу: https://www.kaggle.com/competitions/quora-question-pairs – Дата доступу: квітень 2025.
58. Microsoft Research Paraphrase Corpus Pairs [Електронний ресурс] – Режим доступу: https://www.kaggle.com/datasets/doctri/microsoft-research-paraphrase-corpus – Дата доступу: квітень 2025.

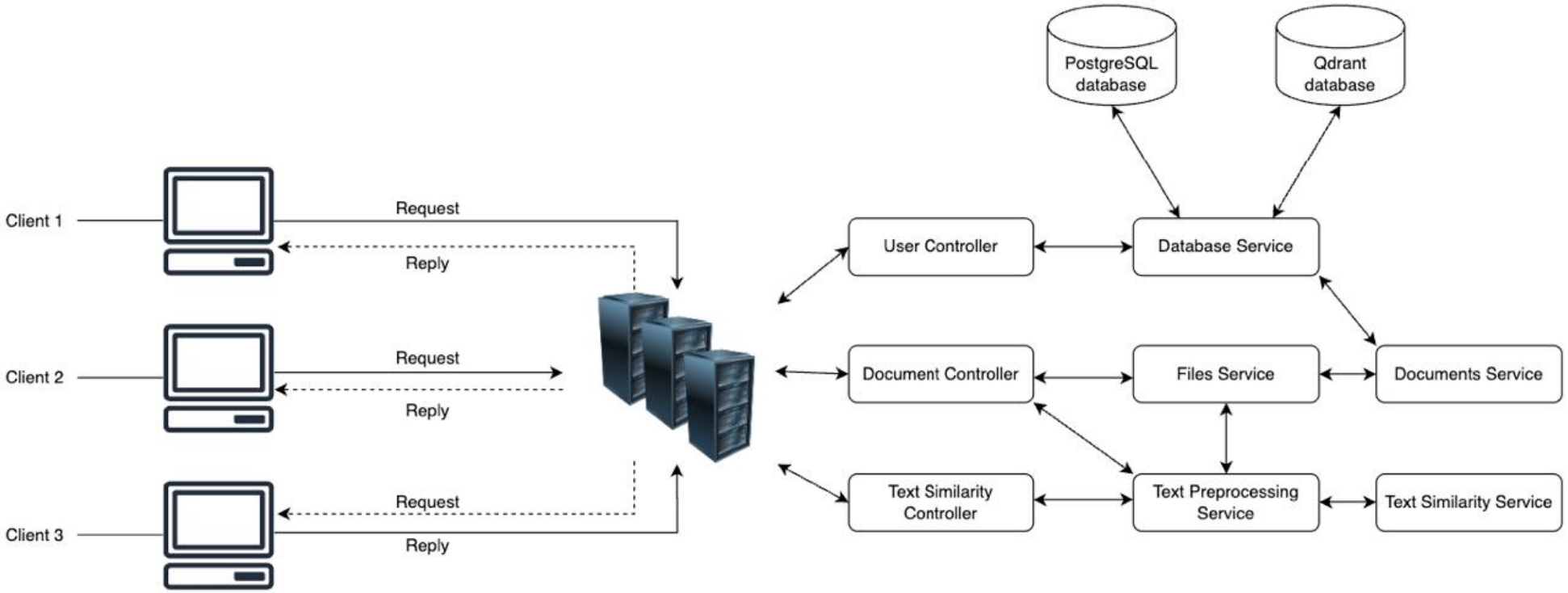
ДОДАТКИ

Додаток 1

Копії графічних матеріалів

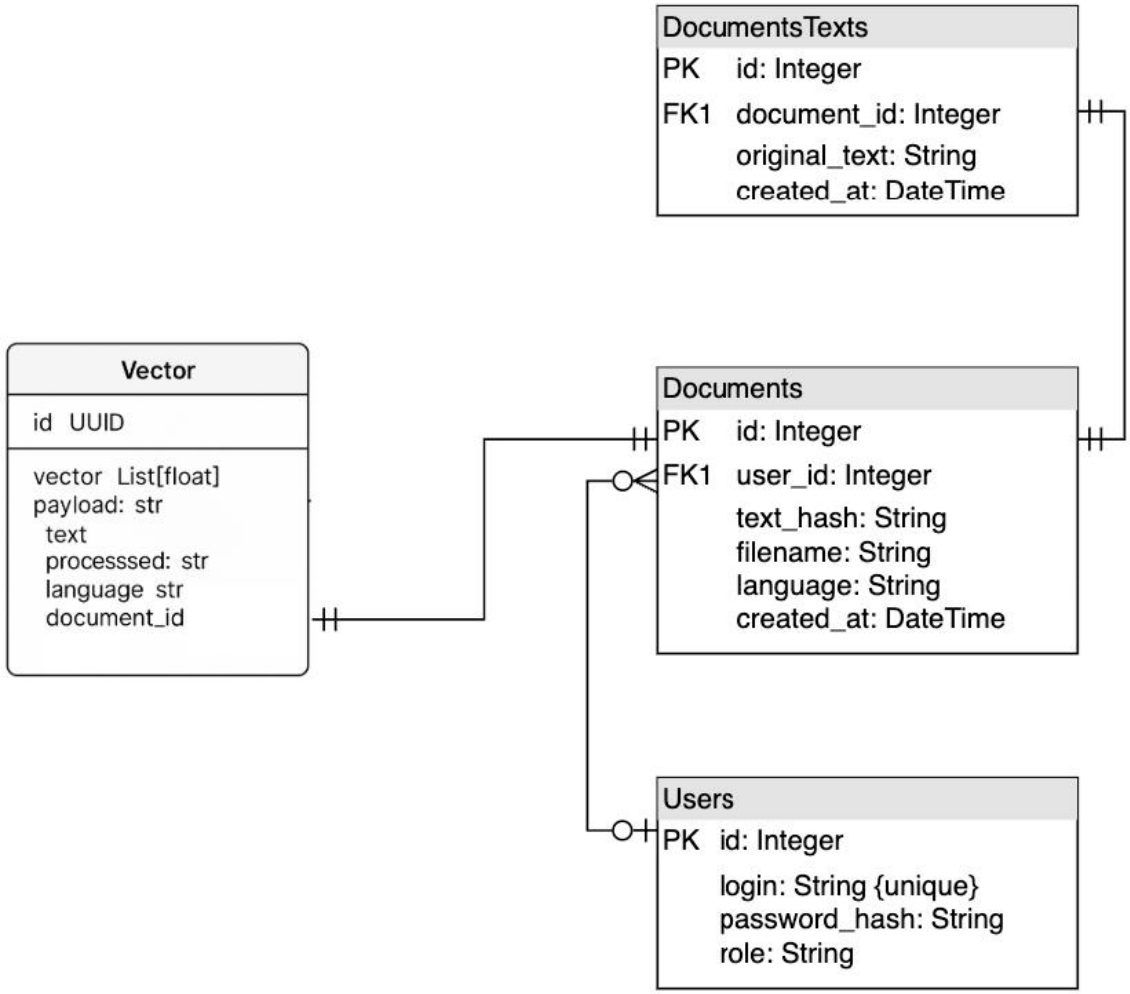


Графічне представлення алгоритму роботи модифікованого методу. Блок-схема
Козинець Н. В., група КП-31-мн

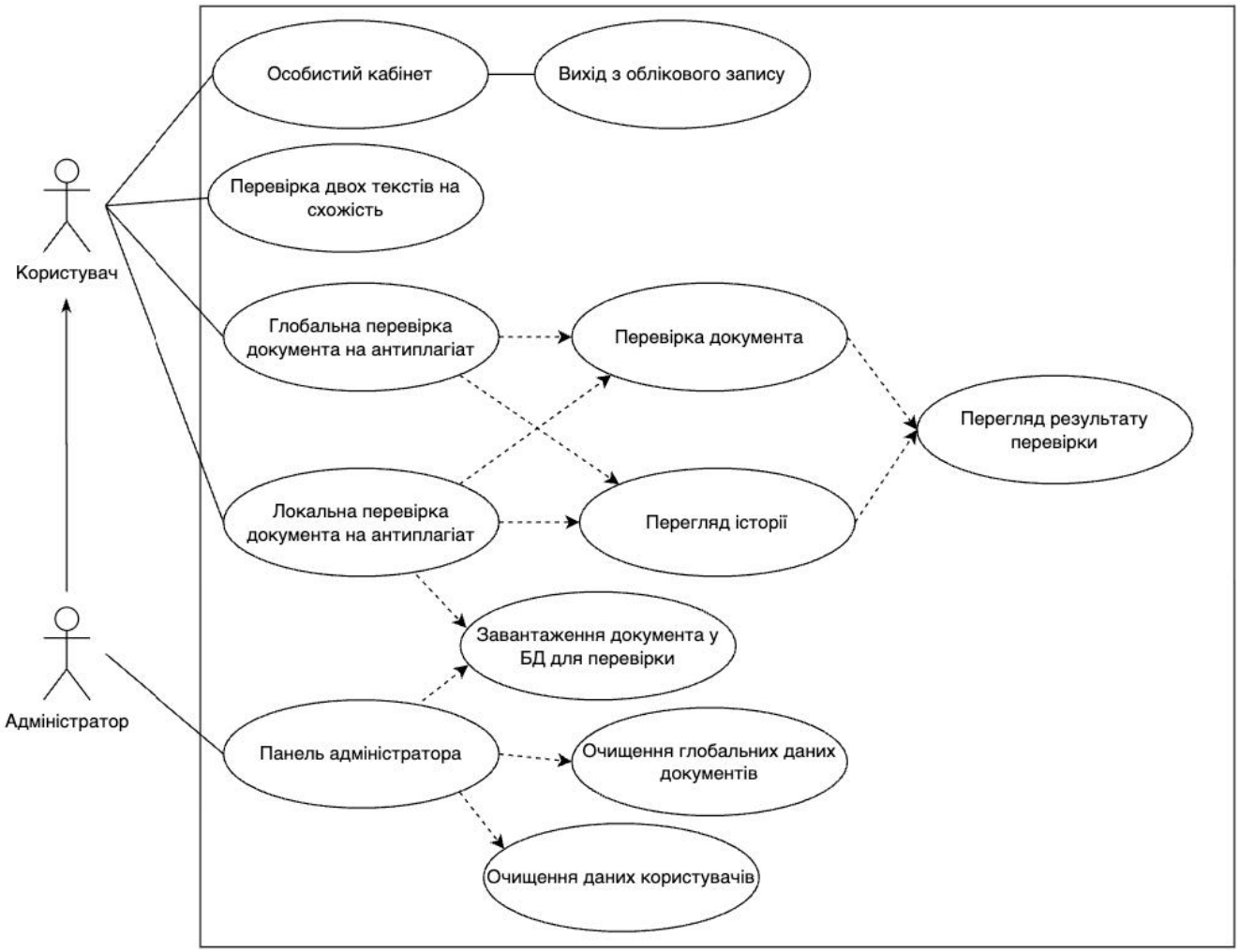


Архітектура розробленого програмного забезпечення модифікованого методу косинусної подібності для виявлення нечітких дублікатів у текстових даних. Плакат

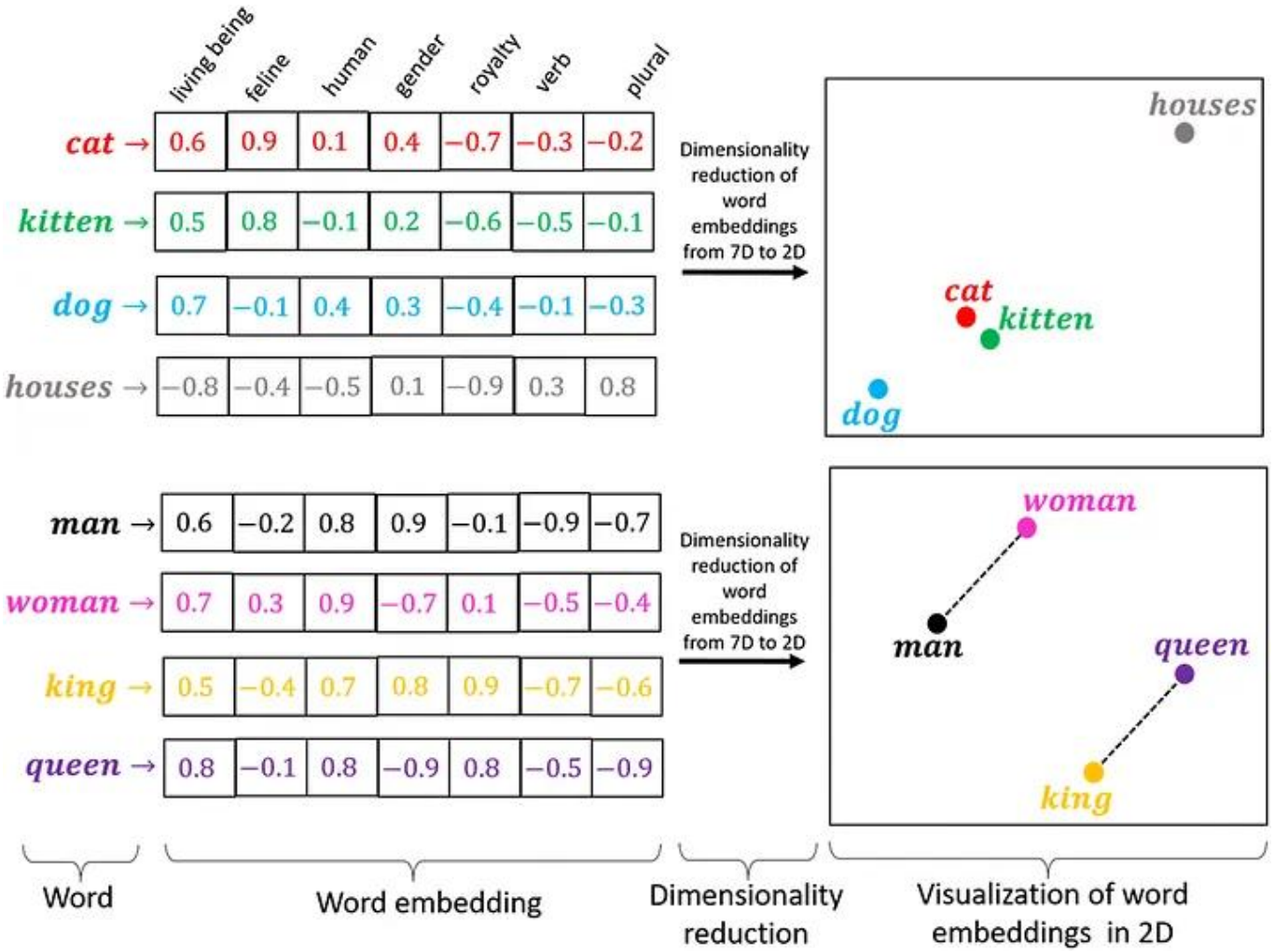
Козинець Н. В., група КП-31-мн



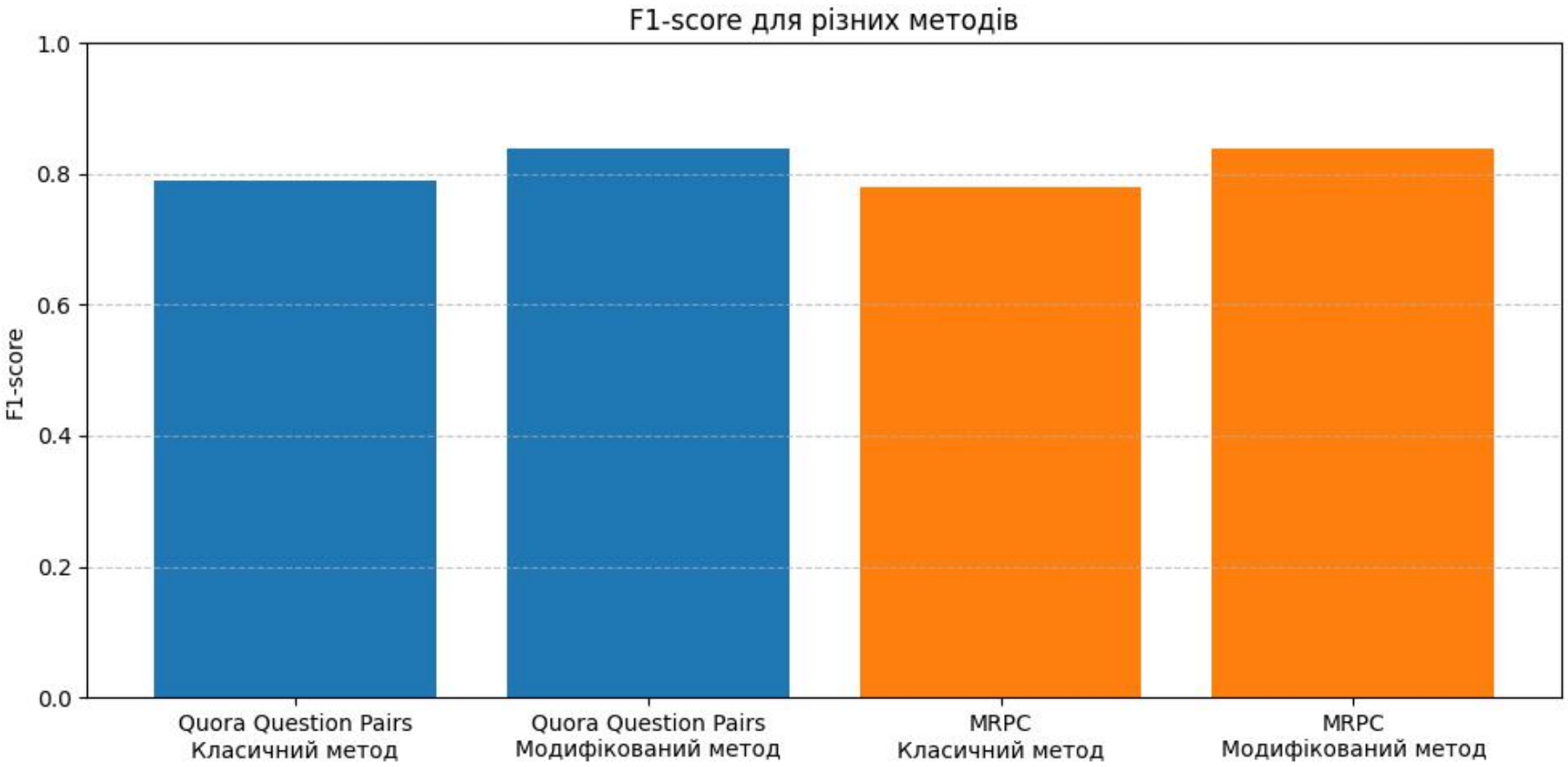
Структура баз даних модифікованого методу косинусної подібності для виявлення нечітких дублікатів у текстових даних. ER-діаграма
Козинець Н. В., група КП-31-мн



UML діаграма сценаріїв використання розробленого програмного забезпечення для виявлення нечітких дублікатів в тексті користувачами
Козинець Н. В., група КП-31-мн



Візуальний процес векторного представлення текстових даних. Плакат
Козинець Н. В., група КП-31-мн



Візуальне представлення роботи програмного забезпечення класичного та модифікованого методу
Козинець Н. В., група КП-31-мн

Додаток 2

Лістинг програми

Лістинг 1. Сервіс обробки тексту і перетворення у вектори

```
import string
import re
import nltk
import spacy # type: ignore
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from langdetect import detect
from sentence_transformers import SentenceTransformer
from typing import Dict, List, Optional
from app.resources.stop_word_ua import stop_words_ua
import json
from pathlib import Path
import numpy as np

ALPHA = np.asarray(json.loads(Path("alpha_weights.json").read_text()))

# Завантаження української моделі для лематизації
uk_nlp = spacy.load('uk_core_news_lg')

class SBERTModelManager:
    _instance: Optional['SBERTModelManager'] = None
    _model: Optional[SentenceTransformer] = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super(SBERTModelManager, cls).__new__(cls)
        return cls._instance

    @classmethod
    def get_model(cls) -> SentenceTransformer:
        if cls._model is None:
            cls._model = SentenceTransformer('sentence-transformers/paraphrase-
multilingual-MiniLM-L12-v2')
        return cls._model

# Завантаження необхідних ресурсів NLTK
try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    nltk.download('stopwords')

try:
    nltk.data.find('tokenizers/punkt')
except LookupError:
    nltk.download('punkt')

class TextPreprocessingService:
    def __init__(self):
        # Доступні мови для SnowballStemmer
        self.supported_languages = {
            'ar': 'arabic',
            'da': 'danish',
            'nl': 'dutch',
            'en': 'english',
            'fi': 'finnish',
            'fr': 'french',
            'de': 'german',
            'hu': 'hungarian',
            'it': 'italian',
```

```
            'no': 'norwegian',
            'pt': 'portuguese',
            'ro': 'romanian',
            'es': 'spanish',
            'sv': 'swedish',
        }

        # Ініціалізація стеммерів для всіх підтримуваних мов
        self.stemmers = {
            lang_code: SnowballStemmer(lang_name)
            for lang_code, lang_name in self.supported_languages.items()
        }

        # Стоп-слова для різних мов
        self.stop_words = {
            'ar': set(stopwords.words('arabic')),       # Arabic
            'da': set(stopwords.words('danish')),       # Danish
            'nl': set(stopwords.words('dutch')),        # Dutch
            'en': set(stopwords.words('english')),      # English
            'fi': set(stopwords.words('finnish')),      # Finnish
            'fr': set(stopwords.words('french')),       # French
            'de': set(stopwords.words('german')),       # German
            'hu': set(stopwords.words('hungarian')),    # Hungarian
            'it': set(stopwords.words('italian')),      # Italian
            'no': set(stopwords.words('norwegian')),    # Norwegian
            'pt': set(stopwords.words('portuguese')),   # Portuguese
            'ro': set(stopwords.words('romanian')),     # Romanian
            'es': set(stopwords.words('spanish')),      # Spanish
            'sv': set(stopwords.words('swedish')),      # Swedish
            'uk': set(stop_words_ua),                   # Ukrainian
        }

    def detect_language(self, text: str) -> str:
        """
        Визначення мови тексту.
        Повертає код мови (en, de, fr, etc.).
        """
        try:
            lang = detect(text)
            if lang == 'uk':
                return 'uk'
            # Перетворення коду мови до нашого формату
            if lang in self.supported_languages:
                return lang
            return 'en'  # За замовчуванням англійська
        except:
            return 'en'  # У випадку помилки повертаємо англійську

    def apply_alpha(self, vec: np.ndarray) -> np.ndarray:
        """Hadamard-множення на α і повторна L2-нормалізація."""
        v = vec * ALPHA                     # зважування
        return v / np.linalg.norm(v)

    def __remove_punctuation(self, text: str) -> str:
        """
        Видалення знаків пунктуації з тексту.
        """
        return text.translate(str.maketrans('', '', string.punctuation))

    def __remove_stopwords(self, text: str, language: str) -> str:
        """
        Видалення стоп-слів з тексту.
        """
        words = word_tokenize(text)
```

```
        stop_words = self.stop_words.get(language, set()) if language is not
None else set()
        filtered_words = [word for word in words if word not in stop_words]
        return ' '.join(filtered_words)

    def __lemmatize(self, text: str, language: str) -> str:
        """
        Лематизація тексту.
        Для української мови використовується uk_core_news_lg,
        для інших мов - SnowballStemmer.
        """
        if not text:
            return ""

        if language == 'uk':
            # Використовуємо spacy для української мови
            doc = uk_nlp(text)
            lemmatized_words = [token.lemma_ for token in doc]
        elif language in self.stemmers:
            # Використовуємо SnowballStemmer для інших підтримуваних мов
            words = text.split()
            stemmer = self.stemmers[language]
            lemmatized_words = [stemmer.stem(word) for word in words]
        else:
            # Для непідтримуваних мов повертаємо оригінальні слова
            return text

        return ' '.join(lemmatized_words)

    def __split_text_to_sentences(self, text: str) -> List[str]:
        """
        Розділення тексту на речення.
        """
        sentences = sent_tokenize(text)
        return sentences

    def preprocess_text(self, text: str) -> List[Dict[str, str]]:
        """
        Основна функція обробки тексту.
        Повертає словник з оригінальним та обробленим текстом.
        """
        if not text:
            return {
                "language": "en",
                "original": "",
                "original_sentences": [],
                "sentences": [],
                "sentence_vectors": [],
            }

        # Розділення тексту на речення
        sentences = self.__split_text_to_sentences(text)
        final_sentences = []

        for sent in sentences:
            processed_sent = sent
            # Приведення до нижнього регістру
            processed_sent = sent.lower()

            # Видалення спецсимволів
            processed_sent = re.sub(r'\n+', ' ', processed_sent).strip()

            # Видалення багато пробілів
            processed_sent = re.sub(r'\s+', ' ', processed_sent).strip()
```

```
            # Видалення знаків пунктуації
            processed_sent = self.__remove_punctuation(processed_sent)

            # перевірка чи це цифри та пробіли лише
            if re.match(r'^[\d\s]*$', processed_sent):
                continue

            # Визначення мови
            language = self.detect_language(processed_sent)

            # Видалення стоп-слів
            # processed_sent = self.__remove_stopwords(processed_sent, language)

            # Лематизація
            # processed_sent = self.__lemmatize(processed_sent, language)

            # Видалення багато пробілів
            processed_sent = re.sub(r'\s+', ' ', processed_sent).strip()

            if not processed_sent.strip():
                continue

            # Обчислення векторів з використанням кешованої моделі
            sentence_vector =
SBERTModelManager.get_model().encode(processed_sent).tolist()
            # sentence_vector = self.apply_alpha(sentence_vector).tolist()

            final_sentences.append({
                "original": sent,
                "processed": processed_sent,
                "language": language,
                "sentence_vector": sentence_vector,
            })


        return final_sentences
```

Лістинг 2. Сервіс пошуку подібності

```
import numpy as np
from fastapi import HTTPException
import uuid
from typing import Dict
import re
from app.services.text_preprocessing_service import TextPreprocessingService
from qdrant_client.models import Distance, PointStruct, Filter, FieldCondition,
MatchValue
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from app.services.qdrant_service import QdrantService
import traceback

class TextSimilarityService:
    def __init__(self):
        self.preprocessing_service = TextPreprocessingService()
        self.qdrant_service = QdrantService()

    def __cosine_similarity(self, vec1: np.ndarray, vec2: np.ndarray) -> float:
        """Calculate cosine similarity between two vectors"""
        return np.dot(vec1, vec2) / (np.linalg.norm(vec1) *
np.linalg.norm(vec2))
```

```
    def calculate_base_cosine_similarity(self, text1: str, text2: str) -> float:
        """Calculate base cosine similarity between two texts"""
        X_list = word_tokenize(text1)
        Y_list = word_tokenize(text2)

        # sw contains the list of stopwords
        sw = stopwords.words('english')
        l1 =[];l2 =[]

        # remove stop words from the string
        X_set = {w for w in X_list if not w in sw}
        Y_set = {w for w in Y_list if not w in sw}

        # form a set containing keywords of both strings
        rvector = X_set.union(Y_set)
        for w in rvector:
            if w in X_set: l1.append(1) # create a vector
            else: l1.append(0)
            if w in Y_set: l2.append(1)
            else: l2.append(0)
        c = 0

        # cosine formula
        for i in range(len(rvector)):
                c+= l1[i]*l2[i]
        cosine = c / float((sum(l1)*sum(l2))**0.5)

        return float(cosine)

    def __calculate_metrics(self, text1: str, text2: str, threshold: float =
0.7) -> Dict[str, float]:
        """Calculate precision, recall, and F1-score for text similarity"""
        # Preprocess texts
        processed1 = self.preprocessing_service.preprocess_text(text1)
        processed2 = self.preprocessing_service.preprocess_text(text2)

        collection_name = str(uuid.uuid4())

        self.qdrant_service.delete_collection(collection_name=collection_name)
        self.qdrant_service.create_collection(collection_name=collection_name,
vector_size=len(processed1[0]["sentence_vector"]), distance=Distance.COSINE)

        self.qdrant_service.upsert(
            collection_name=collection_name,
            points=[
                PointStruct(id=str(uuid.uuid4()), vector=pr["sentence_vector"],
payload={"text": pr["original"], "processed": pr["processed"], "language":
pr["language"]})
                for pr in processed1 if (len(pr["processed"]) > 1) and (not
re.match(r'^\d+$', pr["processed"]))
            ],
        )

        results = []

        for pr in processed2:
            if (len(pr["processed"]) > 1) and (not re.match(r'^\d+$',
pr["processed"])):
                _result = self.qdrant_service.query_points(
                    collection_name=collection_name,
                    query=pr["sentence_vector"],
                    query_filter=Filter(must=[
                        FieldCondition(key="language", match=MatchValue(
                            value=pr["language"],
```

```
                        )),
                    ]),
                    score_threshold=threshold, # 0.7 if len(text1) > 50000 else
0.5,
                    with_payload=True,
                    with_vectors=False,
                    limit=1
                )

                results.append([_result, pr["original"]])

        avg_score = sum([result.points[0].score if result and len(result.points)
> 0 else 0 for [result, _] in results]) / len(results)

        common_phrases = []
        for [result, original] in results:
            if result and len(result.points) > 0:
                base_similarity =
self.calculate_base_cosine_similarity(result.points[0].payload["text"],
original)
                common_phrases.append({
                    "from": result.points[0].payload["text"],
                    "to": original,
                    "similarity": result.points[0].score,
                    "base_similarity": base_similarity
                })

        common_phrases = [phrase for phrase in common_phrases if phrase is not
None]

        self.qdrant_service.delete_collection(collection_name=collection_name)

        # get precision, recall, f1_score
        # Calculate metrics based on Qdrant results
        true_positives = len([r for [r, _] in results if r and len(r.points) > 0
and r.points[0].score >= threshold])
        false_positives = len([r for [r, _] in results if r and len(r.points) >
0 and r.points[0].score < threshold])
        false_negatives = len(processed2) - true_positives

        precision = true_positives / (true_positives + false_positives) if
(true_positives + false_positives) > 0 else 0
        recall = true_positives / (true_positives + false_negatives) if
(true_positives + false_negatives) > 0 else 0
        f1_score = 2 * (precision * recall) / (precision + recall) if (precision
+ recall) > 0 else 0

        return {
            "precision": float(precision),
            "recall": float(recall),
            "f1_score": float(f1_score),
            "cosine_similarity": avg_score,
            "common_phrases": common_phrases
        }

    def compare_texts(self, text1: str, text2: str) -> Dict[str, float]:
        """Compare two texts and return similarity metrics"""
        try:
            base_cosine_similarity =
self.calculate_base_cosine_similarity(text1, text2)
            print('🚀 ~ base_cosine_similarity', base_cosine_similarity)

            modified_metrics = self.__calculate_metrics(text1, text2)
            print(' ~ modified_metrics', modified_metrics["cosine_similarity"])
```

```
            return {
                "base_cosine_similarity": base_cosine_similarity,
                "modified_metrics": modified_metrics
            }
        except Exception as e:
            print(' ~ e', traceback.format_exc())
            raise HTTPException(status_code=500, detail=str(e))
```

Лістинг 3. Сервіс обробки файлів

```
from fastapi import HTTPException, UploadFile
import docx
import PyPDF2
import io
from app.services.text_preprocessing_service import TextPreprocessingService
from typing import Dict, List, Optional
from app.services.text_service import TextService

class FileService:
    def __init__(self):
        self.preprocessing_service = TextPreprocessingService()
        self.text_service = TextService()

    @staticmethod
    def extract_text_from_docx(file_content: bytes) -> str:
        """Extract text from DOCX file."""
        doc = docx.Document(io.BytesIO(file_content))
        return "\n".join([paragraph.text for paragraph in doc.paragraphs])

    @staticmethod
    def extract_text_from_pdf(file_content: bytes) -> str:
        """Extract text from PDF file."""
        pdf_reader = PyPDF2.PdfReader(io.BytesIO(file_content))
        text = ""
        for page in pdf_reader.pages:
            text += page.extract_text()
        return text

    @staticmethod
    def extract_text_from_txt(file_content: bytes) -> str:
        """Extract text from TXT file."""
        return file_content.decode('utf-8')

    async def get_file_content(self, file: UploadFile) -> bytes:
        """Get file content."""
        if not file:
            raise HTTPException(
                status_code=400,
                detail="No file provided"
            )

        try:
            content = await file.read()
            file_extension = file.filename.split('.')[-1].lower()

            if file_extension == 'docx':
                text = self.extract_text_from_docx(content)
            elif file_extension == 'pdf':
                text = self.extract_text_from_pdf(content)
            elif file_extension == 'txt':
```

```
                text = self.extract_text_from_txt(content)
            else:
                raise HTTPException(
                    status_code=400,
                    detail="Unsupported file format. Supported formats: docx,
pdf, txt"
                )

            return text

        except Exception as e:
            raise HTTPException(
                status_code=500,
                detail=f"Error processing file: {str(e)}"
            )

    async def process_file(self, db, file: UploadFile, user_id: Optional[int] =
None) -> Dict[str, float | List[Dict[str, str]]]:
        """
        Process uploaded file and extract text with preprocessing.
        Returns tuple of (filename, processed_text_dict)
        """
        if not file:
            raise HTTPException(
                status_code=400,
                detail="No file provided"
            )

        try:
            content = await file.read()
            file_extension = file.filename.split('.')[-1].lower()

            if file_extension == 'docx':
                text = self.extract_text_from_docx(content)
            elif file_extension == 'pdf':
                text = self.extract_text_from_pdf(content)
            elif file_extension == 'txt':
                text = self.extract_text_from_txt(content)
            else:
                raise HTTPException(
                    status_code=400,
                    detail="Unsupported file format. Supported formats: docx,
pdf, txt"
                )

            # Обробка витягнутого тексту
            processed_text = self.text_service.find_plagiarism(db, text,
user_id)

            return processed_text

        except Exception as e:
            raise HTTPException(
                status_code=500,
                detail=f"Error processing file: {str(e)}"
            )
```

Додаток 3

Копія презентації

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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

МОДИФІКОВАНИЙ МЕТОД КОСИНУСНОЇ ПОДІБНОСТІ ДЛЯ ВИЯВЛЕННЯ НЕЧІТКИХ ДУБЛІКАТІВ У ТЕКСТОВИХ ДАНИХ

Доповідач: Козинець Назарій Вікторович

Науковий керівник: доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Тетяна Миколаївна

Київ – 2025

АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ

- Існуючі методи, включно з традиційною косинусною мірою, демонструють недостатнє врахування семантичної близькості текстів при різній структурі речень та високу чутливість до синонімічних замін.
- Інтеграція вдосконалених методів виявлення нечітких дублікатів у сучасні технології дозволить зменшити інформаційний шум та оптимізувати використання обчислювальних ресурсів при роботі з великими масивами даних.

ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Об’єкт дослідження: є процес автоматизованого виявлення нечітких дублікатів в текстових даних.

Предмет дослідження: є методи, алгоритми та програмні засоби автоматизованого виявлення нечітких дублікатів в текстових даних.

ПОСТАНОВКА ЗАДАЧІ

Мета дослідження: підвищення точності у виявлені нечітких дублікатів у текстових даних шляхом розроблення та реалізації модифікованого методу косинусної подібності, що поєднує переваги сучасних векторних представлень із контекстно-залежним зважуванням компонент.

1. Дослідити окремі існуючі алгоритми та методи виявлення нечітких дублікатів, та визначити їх переваги та недоліки.
2. Сформулювати та перевірити гіпотезу про покращення методу.
3. Розробити модифікований метод косинусної подібності для покращення виявлення нечітких дублікатів.
4. Обґрунтувати вибір засобів для програмної реалізації.
5. Створити програмну реалізацію запропонованого методу.
6. Проаналізувати отримані результати та зробити висновки.

ІСНУЮЧІ МЕТОДИ ТА ТЕХНОЛОГІЇ

TF-IDF

I-Match

MinHash

Levenshtein Distance

Jaro-Winkler Distance

МЕТРИКИ ОЦІНКИ РОБОТИ МЕТОДІВ

Влучність (precision) — частка правильно визначених позитивних елементів серед усіх елементів, які модель класифікувала як позитивні.

$$Precision = \frac{TP}{TP+FP}$$

де TP — істинно позитивні, FP — хибно позитивні результати.

Повнота (recall) — частка правильно визначених позитивних елементів серед усіх фактично позитивних елементів у наборі даних.

$$Recall = \frac{TP}{TP+FN}$$

де TP — істинно позитивні, FN — хибно негативні результати.

МЕТРИКИ ОЦІНКИ РОБОТИ МЕТОДІВ

		Експертна оцінка	
		Позитивна	Негативна
Оцінка алгоритму	Позитивна	TP	FP
	Негативна	FN	TN

F-міра – гармонійне середнє між точністю (precision) та повнотою (recall), що збалансовано оцінює якість алгоритму в межах від 0 до 1.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

НАЙБІЛЬШ ВІДОМІ РІШЕННЯ НАУКОВОЇ ПРОБЛЕМИ

Метод	Точність	Влучність	Повнота	F1-міра
Term Frequency	**55 %**	65 %	40 %	46 %
TF × IDF	**60 %**	68 %	45 %	51 %
Cosine Similarity	**65 %**	72 %	55 %	60 %
Jaccard Similarity	**70 %**	70 %	35 %	47 %
MinHash	**78 %**	76 %	60 %	68 %
Levenshtein Distance	**85 %**	74 %	50 %	63 %
Jaro–Winkler Distance	**88 %**	73 %	45 %	60 %
I-Match	**95 %**	79 %	30 %	45 %

ГІПОТЕЗА

Модифікація мотоду косинусної подібності з використанням векторних представлень слів та контекстуального аналізу може значно покращити точність та швидкість виявлення нечітких дублікатів у великих текстових даних порівняно з традиційними методами.

МОДИФІКОВАНИЙ МЕТОД

Пропонується наступна модифікація на базі методу косинусної подібності.

Принцип роботи модифікованого методу:

1. Вхідний документ сегментується на окремі речення, які піддаються нормалізації (лематизація, видалення стоп-слів, усунення шумових елементів), з метою підвищення однорідності даних для подальшої векторизації.
2. Для кожного нормалізованого речення генерується векторне представлення за допомогою моделі Sentence-BERT, яка дозволяє отримати контекстно-залежну ознакову репрезентацію тексту.

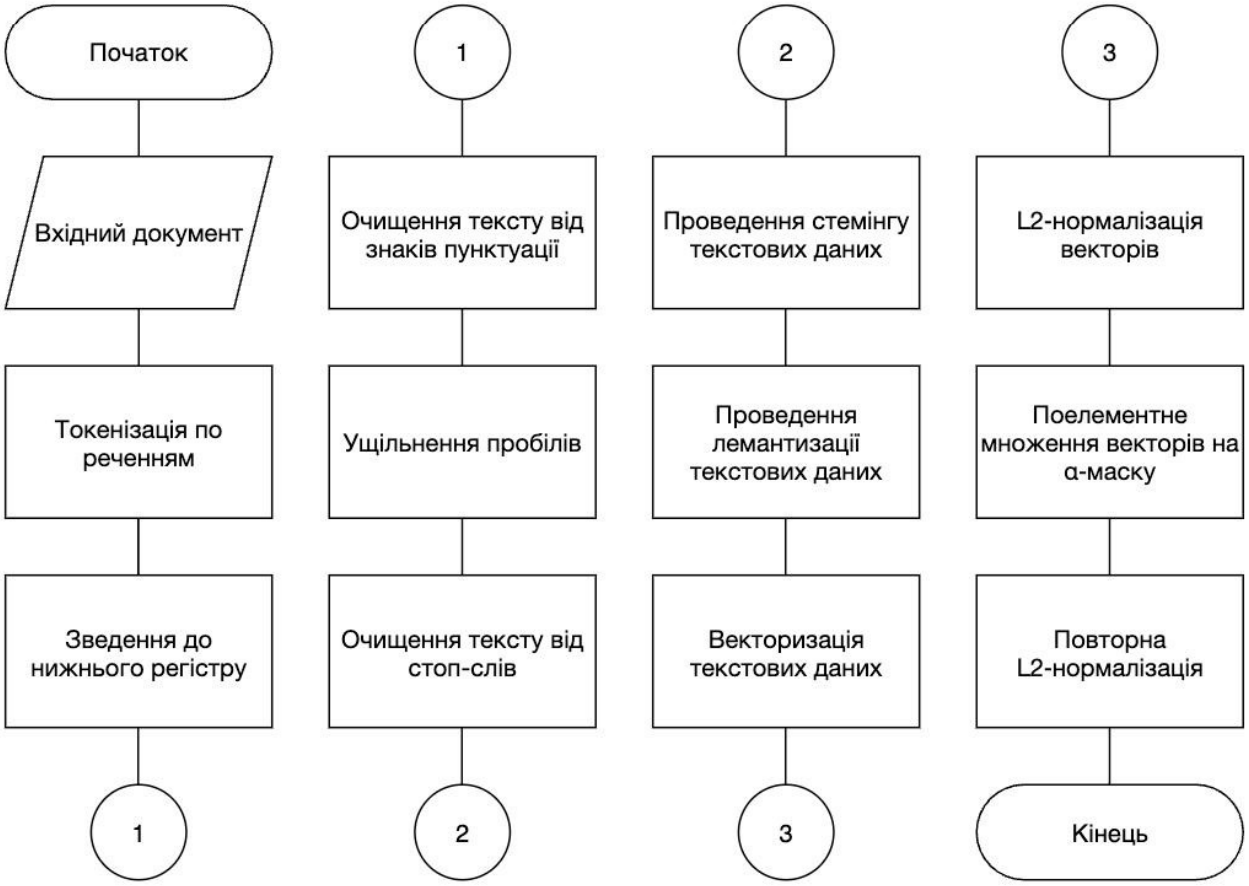
МОДИФІКОВАНИЙ МЕТОД

3. Координати векторів масштабуються із урахуванням оберненої дисперсії ознак у навчальному корпусі, що забезпечує придушення неінформативних компонентів і підсилення значущих семантичних відмінностей.
4. Вектори підлягають повторній нормалізації за L2-нормою, що гарантує коректність подальших обчислень косинусної подібності та уніфікованість простору ознак.

МОДИФІКОВАНИЙ МЕТОД

5. Отримані векторні представлення індексуються та зберігаються у векторній базі даних Qdrant, що використовує алгоритм HNSW для ефективного пошуку найближчих сусідів.
6. Новий текст обробляється за аналогічною схемою. Для кожного його векторного представлення здійснюється пошук найближчих сусідів за метрикою косинусної подібності. У разі перевищення встановленого порогового значення подібності τ документ ідентифікується як нечіткий дублікат.

СХЕМА РОБОТИ МОДИФІКОВАНОГО МЕТОДУ



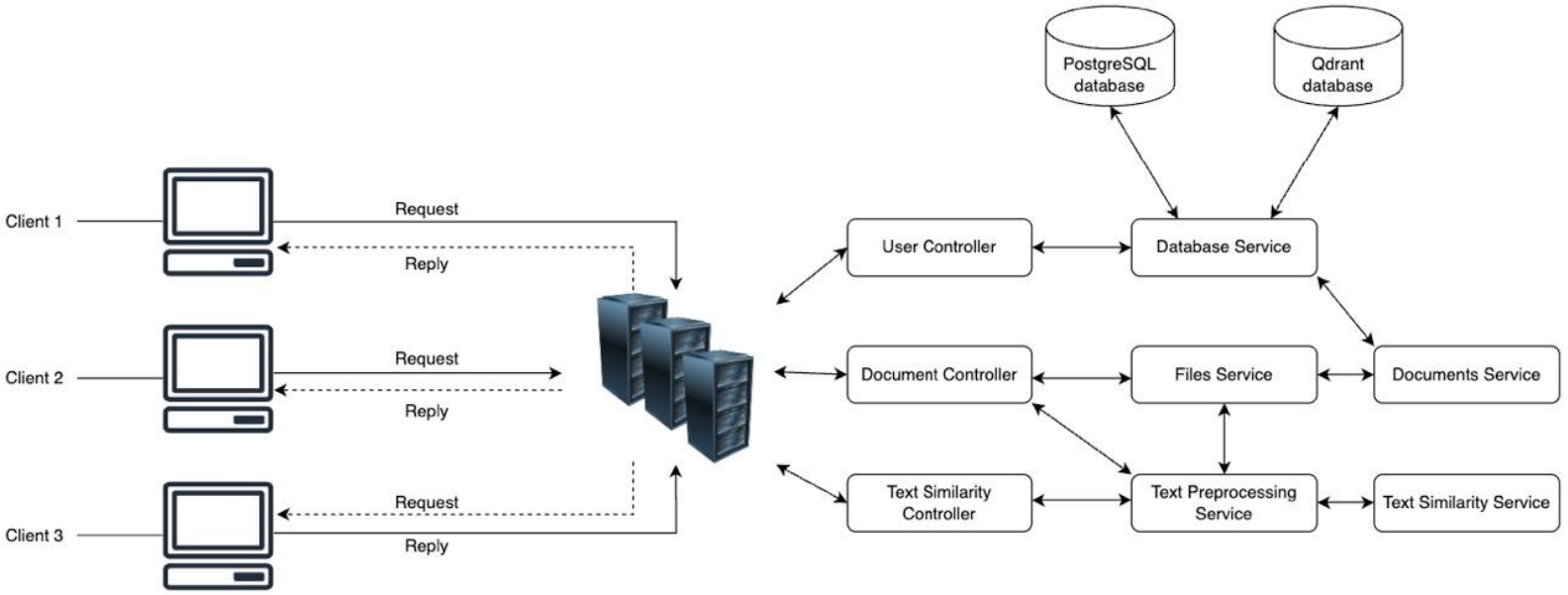
ОБРАНІ ЗАСОБИ РЕАЛІЗАЦІЇ

FastAPI

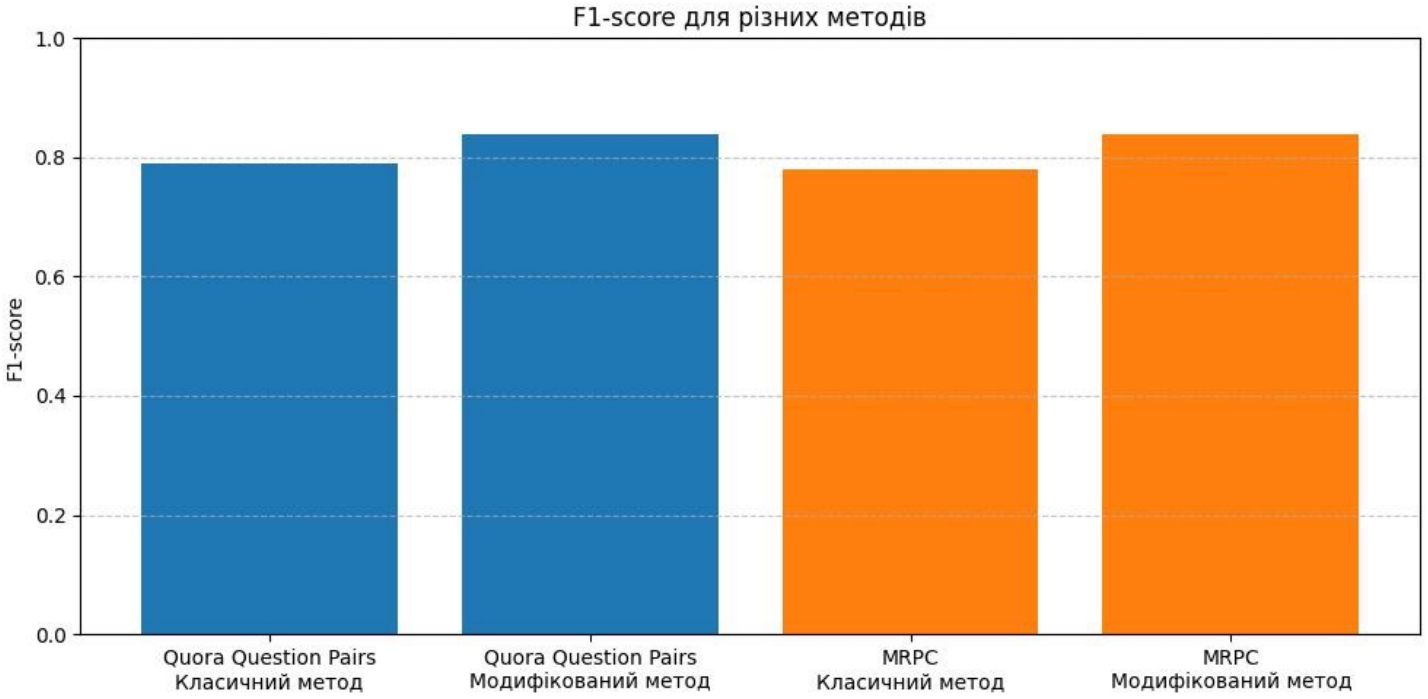
spaCy

NLTK

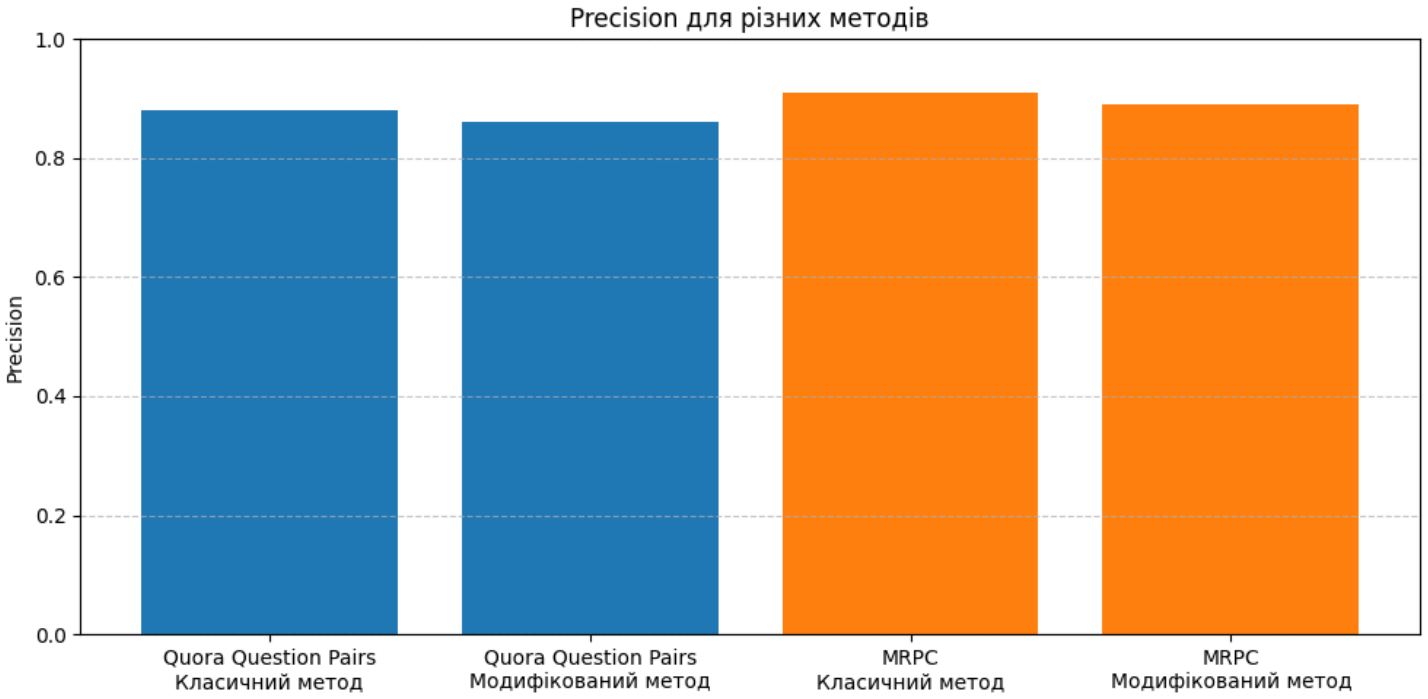
АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



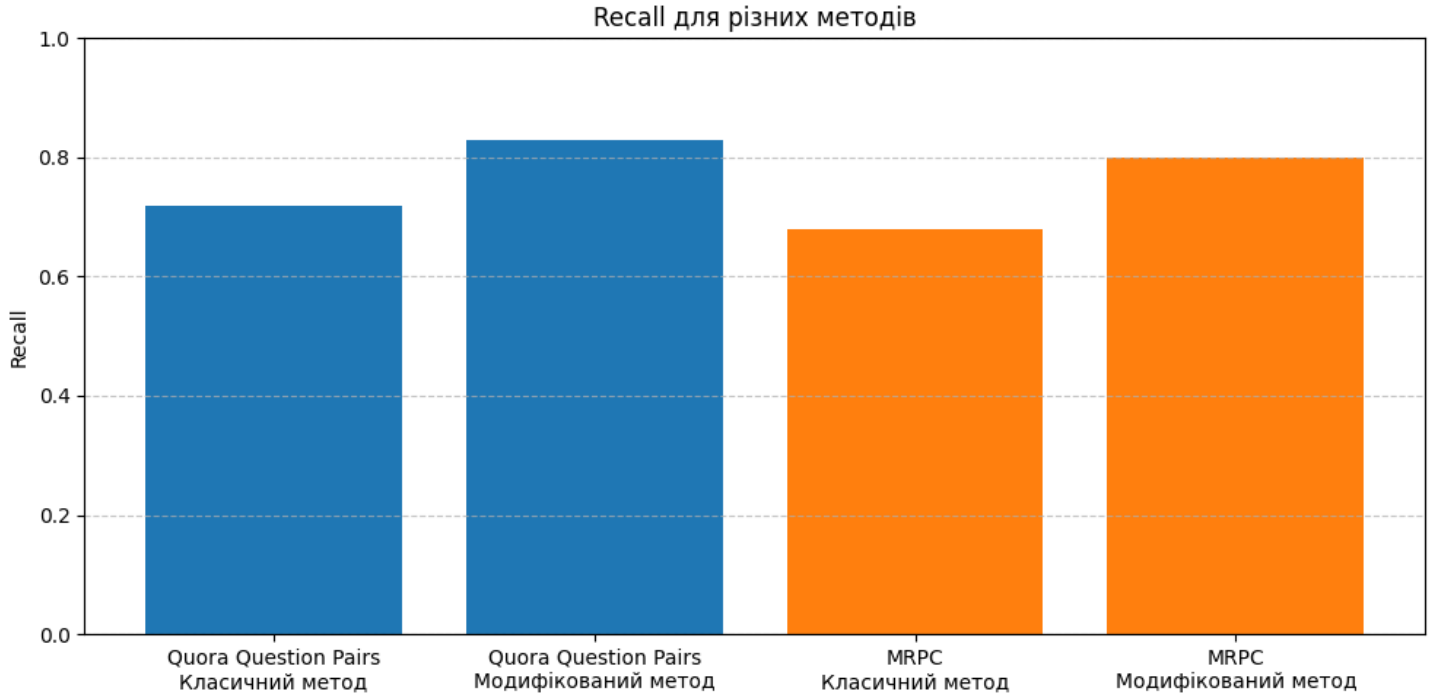
АНАЛІЗ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



АНАЛІЗ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



АНАЛІЗ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



НАУКОВА НОВИЗНА

Удосконалено метод косинусної подібності для виявлення нечітких дублікатів у текстових даних, характерною особливістю якого є поєднання контекстуального зважування координат векторного представлення речень на основі їхньою інформаційної значущості та статистично визначених α-ваг для підвищення чутливості до семантичних перефразувань, що дозволяє підвищити метрики F1 на 5-6% у порівнянні з класичним косинусним методом та на 10-14% у порівнянні з традиційними частотними методами.

ПРАКТИЧНА ЗНАЧУЩІСТЬ

Розроблено ПЗ на основі модифікованого методу косинусної подібності для виявлення нечітких дублікатів тексту з використанням Python, NLTK та SentenceTransformer.

Тестування продемонструвало високу ефективність системи у виявленні семантично еквівалентних текстів. Рішення може інтегруватися в системи перевірки оригінальності, пошукові системи, електронні бібліотеки та корпоративні системи керування документами для підвищення точності виявлення дублікатів.

АПРОБАЦІЯ

1. Основні положення і результати даної роботи були представлені на XVII науковій конференції магістрантів та аспірантів «Прикладна математика та комп’ютинг» ПМК-2024 (Київ, 20-22 листопада 2024 р.).
2. Стаття "Hybrid detection of fuzzy duplicate texts: cosine similarity and transformers" в фаховому журналі «Прикладні аспекти інформаційних технологій Том 8 № 1 (2025)».
3. Стаття «Вплив комбінованих векторних представлень на точність пошуку нечітких дублікатів» в фаховому журналі «Наукові праці ВНТУ №1 (2025)».

УНІКАЛЬНІСТЬ РОБОТИ

За результатами перевірки пояснювальної записки на наявність ознак плагіату встановлено рівень унікальності 99,75%

НАПРЯМКИ ПОДАЛЬШОЇ РОБОТИ

1. Адаптивний поріг τ для оптимізації балансу Precision/Recall на текстах різної довжини
2. Динамічне оновлення α-ваг через batch-оновлення або інкрементні підходи
3. Розширення підтримки мов через локалізовані моделі або cross-lingual mapping
4. Дослідження модифікованого методу за іншими гібридними метриками
5. Оптимізація продуктивності через GPU-прискорення та удосконалення індексування
6. Інтеграція із зовнішніми системами через REST-API, webhooks та CMS/LMS

ВИСНОВКИ

1. Проаналізовано існуючі методи виявлення нечітких дублікатів текстових даних.
2. Сформульовано гіпотезу щодо покращення методу косинусної подібності за допомогою контекстних векторних моделей.
3. Розроблено модифікований метод, що використовує Sentence-BERT, контекстні α-ваги та нормалізацію векторів.
4. Створено вебзастосунок з архітектурою на основі Python та React для реалізації методу.
5. Експериментально підтверджено підвищення ключових метрик порівняно з базовими підходами.
6. Визначенно напрямки подальших досліджень у сфері виявлення семантично схожих текстів.

Дякую за увагу!