Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Катедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів автоматизації
формування текстових корпусів
(тема)

Виконав:
здобувач 2 року навчання
групи ІПЗм-23-3

Данило ГОРЄЛОВ
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Олександр ВЕЧУР
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. катедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Катедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. катедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Горєлову Данилу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження методів автоматизації формування текстових корпусів»

Затверджена наказом по університету від 15.04. 2025р. № 290 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 19.06.2025

3. Вихідні дані до роботи опис досліджуваних моделей, методів та алгоритмів, мова програмування Python, бібліотеки lxml, bs4, spacy, re, середовище розробки DataSpell, новинний портал «Суспільне Новини»

4. Перелік питань, що потрібно опрацювати в роботі

аналіз та порівняння наявних текстових корпусів і методів їх формування, дослідження методів вилучення тексту, дослідження нормалізації (лапок, тел. номерів та апострофів) та токенізації текстів новин, опис вимог та написання системи формування текстового корпусу з покращенням нормалізації та токенізації тексту новин, проведення експериментів та аналіз результатів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
	Отримання завдання	16.04.2025	*виконано*
	Аналіз предметної галузі і постановка задачі	17.04.2025-24.04.2025	*виконано*
	Аналіз літературних джерел	25.04.2025-09.05.2025	*виконано*
	Розробка практичної частини	12.05.2025-16.05.2025	*виконано*
	Експериментальна оцінка системи	16.05.2025-20.05.2025	*виконано*
	Підготовка до апробації результатів дослідження.	21.05.2025-01.06.2025	*виконано*
	Підготовка пояснювальної записки	21.05.2025-03.06.2025	*виконано*
	Підготовка презентації та доповіді	04.06.2025-10.06.2025	*виконано*
	Перевірка на плагіат	11.06.2025	*виконано*
	Нормоконтроль	13.06.2025	*виконано*
	Рецензування	14.06.2025	*виконано*
	Попередній захист	15.06.2025	*виконано*
	Занесення диплома в електронний архів	15.06.2025	*виконано*
	Допуск до захисту у зав. кафедри	16.06.2025	*виконано*

Дата видачі завдання 16 квітня 2025р.

Студент (ка) _____ Данило ГОРЄЛОВ
(підпис)

Керівник роботи _____ доц. Олександр ВЕЧУР
(підпис) (посада, Власне ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 106 с., 49 рис., 4 табл., 57 джерел.

АНАЛІЗ ТЕКСТІВ, АПОСТРОФ, ЗБІР ДАНИХ, КОМП'ЮТЕРНА ЛІНГВІСТИКА, КОРПУС ТЕКСТІВ, ЛАПКИ, НОВИНИ, НОРМАЛІЗАЦІЯ, ОБРОБКА ТЕКСТІВ, ТЕЛЕФОННИЙ НОМЕР, ТОКЕНІЗАЦІЯ, УКРАЇНСЬКА МОВА.

Об’єктом дослідження є процеси формування, обробки та систематизації україномовних текстових даних для створення лінгвістичних корпусів.

Метою роботи є підвищення якості україномовних текстових корпусів та систем для автоматичного їх формування шляхом розробки вдосконаленої системи автоматичного формування текстових корпусів з покращеними методами нормалізації та токенізації текстів.

Методи дослідження охоплюють порівняльний аналіз наявних текстових корпусів, критичне вивчення методів нормалізації та токенізації, емпіричне дослідження проблем послідовності в текстових даних.

У результаті кваліфікаційної роботи було розроблено програмний інтерфейс для автоматизованого формування текстових корпусів з покращеними методами нормалізації та токенізації текстів українських новин.

Розроблена система забезпечує автоматизоване формування текстових корпусів з унікальними алгоритмами нормалізації (стандартизація телефонних номерів, апострофів, лапок) та семантичної токенізації, що підвищує якість обробки україномовних текстових даних.

TEXT ANALYSIS, APOSTROPHE, DATA COLLECTION, COMPUTATIONAL LINGUISTICS, TEXT CORPUS, QUOTATION MARKS, NEWS, NORMALIZATION, TEXT PROCESSING, PHONE NUMBER, TOKENIZATION, UKRAINIAN LANGUAGE.

The object of the research is the processes of formation, processing, and systematization of Ukrainian-language textual data for the creation of linguistic corpora.

The subject of the research is the methods and algorithms of normalization, tokenization, and automated formation of Ukrainian-language text corpora.

The purpose of the work is to improve the quality of Ukrainian-language text corpora and systems for their automatic generation by developing an advanced system for automatic generation of text corpora with improved methods of text normalization and tokenization.

The research methods include comparative analysis of existing text corpora, critical review of normalization and tokenization methods, and empirical study of sequence-related issues in textual data.

As a result of the qualification work, a software interface was developed for the automated formation of text corpora using improved methods of normalization and tokenization of Ukrainian news texts.

The developed system provides automated creation of text corpora with unique normalization algorithms (standardization of phone numbers, apostrophes, and quotation marks) and semantic tokenization, which improves the quality of Ukrainian-language text data processing.

Завідувачу кафедри
ПІ_____
(скорочена назва кафедри)
проф. Кирилу СМЕЛЯКОВУ
(вчене звання, сласне ім'я, прізвище)

ЗАЯВА
щодо самостійності виконання кваліфікаційної роботи та можливості її публікації (та/або публікації анотації кваліфікаційної роботи) в електронному архіві відкритого доступу ElAr KhNURE

Я, Горєлов Данило Олександрович, студент(ка) гр. ІПЗм-23-3, здобувач вищої освіти на другому (магістерському) рівні катедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів автоматизації формування текстових корпусів», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

Дата Підпис

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БрУК – браунський корпус української мови

САФТК – система автоматичного формування текстових корпусів

NLP – Natural language processing

UT2 – текстовий корпус Uber Text 2.0

XML – Extensible Markup Language

ВСТУП

Текстовий корпус – це структурована збірка текстових даних, що відображає справжнє використання мови та слугує фундаментом для навчання машинних моделей. У сучасному інформаційному просторі зростає потреба в системах автоматичного формування текстових корпусів (САФТК), що здійснюють отримання, очищення, виправлення, оцінку та збереження текстових даних.

Для української мови доступна обмежена кількість корпусів для вільного завантаження, найпомітнішими з яких є Браунський корпус української мови (БрУК) [1], UberText 1.0 [2] та UberText 2.0 (UT2) [3]. Корпус БрУК має високу якість, оскільки його тексти відібрані та оброблені вручну з мінімальною автоматизацією. Натомість UT2, створений повністю за допомогою САФТК, містить неоднорідність у форматуванні лапок, телефонних номерів та інших текстових елементів через недосконаліть нормалізації. Також токенізація в UT2 розриває логічно пов'язані токени на декілька (номери телефонів, посилання), що знижує якість корпусу та ускладнює подальшу обробку тексту. Розробка вдосконаленої САФТК з покращеними методами нормалізації та токенізації є актуальним завданням для підвищення якості створюваних україномовних текстових корпусів та моделей, що на них тренуються.

Робота пов’язана з науковими напрацюваннями катедри програмної інженерії у сфері обробки природної мови, що представлені у роботах Дубок А. Ю. [4] та Каук В. І. [5], які зокрема досліджують різноманітні інструменти для вилучення тексту з відео та аудіо файлів.

Метою роботи є підвищення якості україномовних текстових корпусів та систем для автоматичного їх формування шляхом розробки вдосконаленої САФТК з покращеними методами нормалізації та токенізації текстів. Для досягнення поставленої мети необхідно вирішити наступні завдання: проаналізувати наявні текстові корпуси української мови та методи їх формування; дослідити та виявити проблеми наявних методів нормалізації та токенізації україномовних текстів; розробити покращені методи нормалізації

текстів, зокрема для стандартизації телефонних номерів, апострофів та лапок; створити метод семантичної токенізації, що зберігає логічну цілісність складових, таких як номери телефонів, одиниці виміру тощо; реалізувати гнучку та адаптивну САФТК з використанням розроблених методів; провести апробацію та оцінку ефективності розробленої системи.

Об’єктом дослідження є процеси формування, обробки та систематизації україномовних текстових даних для створення лінгвістичних корпусів. Предметом дослідження є методи та алгоритми вилучення, нормалізації й токенізації тексту та автоматизованого формування україномовних текстових корпусів.

Методи дослідження охоплюють порівняльний аналіз наявних текстових корпусів, критичне вивчення методів нормалізації та токенізації, емпіричне дослідження проблем послідовності в текстових даних.

У результаті дослідження було одержано низку наукових положень, які відрізняються від відомих раніше підходів до нормалізації та токенізації українських текстів. Запропоновано уніфіковану нормалізацію телефонних номерів відповідно до вимог дизайн-системи державних вебсайтів України [6], удосконалено правила обробки лапок із дотриманням норм чинного правопису [7] та вкладеності цитат, а також нормалізацію апострофів з урахуванням їх можливої багатозначності. Розроблено покращений підхід до семантичної токенізації, що дозволяє зберігати цілісність складових, таких як номери телефонів чи числові вирази з одиницями виміру. Крім того, дістала подальшого розвитку система автоматизованого формування текстових корпусів, яку реалізовано у вигляді гнучкої та масштабованої архітектури.

Розроблена система має модульну архітектуру, що дозволяє інтегрувати чи підміняти різні компоненти, наприклад, джерела даних, без змін основної логіки. Поточна реалізація орієнтована на роботу з новинним сайтом «Суспільне Новини» [8], що є прикладом вебджерела для тестування системи. Також компоненти, наприклад, нормалізації чи токенізації можна використовувати

окремо. Код системи опубліковано в публічному репозиторії [9], що робить її доступною для використання в інших проєктах з обробки текстів.

Результати роботи пройшли апробацію на 1-й Міжнародній науково-практичній конференції «СУЧАСНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ MIT@AIS-2025» за темою «Text Normalization Challenges in Ukrainian News Corpus: Problem Analysis and Research Framework» у співавторстві з керівником кваліфікаційної роботи доц. Олександром Вечуром, що наведено у додатку Д пояснювальної записки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальний аналіз предметної області дослідження

Текстовий корпус – це структурована збірка текстових даних, що відображає справжнє використання мови в різних контекстах та стилях. У сучасних мовних технологіях корпуси текстів є основою для розвитку систем обробки природної мови. Для тренування сучасних нейромереж, таких як моделі автозаповнення тексту, автоматичного перекладу чи генерації контенту, необхідно мати доступ до великих обсягів якісних мовних даних. Саме корпуси текстів надають системам штучного інтелекту приклади того, як слова та фрази використовуються в певних контекстах, дозволяючи машині передбачати ймовірні наступні слова залежно від попередніх, зважаючи на синтаксис, семантику та стиль тексту.

Зазвичай для укладання текстових корпусів використовуються матеріали з новинних сайтів, оскільки вони доступні, регулярно оновлюються та як правило, мають високу якість. Для створення репрезентативного корпусу важливо робити його різноманітним, тому корпуси також можуть охоплювати тексти з книжок, соціальних мереж, блогів чи інших джерел. Однак, кожне з цих джерел має свої особливості та вимагає специфічних інструментів для ефективного вилучення текстової інформації.

Незалежно від джерела, текстові дані майже завжди містять різноманітні проблеми чи особливості, що ускладнюють їх безпосереднє використання для навчання мовних моделей. Серед основних викликів можна виділити: непослідовність форматування (використання різних типів лапок, апострофів, форматів запису телефонних номерів, дат, посилань тощо); мовні особливості регіонального характеру (діалекти, суржик); граматичні та орфографічні помилки. Для мовних моделей це створює суттєвий шум: різні написання одного й того ж поняття можуть трактуватися як різні об'єкти; граматичні та орфографічні помилки відтворюються в згенерованих текстах.

Наявні інструменти здатні автоматично обробляти лише окремі типи мовних варіативностей і помилок. Наприклад, чергування «у/в» можна вирішити

за допомогою простих правил або регулярних виразів. Водночас значна частина помилок, як-от вибір чергування «і/й», потребують врахування ширшого контексту та не завжди піддаються формалізації, що залишає їх поза зоною виявлення таких систем.

Для підготовки текстових даних до ефективного використання в мовних моделях застосовуються різноманітні методи попередньої обробки текстів, такі як: нормалізація, токенізація, стемінг, лематизація та видалення стоп-слів. Вони не змінюють зміст текстів, але перетворюють їх у формат, зручний для подальшої обробки. Наприклад, видалення стоп-слів зменшує шум у даних, токенізація сегментує текст на аналітичні одиниці, а лематизація зводить слова до базової форми, що спрощує виявлення лексичних закономірностей

Оброблені тексти в корпусі можуть зберігатися в різних форматах залежно від призначення корпусу та вимог до його використання, наприклад, простий текст з поділом на абзаци чи спеціалізовані формати для конкретних NLP-інструментів. Особливу роль відіграє структурна та лінгвістична розмітка тексту, що може містити інформацію про частини мови, синтаксичні зв'язки, семантичні категорії та інші лінгвістичні характеристики текстових одиниць. Також корпуси текстів пропонують токенізований текст, їх проблема полягає в недостатньому збереженні семантично цілісних конструкцій, що людина сприймає як єдині поняття, але більшість алгоритмів розділяє на окремі токени.

Значною проблемою у процесі формування текстових корпусів є дотримання авторських прав, що обмежує можливості збору та використання текстів з різних джерел. Необхідність отримання дозволів від правовласників часто ускладнює створення повноцінних відкритих корпусів, особливо для мов з обмеженими ресурсами, такими як українська.

Окрім застосування в технологіях, текстові корпуси є надзвичайно цінним інструментом для дослідників лінгвістики. Вони дозволяють аналізувати закономірності використання мови: частотність слів, еволюцію граматичних структур, вплив соціальних чи регіональних факторів на мовлення. Наприклад, вивчення текстових корпусів української мови до 1931 року розкриває глибокі

процеси зросійщення, які системно впроваджували російську лексику та граматичні конструкції в українське мовлення як інструмент культурної асиміляції [10].

1.2 Огляд наявних текстових корпусів

Ресурс English Corpora є одним із найпоширеніших та найширше використовуваних корпусів англійської мови, що пропонує розвинуті засоби аналізу мови [11]. Сайт надає доступ до кількох унікальних корпусів текстів із різних сфер та жанрів: новини, кіно, телепередачі, Вікіпедія. Також наявні специфічні збірки текстів щодо конкретної теми, наприклад, коронавірус, чи за джерелами, наприклад, журналу Time. Найбільше слів містить корпус новин – 20,3 мільярда слів. Найменший корпус містить понад 50 мільйонів слів з інтернету, однак це 11 перший великий вебкорпус, який ретельно класифікований за багатьма різними реєстрами: ліричний, виступи, інтерв'ю, поради тощо.

Безплатна версія ресурсу пропонує вебінтерфейс, що дозволяє виконувати пошук слів за різними параметрами, такими як словоформа, частина мови, частота вживання, значення, синоніми, загальніші або конкретніші слова, а також вимова. Для 60000 найпопулярніших слів англійської мови доступні розширені функції, зокрема визначення слова, частота його вживання за жанрами (наприклад, академічний чи розмовний текст), аналіз колокацій (близьких слів, що часто зустрічаються разом), пов'язані теми, які з'являються у текстах, кластери (фрази з 2, 3 або 4 слів, що включають це слово), та рядки узгодження з прикладами контексту. Додатково для цих слів надаються посилання на зовнішні ресурси, такі як статті зі словників, аудіозаписи вимови, зображення, відео та переклади на понад 100 мов.

Платна версія ресурсу дозволяє придбати повний текстовий корпус для завантаження та використовувати ці дані у будь-який спосіб, що відповідає розумним обмеженням. Дані доступні у трьох різних форматах: дані для реляційних баз даних, у вигляді списків слів із зазначенням їхньої леми та

частини мови, а також у форматі суцільного тексту (абзаци). Після покупки користувач отримує права на використання кожного з цих форматів.

Британський національний корпус (BNC) – це 100-мільйонна колекція зразків письмової та усної мови з широкого кола джерел, призначена для представлення широкого зрізу британської англійської мови, як розмовної, так і письмової, з кінця XX століття [12]. Корпус доступний для завантаження та всі файли описані в XML та дотримується формату TEI. Також доступна BNCweb, веб-клієнтська програма для пошуку та отримання лексичних, граматичних і текстових даних з Британського національного корпусу (BNC). Корпус охоплює близько 83% письмових текстів і 17% усних, що забезпечує репрезентативність мови у різних контекстах – від художньої літератури до технічної документації та побутових розмов. Переважна кількість текстів походить з періоду 1985-1993 років.

Для української мови доступно не так багато корпусів. Одним з них є Браунський корпус української мови (БрУк) [1]. Корпус містить мільйон слововживань збалансованих за категоріями: преса, релігійна література, професійно-популярна література, адміністративні документи, науково-популярна література, наукова література, художні тексти та «Естетичні інформативні» тексти. Кожен текст зберігається в окремому текстовому файлі. Браунський корпус мови відкритий та доступний для завантаження охочим. Корпус структурується в репозиторії по теках: перевірені фрагменти, написані літературною українською мовою; перевірені фрагменти, що містять помилки; перевірені фрагменти, що зовсім не відповідають вимогам (наприклад, усне мовлення); фрагменти, що чекають на перевірку.

Корпус Uber Text 1.0 (UT1) містить перемішаний текст з новин приблизно на 665 мільйонів токенів [2]. Корпус Uber Text 2.0 (UT2) є наступником UT1 та містить 2.5 мільярдів токенів. Корпус UT2 охоплює п’ять підкорпусів: новини, художні тексти, соціальні медіа (264 Telegram-канали), Вікіпедію (станом на січень 2023) та судові рішення Верховного Суду України. Для кожного підкорпусу доступні чотири рівні обробки: сирий текст, очищений, розбитий на

речення та токенізований. Токенізований текст розділений лише пробілами. Окрім того, на основі всього корпусу створено словник частот лем із позначками частин мови, що містить частоти лем і кількість документів, у яких вони зустрічаються [3].

Лінгвістичний портал Mova.info містить корпус української мови з понад 100 мільйонів слововживань [13]. Корпус містить лише вебінтерфейс для здійснення статистично лінгвістичних досліджень, використання під час укладання словників, граматик та для довідкового використання. Пряме завантаження текстів також відсутнє для «Генеральний регіонально анотований корпус української мови» (ГРАК) – це корпус української мови з вебінтерфейсом, що дозволяє будувати власні підкорпуси, шукати слова, граматичні форми та їх сполучення, а також одержувати різну статистичну інформацію [14]. Корпус охоплює період з 1816 по 2022 рік і містить понад 130 тисяч текстів різних жанрів, близько 30 тисяч авторів.

1.3 Особливості формування корпусу текстів українською мовою

Формування корпусу українських текстів пов’язане з низкою специфічних викликів, зумовлених історичними, соціальними та культурними особливостями розвитку української мови. Основна мета полягає в тому, щоб зібрати якісний матеріал, який відповідає сучасним мовним стандартам і дозволяє враховувати реальну мовну ситуацію. Серед ключових проблем – вплив інших мов, особливо російської, на мовну свідомість носіїв. Багато людей мислять російською, навіть пишучи українською, що породжує дослівні переклади, які не відповідають літературним нормам. Це явище, хоча й не завжди очевидне, суттєво ускладнює автоматичну обробку текстів, оскільки моделі, натреновані на літературних джерелах, можуть помилково інтерпретувати подібні конструкції.

Прямий вплив російської також проявляється у формуванні суржику – мовного феномена, що є сумішшю української та російської мов. Для багатьох носіїв суржик є звичним способом комунікації, хоча він віддаляється від літературної норми. Це створює виклики не лише для укладання корпусів, а й для

подальшого використання таких даних. Моделі, побудовані виключно на літературних текстах, часто не можуть адекватно працювати з розмовною мовою, яка відображає реальний стан речей, зокрема в соціальних мережах чи неформальному спілкуванні. Це свідчить про необхідність створення корпусів, що включали б як стандартизовану мову, так і живу, з її відхиленнями та особливостями.

Ще однією важливою рисою української мови є наявність діалектів. Українська мова має багатий діалектний спектр, який часто недооцінюється в дослідженнях і практичному використанні. Діалекти не лише збагачують мову, але й відображають культурну ідентичність окремих регіонів. Водночас їх недостатнє представлення в корпусах може призвести до їхньої маргіналізації. Під час укладання текстових корпусів варто знайти баланс між стандартизацією та збереженням діалектного багатства. Це важливо не лише для збереження мовної різноманітності, а й для створення універсальних моделей, здатних працювати з текстами різних регіональних особливостей.

Для врахування мовних особливостей під час формування корпусів української мови доцільно використовувати методи, які дозволяють інтегрувати різні варіанти мовлення, зокрема розмовну мову, діалекти та елементи суржику, через додаткове маркування чи лінгвістичну анотацію. Це може включати позначення регіональних особливостей або мовних відхилень, що сприятиме збереженню мовного багатства та підвищенню адаптивності корпусів до різних задач. Такі підходи дозволяють забезпечити різноманітність корпусів і створити базу, яка відображатиме стан мови, без втрати літературних стандартів.

1.4 Огляд джерел для вилучення тексту

У процесі створення текстових корпусів важливим аспектом є визначення джерел інформації та способів її вилучення. Вибір методів залежить від типу текстів, їхнього обсягу та специфіки подання в різних форматах. Новинні сайти є одним із найбільш доступних джерел сучасних текстів, які відображають літературну норму в публіцистичному стилі. Для вилучення текстів із таких

ресурсів застосовують HTML-парсинг. За допомогою інструментів на кшталт BeautifulSoup можна автоматизувати процес завантаження вебсторінок, вилучення заголовків, статей чи іншого текстового контенту [15].

Енциклопедичні ресурси, як-от Вікіпедія, пропонують структуровану текстову інформацію, яка охоплює широкий спектр тем. Доступ до текстів Вікіпедії здійснюється через її офіційне API, яке дозволяє автоматизовано отримувати дані. Зокрема, можна формувати запити, які повертають статті в текстовому або структурованому форматі JSON чи XML, що спрощує їх подальшу обробку. Окрім цього, існують сервіси, які приймають посилання на статті, автоматично завантажують їхній вміст та повертають готовий текст у зручному форматі.

Відеоплатформи, наприклад, YouTube, є важливим джерелом текстів. Для автоматизованої обробки відеоконтенту застосовують програми розпізнавання голосу, що забезпечують високоточне вилучення текстів із багатомовних матеріалів.

Друковані матеріали, такі як газети, журнали або архівні документи, вимагають використання технологій оптичного розпізнавання символів. Інструменти на кшталт Tesseract дозволяють конвертувати засканові сторінки у текстовий формат [16]. Наприклад, для обробки архіву української періодики слід забезпечити якісне сканування, яке потім проходить через OCR для розпізнавання тексту, з наступним виправленням можливих помилок.

Соціальні мережі, такі як Facebook, Twitter або Instagram, відображають реальний стан мовного середовища та є цінним джерелом розмовної мови. Сервіси мають як офіційне API, так і стороні сервіси. Зібрані тексти містять значний обсяг маргінальної лексики, діалектизмів та неформальних конструкцій, що потребує додаткового очищення та аналізу.

Наукові публікації, доступні через бібліотечні каталоги або відкриті бази даних, також є перспективним джерелом. Багато бібліотек надають доступ до даних через API або вебінтерфейси для пошуку та завантаження статей. Для

обробки матеріалів у форматі PDF застосовують бібліотеки, такі як PyPDF2, які дозволяють автоматично витягувати текст із файлів [17].

Кожен з описаних методів та джерел має свої особливості й обмеження. Наприклад, використання API потребує дотримання політики ліцензування, а застосування OCR може вимагати значних ресурсів для після обробки тексту. Оптимальне поєднання різних технологій дозволяє забезпечити широке охоплення мовного матеріалу та створити збалансовані текстові корпуси.

1.5 Проблеми непослідовності форматів в текстах новин

Одна з проблем нормалізації текстів новин – непослідовне використання форматів телефонних номерів. Різні автори та джерела дотримуються різних стандартів запису, що ускладнює автоматичний аналіз даних. Це можна проілюструвати на прикладі новинної статті «Як росіяни збирають інформацію про військовослужбовців з Полтавщини» [18], де в межах одного тексту зустрічаються одразу кілька варіантів запису номерів телефону. Позначимо довільну цифру буквою «X», тоді в тексті новини використано три різні формати номерів: +38 0XX XXX XX XX, +380 XXX XXX XXX та +380 XX XXX XXX. Якщо додатково розглянути статтю «Весняний ярмарок, день відкритих дверей і виготовлення окопних свічок: куди піти в Миколаєві вихідними» [19], то зустрічаються додаткові формати: +38 (0XX) XXX-XX-XX та (0XXX)XX-XX-XX.

Також є проблема з послідовним використанням лапок у текстах новин, проблема полягає в тому, що для їх позначення можуть використовуватись різні символи. Наприклад, в новині «Найбільші країни ЄС не підтримують пропозицію виділити 20 млрд пакет допомоги Україні – топдипломат ЄС» [20] використано декілька символів на позначення лапок: U+0022 та U+201D. Також в новинах часто ігнорується правопис вкладених лапок при цитуванні в цитаті, або пропускають символ зовсім.

Проблема апострофів у тексті новин полягає в тому, що для його позначення можуть використовуватись різні символи, які схожі між собою. Наприклад, в новині «Новий прем’єр-міністр Канади Марк Карні складе присяги

як 24-й очільник уряду країни в п'ятницю, 14 березня» [21] в одному реченні вжито одразу два різних символи на позначення апострофа: U+02BC та U+0027. Ці символи апострофів також можуть вживатись на позначення лапок, що створює додаткову неоднозначність.

Ці проблеми значно ускладнюють автоматичну обробку новинних текстів. Різноформатні записи одного й того ж типу даних – як-от телефонні номери, лапки чи апострофи – сприймаються системами обробки як різні сутності. Це збільшує розмірність текстових представлень, ускладнює побудову моделей та негативно впливає на якість класифікації, пошуку й інших NLP-завдань. Відсутність єдиних стандартів запису призводить до того, що моделі змушені враховувати зайві варіації, які не мають лінгвістичної цінності.

1.6 Методи попередньої обробки та нормалізації текстів

Попередня обробка текстів включає низку етапів, кожен із яких має на меті підготовку текстових даних до подальшого аналізу чи використання в мовних моделях. Серед основних завдань: токенізація, стемінг, лематизація та синтаксичний аналіз. Розглянемо ключові інструменти, їхні характеристики та порівняння.

Токенізація – це розбиття тексту на окремі значущі одиниці (токени). Складність для української мови полягає у багатій морфології. Для цього застосовують такі інструменти, як UDPipe, який пропонує алгоритми для токенізації з високою точністю [22]. Крім цього, бібліотека NLTK забезпечує базову токенізацію і є універсальним інструментом для роботи з багатьма мовами [23]. SpaCy, з іншого боку, забезпечує швидшу токенізацію та інтеграцію з іншими етапами обробки текстів, але вимагає більше ресурсів [24].

Однак переважно всі алгоритми розбивають на занадто дрібні токени. Ефективний токенізатор для українських текстів повинен зберігати смислові зв'язки між елементами складених конструкцій, що дозволить зберегти контекст та покращити якість подальшої обробки тексту. Розбиття семантично пов'язаних

елементів на окремі токени призводить до втрати контекстуальних зв'язків та збільшення розмірності моделі.

Стемінг зменшує слово до його основи (стему), відсікаючи закінчення. Для української мови існує кілька підходів. Алгоритм «tree_stem» забезпечує найменшу частку помилок (ERRT = 0.125) і є найшвидшим методом – у 24 рази швидше за лематизацію [25]. Він також має низький рівень недостатньої (UI = 0.0907) та надлишкової (OI = 2.71e-06) стемізації. Алгоритм «pymorphy2» використовує словникову базу, що дозволяє досягати високої точності (ERRT = 0.391), але програє «tree_stem» у швидкості. А «tapkomet» демонструє середні результати за всіма показниками (ERRT = 0.603) та є менш оптимізованим для великих корпусів.

Лематизація передбачає приведення слова до його базової форми, враховуючи контекст. UDPipe є універсальним інструментом для цього завдання, забезпечуючи підтримку формату CoNLL-U і можливість тренування на специфічних мовних даних [26]. NLTK, хоча і має модулі для лематизації, менш ефективний для української мови через обмежену підтримку морфологічних особливостей. SpaCy пропонує інтегровану систему для лематизації разом із токенізацією та синтаксичним парсингом, що робить його зручним для комплексних задач. Однак, для української мови його використання може бути обмеженим через нестачу спеціалізованих моделей.

Синтаксичний аналіз включає побудову дерев залежностей між словами. UDPipe забезпечує точні результати для української мови за умови навчання на якісних корпусах. У той час як NLTK більше підходить для базових задач, SpaCy демонструє кращі результати в реальних додатках завдяки використанню нейронних мереж і попередньо навчених моделей.

Методологічні виклики у створенні ефективних систем обробки текстів пов'язані з необхідністю врахування глибоких морфологічних особливостей мови, забезпеченням максимальної точності нормалізації та мінімізацією втрати семантичної інформації під час автоматизованої обробки. Також методи повинні розуміти контекст в якому вживається те, чи інше слово.

1.7 Лінгвістична анотація та розмітка

Лінгвістична анотація та розмітка є центральними етапами у створенні текстових корпусів, які забезпечують можливість автоматизованої обробки природної мови. Вони включають аналіз різних аспектів тексту, таких як морфологія, синтаксис, семантика та стилістика, що дозволяє не лише зберігати текстові дані, але й робити їх доступними для складних обчислювальних моделей.

Одним із найпоширеніших стандартів для анотації текстів є TEI [27]. Цей стандарт забезпечує універсальну систему позначення для текстів будь-якого типу – від літературних творів до листів і технічної документації. Наприклад, у TEI можна анотувати заголовки, епіграфи, адресати листів, підписи, а також структурні елементи, як-от початок або кінець текстових секцій. Завдяки своїй гнучкості TEI дозволяє деталізовано відображати метадані тексту. Для кожного розділу тексту можливо зазначити не лише заголовок або авторство, але й елементи відкриття (salutation) і закриття (closer), такі як підписи, постскриптуми або завершальні титули. Наприклад, для листів або передмов можливо окремо анотувати дати, авторів або місця створення тексту, що значно полегшує як аналіз, так і повторне використання цих даних.

Іншим прикладом сучасних підходів є CoNLL-U – формат, широко застосовуваний у галузі обробки природної мови, особливо у проєктах Universal Dependencies. CoNLL-U кодує текст у вигляді таблиць із 10 полів, кожне з яких відповідає певному аспекту анотації: слово, його основна форма (лема), частина мови (UPOS), синтаксичні відношення та морфологічні характеристики. Наприклад, поле UPOS містить універсальні частини мови, такі як іменник, дієслово чи прикметник, тоді як поле XPOS може включати більш детальні мовні специфікації. Поля FEATS та DEPREL дозволяють кодувати складніші морфологічні риси або відношення між словами у синтаксичному дереві. Така структура забезпечує високу уніфікацію, що дозволяє легко використовувати ці дані в багатомовних дослідженнях.

Особливий акцент у CoNLL-U робиться на синтаксичній розмітці. Вона не лише описує базові залежності між словами у реченні, але й підтримує розширені

графи залежностей, які включають додаткові зв’язки, наприклад, у випадках координації. Ці графи зберігаються у полі DEPS, де кожен зв’язок описується як пара з ідентифікатора слова та типу відношення. Наприклад, для речення «Вони купують і продають книги» можуть бути закодовані зв’язки, що відображають, як іменник «книги» підпорядковується одразу двом дієсловам.

Таким чином, стандарти TEI та CoNLL-U пропонують різні підходи до анотації текстів: перший фокусується на збереженні структурних і семантичних особливостей тексту, тоді як другий забезпечує компактне подання синтаксичних і морфологічних залежностей. Їхнє комбіноване використання дозволяє створювати багаторівневі корпуси, які можуть застосовуватися як для академічних досліджень, так і для практичних задач, таких як навчання нейронних мереж або розробка лінгвістичних застосунків.

1.8 Проблеми авторських прав

Створення корпусів текстів для потреб досліджень у сфері обробки природної мови супроводжується необхідністю врахування етичних норм та дотримання чинного законодавства щодо використання матеріалів, захищених авторським правом. Особливо це стосується вебресурсів, де більшість контенту підлягає під авторсько-правовий захист. На практиці це означає, що для збору текстів із таких джерел майже завжди потрібно отримувати дозвіл власників авторських прав. Винятком можуть бути коли тексти використовуються в межах ліцензій з відкритим доступом, або працювати з творами, що перебувають у суспільному надбанні. Наприклад, якщо брати новий портал Суспільне, то достатньо залишити посилання, бо медіа працює за кошти платників податків.

1.9 Постановка задачі

Метою роботи є підвищення якості україномовних текстових корпусів та систем для автоматичного їх формування шляхом розробки вдосконаленої САФТК з покращеними методами нормалізації та токенізації текстів.

Для досягнення поставленої мети потрібно провести дослідження методів формування текстових корпусів, виявити проблеми та запропонувати покращення. Необхідно дослідити методи та інструменти для вилучення тексту з різних джерел. Також необхідно дослідити підходи до нормалізації тексту при формуванні текстових корпусів, з акцентом на рішеннях щодо проблем непослідовності форматування телефонних номерів, лапок та апострофів. Важливою частиною роботи є дослідження підходів до токенізації, зі збереженням семантики токенів. Дослідити підходити для оцінки тексту та запропонувати методи щодо виправлення помилок. Також необхідно запропонувати підхід до структури корпусу та розмітки окремих документів.

2 АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

2.1 Аналіз підходів до створення українських корпусів

Розробка САФТК для української мови демонструє різноманітність архітектурних рішень, що відрізняються за ступенем автоматизації, гнучкістю та модульністю. Аналіз наявних відкритих систем дозволяє визначити їхні переваги та обмеження з погляду практичного застосування.

Система формування БрУК застосовує підхід з високим ступенем участі людини та обмеженою автоматизацією. Процес включає ретельний відбір текстів, їх попередню обробку та валідацію лінгвістами [1]. Для перевірки помилок використовується LanguageTool [28] в обгортці NLP-UK [29], проте остаточне рішення про включення тексту до корпусу приймає експерт. Така методологія забезпечує високу якість даних, але обмежує масштабованість системи.

Подібний підхід використовується при формуванні корпусу ГРАК [14], де залучається значна кількість людей для збору матеріалів. Обробка текстів також здійснюється за допомогою NLP-UK, але роль людини у відборі та валідації залишається суттєвою, що також обмежує масштабованість системи.

САФТК для UberText 1.0 являє собою перший крок до повністю автоматизованого формування масштабного корпусу української мови. Перша версія корпусу складається з перемішаних речень без збереження структури документів та метаданих, що обмежує її практичне застосування для деяких типів досліджень. Деталей щодо реалізації опубліковано не було.

Система формування UberText 2.0 представлена як розвинутий конвеєрний підхід з використанням MongoDB для зберігання документів. Однак аналіз публічно доступного вихідного коду [30] демонструє суттєві обмеження: у репозиторії наявний лише код, що працює з уже заповненою базою даних, надсилає тексти до NLP-UK та зберігає результати обробки окремими документами. Архітектура UberText 2.0, згідно з відкритим кодом, організована навколо завдань (TaskRQ) та відповідних їм класів задач (Job), де кожне завдання представляє окремий етап обробки (ProcessWithNlpUKJob, TagWithUDPipeJob, ExportCorpusJob). У межах обробки ProcessWithNlpUKJob передбачається

зберігання шарів обробки (токенізації, лематизації тощо) в окремій колекції MongoDB з посиланнями з документів основної колекції. Важливо зазначити, що код для збору початкових даних, включно з описаними в публікації Scrapy-павуками, у відкритому доступі відсутній. Також, хоча автор стверджує, що система зберігає структуру новинних текстів (заголовки, абзаци, цитати та списки), це неможливо підтвердити аналізом відкритого коду. Ба більше, на офіційному сайті корпусу [31] для завантаження доступний лише базовий текст без будь-якої розмітки.

2.2 Огляд методів та технологій для вилучення текстів

Для кожного типу джерел даних потрібно використовувати різні інструменти. Якщо для вилучення тексту потрібно оптичне розпізнавання, Python надає широкий вибір інструментів. Кожна бібліотека має свої особливості, які можуть бути корисними залежно від завдань. Tesseract є однією з найпоширеніших бібліотек, відомою своєю високою точністю, підтримкою багатьох мов і широкими можливостями налаштування. Для інтеграції в Python можна використовувати обгортку Pytesseract [32]. EasyOCR відзначається простотою використання, підтримкою різних мов і форматів тексту, а також здатністю розпізнавати як друковані, так і рукописні тексти [33]. Doctr спеціалізується на аналізі документів, включаючи визначення текстових областей, таблиць та інших елементів документа [34]. Keras-OCR дозволяє легко інтегрувати розпізнавання тексту в проєкти завдяки гнучкості налаштувань і високій точності [35]. GOCR є простішим інструментом із базовою функціональністю, проте має обмеження в підтримці мов і форматів [36].

Якщо текст треба вилучити з аудіо, то можна використовувати хмарні сервіси. IBM Speech to Text – забезпечує високоякісну транскрипцію аудіо завдяки поєднанню знань про граматику, структуру мови та звукові сигнали, однак не підтримує українську мову [37]. Microsoft Bing Voice Recognition –для української доступний лише звичайний текст без автоматичної пунктуації та інших розширених функцій як визначення мовців, оцінки впевненості тощо [38].

Google Cloud – для української доступні такі функції як оцінка впевненості транскрипції та фільтр нецензурних слів, однак пунктуація та визначення мовців не доступні [39]. OpenAI Whisper – автоматично додає коми, крапки, знаки питання та інші пунктуаційні символи, забезпечуючи зручне й читабельне оформлення тексту. Українська мова є в списку підтримуваних мов, однак немає уточнення щодо конкретних функцій для кожної мови [40]. Модель підтримує підказки (prompts), які допомагають покращити транскрипцію, додаючи пунктуацію, коригуючи складні слова, зберігаючи контекст і налаштовуючи стиль тексту. Amazon Transcribe – сервіс для перетворення мови в текст. Забезпечує високу точність розпізнавання мови навіть при наявності фонових шумів або присутності акценту [4]. Серед сервісів розпізнавання мови з аудіо та відео джерел слід зазначити TurboScribe, що дозволяє безплатно щоденно обробляти до 3-х файлів тривалістю до 30 хвилин та підтримує українську мову [5].

Одним з джерел даних може бути платформа YouTube. Для завантаження відео можна використовувати різні інструменти, наприклад, Pafy – бібліотека Python для завантаження контенту з YouTube та отримання метаданих [41]. Також можна скористатись PyTube, якщо потрібно завантажити лише mp3 [42].

Якщо ж немає API чи відповідної бібліотеки, наприклад, треба отримати текст з конкретного сайту новин, то можна використовувати стандартний парсинг HTML. Програмно це можна реалізувати за допомогою бібліотек Beautiful Soup або Scrapy. Beautiful Soup – це бібліотека для парсингу HTML і XML документів. Вона допомагає організувати та обробити структуру HTML для подальшого витягнення даних з вебсторінки. Scrapy – це вебфреймворк для побудови вебскрейперів, з акцентом на швидкість, масштабованість і розширюваність [43]. Також може працювати з сайтами, що вимагають авторизації, містять AJAX або динамічно створюваний контент.

2.3 Аналіз підходів нормалізації українських текстів

Нормалізація тексту є критичним етапом попередньої обробки для більшості завдань з обробки природної мови. Як пояснюють Aliero та ін. [44] у

своєму систематичному огляді, нормалізація включає «виправлення орфографічних помилок, розширення абревіатур, розв'язання скорочень, нормалізацію пунктуації, капіталізації та інших лінгвістичних варіацій для забезпечення узгоджених і зв'язних представлень текстових даних». Їхній аналіз 54 статей, опублікованих між 2018-2022 роками, визначає чотири основні підходи до нормалізації: методи на основі правил, що використовують попередньо визначені правила; методи на основі статистичних моделей, що навчаються на даних; методи на основі нейронних мереж, що використовують глибоке навчання; та гібридні методи, що поєднують декілька підходів. Для спеціалізованих завдань нормалізації, як-от стандартизація телефонних номерів, часто застосовуються підходи на основі правил, хоча в галузі спостерігається тенденція до використання нейромережевих рішень для складніших завдань нормалізації. Для українських текстів розроблено декілька підходів, які розв’язують специфічні проблеми нормалізації.

Старко та ін. [45] описують систему попередньої обробки текстів для корпусу ГРАК, реалізовану на Groovy та доступну на GitHub. Ця система використовує український модуль у LanguageTool для верифікації токенів і розв’язує такі проблеми, як змішування латинських і кириличних символів, нестандартні апострофи та проблеми з дефісами. Хоча система ефективна для загального очищення тексту, вона не має спеціалізованих механізмів для стандартизації телефонних номерів, належного оброблення вкладених лапок та розрізнення апострофів і лапок, що є особливо проблематичним у новинних текстах українською мовою.

Вакуленко [46] представляє інший підхід до нормалізації українського тексту, зосереджуючись на стандартизації літер, числівників і одиниць вимірювання для перетворення тексту в мовлення. Його робота містить детальні правила для перетворення українських літер у мовлення та нормалізації кількісних і порядкових числівників, простих дробів та вимірювань. Цей підхід важливий для завдань синтезу мовлення, але вирішує інші аспекти нормалізації порівняно з форматуванням у новинних текстах.

Чаплинський [3] представляє UberText 2.0, масштабний корпус сучасної української мови з конвеєрами попередньої обробки, що включають різні етапи нормалізації. Система корпусу розв’язує проблеми якості тексту як автоматичними, так і ручними засобами, зазначаючи, що «підтримка бажаної якості даних у масивному корпусі є складною, особливо коли вони збираються з джерел, які редактори корпусу не контролюють» [3]. Їхній підхід включає спеціальні павуки, написані для кожного джерела даних, щоб відфільтрувати шаблонні тексти, та автоматичне визначення мови для виключення неукраїнського контенту. Однак для очищення так само використовується модуль NLP-UK.

Cudak та ін. [47] розглядають нормалізацію телефонних номерів у рамках своїх досліджень щодо зіставлення POI (Points of Interest). Їхній підхід до нормалізації зосереджений на забезпеченні точного порівняння телефонних номерів з різних джерел даних, а не на стандартизації форматів відображення. Автори нормалізують телефонні номери, видаляючи всі непотрібні символи (знак плюса, дефіси, пробіли та дужки) та приводить номери до однакової довжини шляхом видалення кодів регіонів або початкових нулів за потреби. Ця попередня обробка спеціально розроблена для оптимізації обчислення метрик подібності за допомогою відстані Левенштейна. Їхня метрика подібності телефонних номерів досягла високої точності (93,8%) та показників AUC-ROC (0,904) у їхніх експериментах, демонструючи її ефективність для порівняння POI. На відміну від нашого підходу, який спрямований на стандартизацію форматів представлення відповідно до офіційних рекомендацій, Cudak та ін. зосереджуються на нормалізації для цілей обчислювального порівняння, особливо в інтеграції геопросторових даних, де телефонні номери служать одним із кількох атрибутів для зіставлення сутностей.

У той час як більшість описаних підходів до нормалізації зосереджені на одномовних текстах, Van der Goot та Çetinoğlu [48] розв’язують проблему лексичної нормалізації в даних зі змішаними мовами. Вони оцінюють три моделі нормалізації, спеціально розроблені для текстів зі змішаними мовами для двох

мовних пар (турецько-німецька та індонезійсько-англійська), показуючи, що нормалізація покращує подальші задачі, такі як POS-розмітка, із 5,4% відносним підвищенням продуктивності. Їхнє порівняння стратегій нормалізації з урахуванням мови та незалежних від мови надає цінні уявлення про обробку текстів зі змішаними наборами символів та орфографії – проблема, актуальна для українських ЗМІ, де запозичені слова та конвенції форматування вносять непослідовності. На відміну від нашого підходу, який зосереджений на специфічних проблемах нормалізації, їхня робота розглядає широку лексичну нормалізацію в багатомовних контекстах, але обидва підходи мають на меті стандартизацію непослідовних текстових елементів

Хоча ГРАК, Браунський корпус та UberText 2 використовують бібліотеку NLP-UK для базової обробки українського тексту, цей інструмент не надає повного рішення для специфічних проблем нормалізації, що постають у новинних текстах.

2.4 Аналіз підходів оцінки якості українських текстів

Оцінка якості текстів є важливим етапом у створенні лінгвістичних корпусів, оскільки від неї залежить точність подальших лінгвістичних досліджень та моделей обробки природної мови. Основна мета цього процесу – виявити помилки в текстах, оцінити їхню репрезентативність та відповідність поставленим завданням.

Сучасні інструменти, такі як LanguageTool або Grammarly [49], дозволяють автоматизувати процес виявлення орфографічних, граматичних та стилістичних помилок. LanguageTool, наприклад, можна налаштувати для аналізу україномовних текстів, що є важливим для врахування специфіки граматики та синтаксису. Однак, навіть найкращі інструменти такого типу не гарантують абсолютної точності, особливо для текстів, що містять складні синтаксичні конструкції або специфічну лексику.

Основною складністю у роботі є коректне виявлення помилок у тексті. Наприклад, помилка в чергуванні «у/в» відстежується за допомогою регулярного

виразу, і її виправлення завжди покращує текст, незалежно від контексту. Однак у випадках із чергуванням «і/й» виникають проблеми. Наприклад, у реченні «Батьки і діти» заміна сполучника на «й» є неправильною, хоча Language Tool пропонує таке виправлення. Подібні помилки потребують складнішого аналізу контексту. Інший приклад – слово «графіку» у реченні «відбулись зміни до графіку погодинних відключень електроенергії». Language Tool може вказати на неузгодженість і запропонувати виправлення на «графіка» чи «графіки». Але немає простого рішення щодо обрання заміни. Водночас у реченні «введено додатково 1 черга відключень» це ж саме слово «черга» залишається поза увагою. Такі ситуації потребують використання комбінованих методів для пошуку та виправлення помилок.

Одним з рішень, що краще оцінюють контекст є ChatGPT [50]. Однак рішення має свої проблеми, оскільки модель може придумувати або ігнорувати правила, чи просто робити надмірне редагування. Використання ChatGPT вимагає правильно формування запитів та формування бази знань моделі.

Кількість помилок, виявлених такими системами, може слугувати показником якості тексту. Проте, важливо зазначити, що виявлення помилок – це лише початковий етап. Після цього варто ще виконати виправлення помилок, що може вимагати додаткових досліджень або ручної обробки.

2.5 Аналіз підходів токенізації українських текстів

Токенізація є одним з основних етапів обробки природної мови, що полягає в поділі тексту на базові елементи, токени. Для української мови, з її багатою морфологією та специфічними орфографічними особливостями, токенізація представляє низку особливих викликів: обробка дефісів та апострофів; скорочення та абревіатури (наприклад, ДСТУ 3582:2013 [51]); одиниці виміру та спеціальні позначення; числові вирази; телефонні номери, електронні адреси та посилання.

Для поділу українських текстів на слова наявно декілька популярних рішень. Серед них можна виділити spaCy (з пакетом «uk_core_news_sm») [24],

stanza (з пакетом «uk») [52], NLTK [23], а також токенізатор на основі Language Tool, що також використовується в БрУК. Додатково, проєкт Lang-UK пропонує власний токенізатор [53].

Наявні токенізатори відрізняються підходами до визначення границь токенів. SpaCy використовує правила на основі префіксів, суфіксів та інфіксів, що дозволяє гнучко налаштовувати токенізацію. Stanza базується на навчених нейронних мережах і універсальному токенізаторі. NLTK пропонує кілька варіантів токенізації, включаючи регулярні вирази та правила. Токенізатор Language Tool орієнтований на роботу з перевіркою правопису, що зумовлює його особливості. Lang-UK використовує спеціально розроблений для української мови алгоритм, заснований на регулярних виразах.

Головною проблемою токенізації текстів є відсутність збереження семантичної цілісності певних конструкцій. Такі лексичні одиниці як «км/год», «°С», «№11» сприймаються людиною як єдині семантичні конструкції, що представляють цілісні поняття або величини. Однак більшість токенізаторів розділяють їх на окремі частини, що може призвести до втрати контексту в подальших етапах обробки.

2.6 Підсумки розділу

Наразі немає гнучкого та масштабованого рішення для систем автоматичного формування текстових корпусів. До того ж компоненти нормалізації та токенізації, що використовуються в сучасних рішеннях, мають інші проблеми, тому потребують покращення. Компоненти нормалізації не пропонують всеосяжного рішення для нормалізації українських телефонних номерів, лапок та апострофів, а компоненти токенізації ігнорують семантичну єдність типових конструкцій. Для розв’язання цих питань необхідно розробити нову САФТК з покращеними компонентами нормалізації та токенізації згідно з чинними рекомендаціями щодо правопису та інших державних стандартів.

3 ОПИС ВИМОГ ДО СИСТЕМИ

3.1 Вимоги до нормалізації українських текстів

На основі проведеного аналізу літературних джерел та виявлених обмежень наявних рішень, необхідно розширити стандартні методи нормалізації, що пропонує пакет Language Tool, специфічними покращеннями для обробки новинних текстів українською мовою [54]. Ці покращення повинні забезпечити однорідність представлення телефонних номерів, апострофів та лапок, що значно підвищить якість автоматизованого формування текстових корпусів.

3.1.1 Нормалізація телефонних номерів

Згідно з дизайн-системою державних сайтів України [55] рекомендований формат телефонних номерів: +380 (XX) XXX-XX-XX. Однак на сайті не описаний формат для спеціальних номерів, що мають три цифри коду. Тому пропонуємо додатковий формат для таких номерів: +380 (XXX) XX-XX-XX. Такий формат введено оскільки номер, що починається на «+380 (800) XX…» краще розрізняється людиною як безплатний номер на відміну від «+380 (80) 0X…».

Щоб зменшити неправильну заміну через неоднозначність групи цифр пропонуємо обмежити нормалізацію українськими номерами: номери повинні починатися з коду країни «+380» або з нуля, і після цього повинен йти код українського оператора згідно зі списком Vodafone [56].

На основі цих вимог необхідно розробити нормалізатор, який забезпечуватиме уніфіковану обробку телефонних номерів у новинних текстах.

3.1.2 Нормалізація лапок

Згідно з чинним правописом [57, с. 246], доцільно використовувати «лапки-ялинки» («…» – U+00AB та U+00BB) для зовнішніх лапок та «лапки-лапки» для внутрішніх (“…” – U+201C та U+201D). Такий підхід відповідає українській типографській традиції та полегшує автоматичне розпізнавання структури цитат.

Основна проблема нормалізації лапок полягає в тому, що рекомендовані символи для лапок залежать від позиції: U+00AB перед цитатою, а U+00BB після,

однак типовий символ на їх позначення – U+0022, що не вказує, з лівого чи правого боку він встановлений.

Для нормалізації апострофа необхідно розробити алгоритм, що замінює всі лапки на рекомендовані, з врахуванням контексту та забезпечує правильне чергування внутрішніх і зовнішніх лапок.

3.1.3 Нормалізація апострофів

Хоча в українському правописі немає конкретних рекомендацій щодо використання певного символу для позначення апострофа, для уніфікації корпусу необхідно обрати єдиний стандарт. LanguageTool використовує U+0027, однак для нашої нормалізації пропонуємо використовувати U+02BC. Це зумовлено, зокрема, тим, що U+02BC використовується за замовчуванням під час введення тексту на пристроях Apple в українській клавіатурі, а також тим, що шрифти надають йому закругленої форми, що відповідає традиційному написанню апострофа в рукописних текстах.

Для нормалізації апострофа необхідно розробити алгоритм, що розпізнає та замінює різні символи, що використовуються як апострофи та розрізняє апострофи, що вжиті як лапки.

3.2 Вимоги до токенізації українських текстів

Ефективний токенізатор для українських текстів повинен зберігати смислові зв'язки між елементами складених конструкцій, що дозволить зберегти контекст та покращити якість подальшої обробки тексту. Розбиття семантично пов'язаних елементів на окремі токени призводить до втрати контекстуальних зв'язків та збільшення розмірності моделі. Токенізатор повинен коректно опрацьовувати наступні випадки, зберігаючи їх як єдині токени:

- слова з дефісом, наприклад, складні слова «будь-який», «бозна-що», а також словоформи на кшталт «2022-му», «37-річна»;
- числові діапазони, зокрема конструкції «15-21», «№1-4», що позначають певний діапазон;

– дати та час, наприклад, формати дат «18.05.2021» і часу «22:37»;
– порядкові номери, зокрема «№11», «№3,4»;
– електронні адреси, наприклад «rename@city.kharkiv.ua»;
– телефонні номери у різних форматах, наприклад «+38 (095) 568 38 77»;
– одиниці виміру, наприклад «м²», «км²/місяць», «м/с»;
– температурні показники, зокрема записи на кшталт «-21°C»;
– грошові суми, наприклад «$1,461»;
– спеціальні скорочення, що відповідають чинному ДСТУ 3582:2013 [51], наприклад, «м.» (місто), «вул.» (вулиця), «пл.» (площа) та «ред.» (редакція);
– слова з апострофом, наприклад «сім’я»;
– числа з розділювачами, зокрема десяткові дроби «234,5» та числа з пробілами між розрядами «1 000 000»;
– приблизні значення, наприклад, конструкції з тильдою «~100».

Варто зазначити, що за потреби додаткову токенізацію таких конструкцій легше виконати на наступних етапах обробки, аніж відновлювати надмірно розділені токени.

3.3 Вимоги до архітектури системи обробки текстових корпусів

Для ефективного формування текстових корпусів необхідно розробити модульну систему, що забезпечуватиме повний цикл обробки тексту від отримання даних з різних джерел до їх аналізу та структурованого збереження. Система повинна відповідати вимогам гнучкості, розширюваності та забезпечувати високу якість обробки текстів.

3.3.1 Процес обробки тексту та функціональні вимоги

Система автоматизованого формування текстових корпусів повинна забезпечувати послідовне виконання наступних етапів обробки:

– отримання даних. Система має отримувати дані з різноманітних джерел (вебсайти, електронні документи, медіаресурси) за визначеним

інтервалом. Кожен запис повинен містити необхідні метадані (джерело, автор, заголовок, дата публікації, посилання) та механізм доступу до оригінального контенту;

– парсинг даних. Система повинна витягувати з вихідних даних основний текстовий контент, зберігаючи його структуру (заголовки, абзаци, цитати, списки). Сторонній вміст, що не належить до основного матеріалу (реклама, навігаційні елементи, коментарі) має бути відфільтрований. Результат цього етапу – документ з базовою структурою тексту;
– нормалізація тексту. Система повинна уніфікувати текст згідно з визначеними у пункті 3.1 правилами. Кожна виявлена проблема форматування повинна бути зафіксована з унікальним ідентифікатором та відповідним типом (попередження чи помилка). Результатом обробки є нормалізований документ;
– оцінка та виправлення. Система має оцінити текст на наявність граматичних, орфографічних та стилістичних помилок і передати цю інформацію в компонент виправлення. Процес оцінки та виправлення повинен бути ітеративним, з обмеженням на максимальну кількість ітерацій. Результатом обробки є оцінений документ з виправленнями та списком залишкових проблем;
– токенізація. Текст має бути розбитий на речення та токени згідно з вимогами, описаними в пункті 3.2. Результатом обробки є токенізований документ.

3.3.2 Формат документа на кожному етапі обробки

Для забезпечення послідовного збереження структури та семантики тексту на всіх етапах обробки, система має підтримувати єдиний формат документа з поступовим розширенням його структури.

Базовий формат має зберігати оригінальну структуру тексту (заголовки, параграфи, списки та цитати), водночас включаючи всі необхідні метадані:

джерело, автор, заголовок, мова, посилання та дата публікації. Цей формат відображає результат первинної обробки тексту після парсингу.

Нормалізований формат повинен містити уніфікований текст та початкову аналітичну інформацію у вигляді списку попереджень та помилок форматування. Кожна проблема повинна мати унікальний ідентифікатор, а елементи тексту повинні містити посилання на проблеми, які їх стосуються, забезпечуючи таким чином прозору діагностику.

Оцінений формат доповнює нормалізований інформацією про виявлені лінгвістичні помилки та результати їх виправлення. Помилки чи інші попередження повинні мати схожу систему ідентифікаторів та посилань на елементи тексту. Це дозволяє відстежувати процес обробки тексту та аналізувати його якість.

Токенізований формат додає деталізацію до кожного елемента тексту, розбиваючи його на речення та токени. Структура документа зберігається, але кожен структурний елемент тепер містить ієрархію речень та токенів. При цьому зберігаються всі ідентифікатори попереджень та помилок, що забезпечує цілісність аналітичної інформації та можливість подальшого аналізу.

3.3.3 Архітектурні вимоги до системи

Для забезпечення гнучкості, розширюваності та ефективності система повинна відповідати наступним архітектурним вимогам:

- модульність. Система повинна складатися з незалежних компонентів, кожен з яких відповідає за конкретний етап обробки тексту. Це дозволить замінювати окремі компоненти без необхідності перебудови всієї системи;
- конвеєрна архітектура. Компоненти повинні бути об'єднані в єдиний конвеєр обробки, де вихідні дані одного компонента стають вхідними для наступного. Конвеєр повинен забезпечувати послідовну трансформацію даних через усі етапи обробки.

– розділення відповідальності. Кожен компонент повинен мати чітко визначену сферу відповідальності;
– розширюваність. Система повинна легко адаптуватися до нових джерел даних через механізми абстракції та інтерфейси. Додавання нового джерела даних повинно вимагати лише реалізації відповідного компонента без зміни загальної архітектури;

3.3.4 Практичні вимоги до реалізації

Для практичної реалізації та демонстрації архітектури необхідно розробити конвеєр обробки текстів із порталу «Суспільне Новини». Конвеєр повинен відповідати таким вимогам:

– інтеграція з джерелом. Система має отримувати новинні статті з порталу «Суспільне Новини» за вчорашній день;
– парсинг HTML. Система має вилучати основний текст зі збереженням його структури з HTML-сторінки;
– нормалізація специфічних елементів. Система має виконувати нормалізацію телефонних номерів, апострофів та лапок відповідно до правил, визначених у пункті 3.1;
– інтеграція з оцінювачем якості. Система має взаємодіяти з інструментами перевірки української мови. Компонент виправлення є опціональним для першої версії системи;
– токенізація з урахуванням семантики. Система має виконувати токенізацію, зберігаючи семантичну цілісність спеціальних конструкцій згідно з вимогами, описаними в пункті 3.2;
– зберігання проміжних результатів. Система має зберігати документи на кожному етапі обробки (базовий, нормалізований, оцінений, токенізований);
– можливість підміни компонентів. Архітектура має дозволяти легко замінювати окремі компоненти, наприклад, використовувати різні токенізатори чи нормалізатори без зміни загальної логіки конвеєра.

4 РОЗРОБКА ПРАКТИЧНОЇ ЧАСТИНИ

4.1 Абстрактний конвеєр обробки текстів

Ядром системи автоматизованого формування текстових корпусів є абстрактний клас AbstractPipeline, що реалізує конвеєрний підхід до обробки текстів. Цей клас визначає загальну структуру процесу обробки та взаємодію між компонентами, забезпечуючи розширюваність та гнучкість системи.

4.1.1 Архітектура конвеєра

Клас AbstractPipeline (конвеєр) спроєктовано за принципом шаблонного методу, де базовий клас визначає скелет алгоритму, а конкретні реалізації підкласів заповнюють необхідні деталі. Цей підхід забезпечує максимальну гнучкість у використанні різних джерел даних, методів парсингу, нормалізації та токенізації без зміни загальної логіки обробки. Архітектура конвеєра передбачає послідовне виконання наступних етапів з використанням відповідних інтерфейсів:

- отримання записів із джерела даних через метод records(start, end) класу AbstractSource. Цей метод повертає ітератор об'єктів Record, кожен з яких надає доступ до метаданих через атрибут metadata та до вмісту через метод content(). Параметри start та end дозволяють обмежити вибірку записів певним часовим діапазоном;
- парсинг вмісту та створення базового документа за допомогою методу parse(content, metadata, annotator) класу AbstractParser. Цей метод аналізує вміст, витягує структуровану інформацію та створює документ відповідного формату. Важливо зазначити, що парсер отримує клас анотатора, оскільки тільки анотатор знає структуру документа та надає методи для його створення (create_document, add_heading, add_paragraph тощо);
- нормалізація тексту згідно з визначеними правилами через метод normalize(text) класу AbstractNormalizer. Метод повертає кортеж з трьох елементів: нормалізований текст, список попереджень та список помилок;

– оцінка та виправлення тексту з використанням методу evaluate(text) класу AbstractEvaluator, який аналізує текст на наявність помилок, та методу correct(text, warnings, errors) класу AbstractCorrector, який виправляє виявлені проблеми;
– токенізація текстових елементів через метод tokenize(text) класу AbstractTokenizer для розбиття тексту на речення та токени, з подальшим оновленням структури документа за допомогою методу split_elements() класу AbstractAnnotator;
– збереження документів на кожному етапі обробки за допомогою методу save(content, name) класу AbstractSaver, який записує вміст документа у відповідний формат.

Для взаємодії з цими компонентами анотатор надає ряд методів для роботи з документом:

– get_string(document) перетворює документ у текстове представлення;
– update_elements(document, update_fn) застосовує функцію оновлення до текстових елементів, отримує функцію що модифікує текст елемента та повертає новий ідентифікатор для нього;
– add_warnings(document, warnings) та add_errors(document, errors) додають відомості про виявлені проблеми.

Така архітектура забезпечує чітке розділення відповідальності між компонентами, дозволяючи незалежно розробляти та тестувати кожен етап обробки. Крім того, завдяки використанню абстрактних інтерфейсів система легко масштабується для підтримки нових джерел даних, форматів документів та методів обробки текстів.

4.1.2 Процес ініціалізації та виконання конвеєра

Конструктор класу AbstractPipeline відповідає за ініціалізацію всіх необхідних компонентів через абстрактні методи (див. рисунок 4.1).

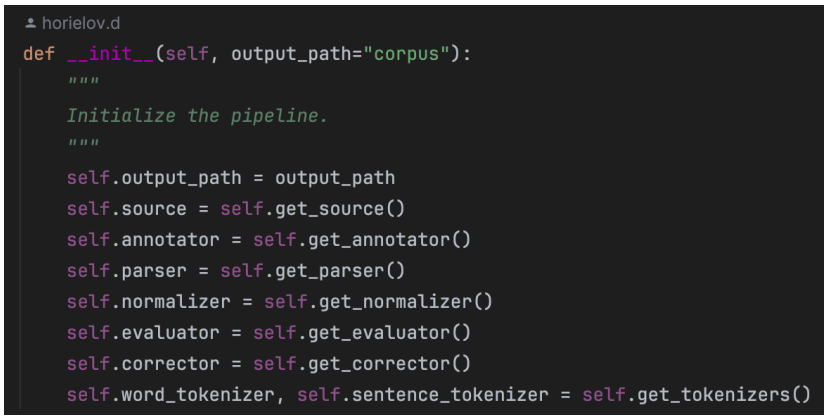


Рисунок 4.1 – Ініціалізація конвеєра (виконано самостійно)

Кожен з методів отримання (наприклад, get_source) є абстрактним і повинен бути реалізований у підкласах для надання конкретних реалізацій компонентів.

Метод execute(start, end) є основною точкою входу для запуску конвеєра (див. рисунок 4.2).

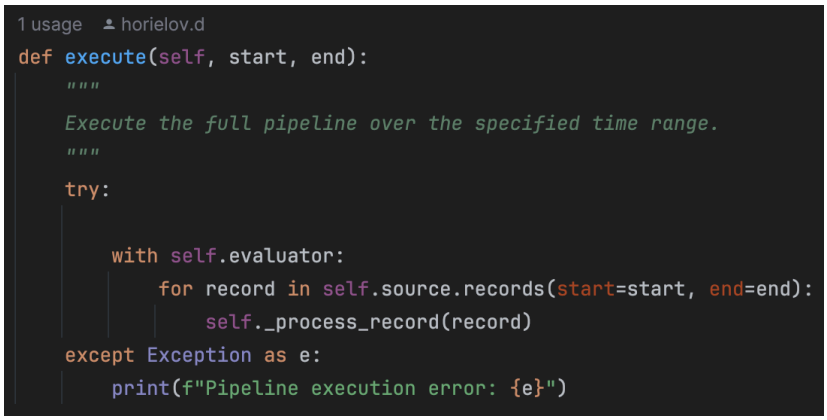


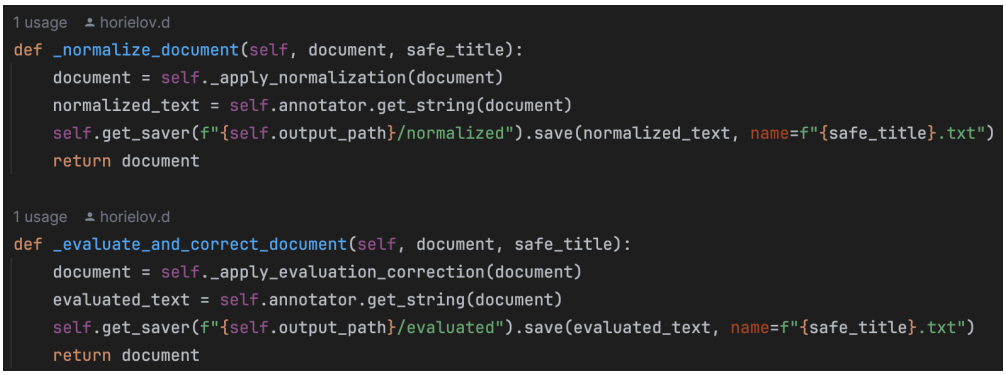
Рисунок 4.2 – Основний метод виконання (виконано самостійно)

Він відповідає за ітерацію по всіх записах, отриманих з джерела даних у заданому часовому діапазоні, та обробку кожного запису через метод _process_record(). Використання менеджера контексту with self.evaluator забезпечує коректну ініціалізацію та завершення роботи компонента оцінки, що особливо важливо для ресурсомістких інструментів, таких як LanguageTool. Метод _process_record() координує обробку окремого запису через виклик спеціалізованих методів для кожного етапу обробки (див. рисунок 4.3).

```
1 usage  horielov.d
def _process_record(self, record):
    """
    Process a single record through the pipeline.
    """
    title = record.metadata.title
    safe_title = sanitize_and_transliterate(title)
    print(f"Processing: {title}")

    self.warning_counter = 1
    self.error_counter = 1

    document = self._parse_document(record, safe_title)
    document = self._normalize_document(document, safe_title)
    document = self._evaluate_and_correct_document(document, safe_title)
    document = self._tokenize_document(document, safe_title)

    print(f"Finished: {title}")
```

Рисунок 4.3 – Метод обробки новин (виконано самостійно)

Кожен з методів обробки, наприклад, _normalize_document() та _evaluate_and_correct_document(), відповідають за один етап обробки та зберігають проміжний результат, що дозволяє аналізувати та налагоджувати різні аспекти системи (див. рисунок 4.4).



```
1 usage  horielov.d
def _normalize_document(self, document, safe_title):
    document = self._apply_normalization(document)
    normalized_text = self.annotator.get_string(document)
    self.get_saver(f"{self.output_path}/normalized").save(normalized_text, name=f"{safe_title}.txt")
    return document

1 usage  horielov.d
def _evaluate_and_correct_document(self, document, safe_title):
    document = self._apply_evaluation_correction(document)
    evaluated_text = self.annotator.get_string(document)
    self.get_saver(f"{self.output_path}/evaluated").save(evaluated_text, name=f"{safe_title}.txt")
    return document
```

Рисунок 4.4 – Методи нормалізації та оцінки документа (виконано самостійно)

Метод _apply_normalization(), що використовується в _normalize_document(), оновлює текстові елементи документа, застосовуючи нормалізатор до кожного елемента. (див. рисунок 4.5). Якщо під час нормалізації були виявлені помилки чи попередження, він відстежує їх, генеруючи відповідні ідентифікатори.

```
1 usage  horielov.d *
def _apply_normalization(self, document):
    warnings = []
    errors = []

    new *
    def normalize(text):
        new_text, new_warnings, new_errors = self.normalizer.normalize(text)

        ids = []
        for err in new_errors:
            e_id = f"e{self.error_counter}"
            self.error_counter += 1
            errors.append((e_id, err))
            ids.append(e_id)

        for warn in new_warnings:
            w_id = f"w{self.warning_counter}"
            self.warning_counter += 1
            warnings.append((w_id, warn))
            ids.append(w_id)

        return new_text, ", ".join(ids) if ids else None

    document = self.annotator.update_elements(document, normalize)
    self.annotator.add_warnings(document, warnings)
    self.annotator.add_errors(document, errors)

    return document
```

Рисунок 4.5 – Виконання нормалізації документа (виконано самостійно)

Метод _apply_evaluation_correction(), що використовується в _evaluate_and_correct_document (див. рисунок 4.4), реалізує ітеративний процес оцінки та виправлення тексту. Він спочатку створює порожні списки для попереджень та помилок. Потім визначає функцію обробки окремого текстового елемента, яка спочатку оцінює текст за допомогою evaluator і далі запускає цикл виправлень. У цьому циклі текст, попередження та помилки передаються компоненту corrector, який намагається виправити проблеми. Процес повторюється до трьох разів або доки текст не перестане змінюватись. Після завершення циклу всім залишковим проблемам присвоюються унікальні ідентифікатори. Ця функція застосовується до всіх елементів документа, а виявлені попередження та помилки додаються до документа. Такий підхід забезпечує поступове покращення якості тексту та збереження інформації про виявлені проблеми.

4.1.3 Конкретна реалізація конвеєра

Для демонстрації роботи системи було реалізовано конкретний підклас SuspilnePipeline, який надає реалізації для всіх абстрактних методів базового класу. Цей підклас використовує компоненти, специфічні для обробки новин з порталу «Суспільне Новини»:

- джерело даних SuspilneSource, що отримує новини з порталу «Суспільне Новини» за вказаний часовий діапазон. Цей компонент використовує HTTP-запити до архівної сторінки порталу для отримання списку новин та створює об'єкти SuspilneRecord для кожної новини;
- анотатор XMLAnnotator, що забезпечує XML-представлення документів. Він надає методи для створення та модифікації документів у форматі XML, а також для розбиття тексту на речення та токени;
- парсер SuspilneParser, що витягує структурований текст з HTML-сторінок «Суспільне Новини». Він аналізує HTML-структуру сторінки та виділяє заголовки, параграфи, списки та цитати, використовуючи анотатор для створення відповідного документа;
- нормалізатор NewsNormalizer, що реалізує покращенні правила нормалізації для новинних текстів, зокрема для телефонних номерів, апострофів та лапок;
- оцінювач LanguageToolEvaluator, що використовує LanguageTool для виявлення орфографічних та граматичних помилок у тексті;
- заглушка для компонента виправлення NoChangesCorrector, що не вносить змін у текст. У майбутніх версіях системи цей компонент може бути замінений на функціональніший, наприклад, на основі ChatGPT;
- удосконалений токенізатор для української мови CustomTokenizer, що враховує особливості семантично цілісних конструкцій;
- компонент для збереження результатів у файлову систему FileSaver, що організовує файли за етапами обробки.

4.2 Анотація документів і структурне представлення текстів

Для забезпечення уніфікованого підходу до роботи з документами різних форматів та збереження їх структури було розроблено абстрактний клас AbstractAnnotator. Цей клас визначає загальний інтерфейс для створення, модифікації та аналізу документів, незалежно від їх кінцевого представлення.

4.2.1 Абстрактний інтерфейс анотатора

Клас AbstractAnnotator визначає набір методів, що повинні бути реалізовані будь-яким анотатором:

- name(), що повертає назву анотатора;
- create_document(metadata), що створює новий документ із вказаними метаданими;
- add_heading(document, text), що додає заголовок до документа;
- add_paragraph(document, text), що додає параграф до документа;
- add_list(document, items, ordered), що додає список до документа;
- add_quote(document, text), що додає цитату до документа;
- get_string(document), що перетворює документ у рядкове представлення;
- update_elements(document, update_fn), що оновлює текстові елементи документа;
- add_warnings(document, warnings), що додає попередження до документа;
- add_errors(document, errors), додає помилки до документа;
- split_elements(document, split_s_fn, split_w_fn), що розбиває текстові елементи на речення та токени.

Такий інтерфейс забезпечує чітке розділення між логікою обробки документів та їх конкретним представленням, що дозволяє легко замінювати формат документів без зміни основної логіки системи. Наприклад, для підтримки формату JSON достатньо реалізувати новий клас JSONAnnotator, що реалізує інтерфейс AbstractAnnotator.

4.2.2 XML-представлення документів

Клас XMLAnnotator реалізує інтерфейс AbstractAnnotator для створення та маніпуляції документами у форматі XML. Він використовує бібліотеку lxml для ефективної роботи з XML-документами. Кожен XML-документ має наступну базову структуру (див. рисунок 4.6).

```xml
<root>
  <metadata>
    <!-- Метадані документа -->
    <title>Заголовок документа</title>
    <url>URL джерела</url>
    <date>Дата публікації</date>
    <!-- Інші метадані -->
  </metadata>
  <document>
    <!-- Текстові елементи -->
    <h>Заголовок</h>
    <p>Параграф тексту</p>
    <ul>
      <li>Елемент списку 1</li>
      <li>Елемент списку 2</li>
    </ul>
    <q>Цитата</q>
  </document>
  <!-- Додаткові секції для аналітичної інформації -->
  <warnings>
    <!-- Попередження -->
  </warnings>
  <errors>
    <!-- Помилки -->
  </errors>
</root>
```

Рисунок 4.6 – Приклад XML-документа (виконано самостійно)

Для відстеження помилок та попереджень, кожній виявленій проблемі присвоюється унікальний ідентифікатор (ID), що має префікс «w» для попереджень та «e» для помилок, після чого йде порядковий номер. Цей ідентифікатор зберігається як в елементі з проблемою (через атрибут id), так і у відповідній секції «warnings» або «errors» (див. рисунок 4.7).

```
<p id="w1, e2">Текст з проблемами</p>
<warnings>
  <warning id="w1">Опис попередження</warning>
</warnings>
<errors>
  <error id="e2">Опис помилки</error>
</errors>
```

Рисунок 4.7 – Приклад введення помилок (виконано самостійно)

Додавання ідентифікатора помилок виконується при оновленні елементів у функції update_elements (див рисунок 4.8):

```
2 usages (2 dynamic)  horielov.d
def update_elements(self, document, update_fn):
    """
    Iterate through the document and update text in elements, including metadata.
    """
    meta_element = document.find("metadata")
    if meta_element is not None:
        for element in meta_element.iter():
            if element.text and element.tag in {"title"}:...

    doc_element = document.find("document")
    if doc_element is not None:
        for element in doc_element.iter():
            if element.text and element.tag in {"p", "h", "q", "li"}:
                updated_text, id = update_fn(element.text)
                element.text = updated_text
                if id:
                    old_id = element.get("id")
                    if old_id:
                        element.set("id", f"{old_id}, {id}")
                    else:
                        element.set("id", id)

    return document
```

Рисунок 4.8 – Функція оновлення елементів документа (виконано самостійно)

Цей метод перебирає всі текстові елементи документа (включаючи вибрані елементи метаданих, як-от заголовок) та застосовує до них функцію оновлення update_fn. Ця функція повертає оновлений текст та опціональний ідентифікатор (ID). Якщо ID було повернуто, він додається до атрибута id елемента. Якщо елемент вже має ID (від попередніх етапів обробки), нові ID додаються через кому.

4.2.3 Взаємодія анотатора з конвеєром обробки

Анотатор є значним компонентом системи обробки текстів, оскільки він забезпечує єдиний інтерфейс для роботи з документами на всіх етапах конвеєра. Клас AbstractPipeline використовує анотатор для:

- створення документів на етапі парсингу через метод parse класу AbstractParser, який отримує анотатор як аргумент і використовує його методи для створення структурованого документа;
- нормалізації тексту через метод _apply_normalization, який використовує update_elements анотатора для застосування нормалізації до кожного текстового елемента;
- оцінки та виправлення через метод _apply_evaluation_correction, який також використовує update_elements для ітеративного процесу оцінки та виправлення;
- токенізації через метод _tokenize_document, який викликає split_elements анотатора для розбиття тексту на речення та токени;
- збереження результатів на кожному етапі через метод get_string анотатора, який перетворює документ у рядкове представлення;

4.3 Отримання та парсинг даних з джерел

Важливим компонентом системи автоматизованого формування текстових корпусів є механізми отримання даних із різноманітних джерел та їх парсингу. Для отримання даних новин було розроблено абстрактні класи AbstractSource та AbstractRecord. Для очищення даних та отримання тексту новини було розроблено AbstractParser. Також було розроблено їх конкретні реалізації для роботи з новинним порталом «Суспільне Новини».

4.3.1 Структура класу Record

Клас AbstractRecord є базовим для представлення окремого запису (документа) з джерела даних. Він інкапсулює метадані запису та забезпечує

доступ до його вмісту. Метадані записуються в структуру Metadata, що містить наступні поля:

– заголовок документа, title;
– назва джерела, source;
– автор документа, author;
– мова документа, language;
– час публікації, publication_time;
– посилання на оригінал, reference.

Основні методи класу AbstractRecord:

– __init__(metadata, link), що ініціалізує запис з метаданими та посиланням на джерело;
– content(), що отримує вміст запису.

Для роботи з порталом «Суспільне» реалізовано клас SuspilneRecord, який наслідує AbstractRecord та додає специфічну логіку отримання вмісту. Цей метод завантажує html вміст статті за посиланням, якщо він ще не був завантажений, та додатково викликає метод fetch_metadata для отримання метаданих, яких може бракувати (наприклад, автора статті).

4.3.2 Реалізація джерела для роботи з «Суспільне Новини»

Клас SuspilneSource реалізує інтерфейс AbstractSource і забезпечує доступ до новин з порталу «Суспільне Новини». Цей клас має два ключові методи:

– name(), що повертає назву джерела;
– records(start, end), що повертає ітератор записів за вказаний період.

Метод records відповідає за отримання списку статей з архіву новин за вказаний період (див. рисунок 4.9). Метод працює наступним чином:

– ітерує через всі дати у заданому діапазоні;
– формує URL архівної сторінки для кожної дати;
– завантажує HTML-сторінку та парсить її за допомогою BeautifulSoup;
– знаходить всі статті на сторінці за класом c-article-card--small-headline;
– для кожної статті видобуває заголовок, посилання та час публікації;

– створює об'єкт SuspilneRecord з отриманими метаданими;
– повертає створений запис через генератор (yield).

```
1 usage (1 dynamic)  horielov.d
@staticmethod
def records(start: datetime, end: datetime):
    for date in daterange(start, end):
        archive_url = f"https://suspilne.media/archive/{date.year}/{date.month}/{date.day}/"
        print(f"Processing: {archive_url}")

        request = requests.get(archive_url)
        request.raise_for_status()

        soup = BeautifulSoup(request.text, features: "html.parser")
        articles = soup.find_all( name: "a", class_="c-article-card--small-headline")
        for article in articles:
            title = article.find("span", class_="c-article-card__headline-inner").text.strip()
            link = article["href"]
            publication_time = article.find("time")["datetime"]

            yield SuspilneRecord(
                metadata=Metadata(
                    title=title,
                    source=SuspilneSource.name(),
                    author=None,
                    language="uk",
                    publication_time=publication_time,
                    reference=link,
                ),
                link=link,
            )
```

Рисунок 4.9 – Функція отримання статей з архіву (виконано самостійно)

Використання генератора дозволяє ефективно обробляти великі обсяги даних, оскільки дозволяє отримувати записи по одному, без необхідності завантаження всіх даних у пам’ять.

4.3.3 Парсинг HTML-сторінок та вилучення структурованого тексту

Парсинг HTML-сторінок для порталу «Суспільне Новини» реалізовано через клас SuspilneParser, що відповідає за вилучення структурованого тексту та збереження його семантики. Цей клас має головний метод parse, що обробляє HTML-контент в декілька етапів:

– створює об'єкт BeautifulSoup для парсингу HTML;
– знаходить основний контейнер контенту div.l-article-content__container-inner.c-art-c__c;
– створює порожній документ через анотатора;
– обробляє вміст контейнера через метод _process_content_elements.

Основна логіка парсингу зосереджена в методі _process_content_elements, що аналізує всі дочірні елементи контейнера:

– розгортає вікна span.c-explainer__word, для отримання тексту з пояснень;
– декодує захищенні CloudFlare електронні адреси __cf_email__, задля уникнення тексту «» (див. рисунок 4.10);
– обробка структурних елементів (h1-h6, p, blockquote, div та ін.) та формування документа;
– очищення тексту за допомогою методу _clean_text з використанням html.unescape та перетворення захищених електронних адрес, якщо такі залишились, в текст «protected@mail.com»;
– вилучення тексту зі структурного елементу з додаванням пробілів після елементів br, p, div та li.

Цей парсер ефективно вилучає структурований текст зі сторінок «Суспільне Новини», зберігаючи його семантику та забезпечуючи правильне форматування.

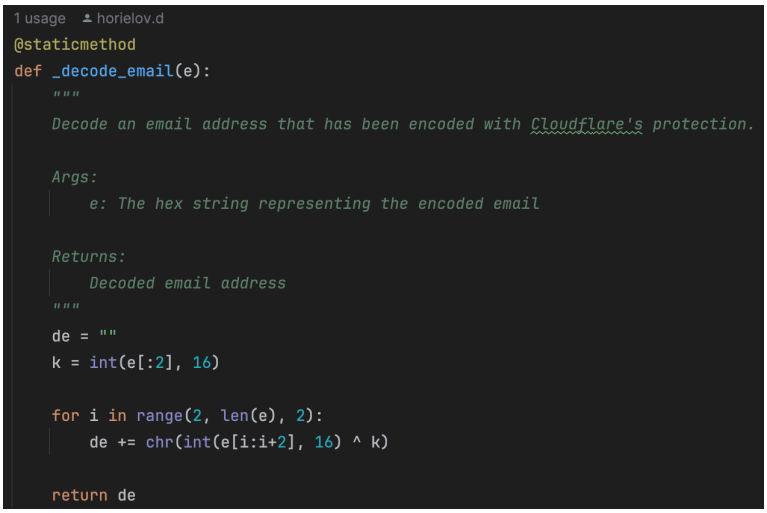


Рисунок 4.10 – Декодування захищених електронних адрес (виконано самостійно)

4.4 Реалізація нормалізації текстів

У контексті розробленої системи було реалізовано спеціалізований клас NewsNormalizer, що об'єднує декілька нормалізаторів для обробки новинних текстів. Розглянемо детально реалізацію основних нормалізаторів.

4.4.1 Архітектура компонента нормалізації

Архітектура компонента нормалізації заснована на абстрактному класі AbstractNormalizer, що визначає загальний інтерфейс для всіх нормалізаторів. Цей клас вимагає реалізації двох методів:

- name(), що повертає назву нормалізатора;
- normalize(text), що виконує нормалізацію тексту та повертає кортеж з трьох елементів: нормалізований текст, список попереджень та список помилок.

Клас NewsNormalizer виступає композитним нормалізатором, який послідовно застосовує спеціалізовані нормалізатори в такому порядку:

- виконує нормалізацію пробілів, WhiteSpaceNormalizer;
- забезпечує стандартизацію апострофів, ApostropheNormalizer;
- уніфікує лапки, QuotationMarksNormalizer;
- стандартизує формати телефонних номерів, UkrainianPhoneNormalizer.

Така послідовність обробки забезпечує коректну нормалізацію тексту, де кожен етап підготовлює текст для наступного. Метод normalize класу NewsNormalizer почергово викликає всі підпорядковані нормалізатори та акумулює попередження й помилки, виявлені на кожному етапі.

4.4.2 Алгоритм нормалізації телефонних номерів

Нормалізація телефонних номерів спрямована на стандартизацію різноманітних форматів до єдиного вигляду: +380 (XX) XXX-XX-XX для звичайних номерів та +380 (XXX) XX-XX-XX для спеціальних номерів. Алгоритм реалізовано у класі UkrainianPhoneNormalizer.

Для ефективної роботи з різними форматами номерів їх було класифіковано на дві категорії:

- номери зі звичайними кодами операторів (двозначні);
- номери зі спеціальними кодами (тризначні, наприклад, 800 та 900).

Для кожної категорії розроблено набір регулярних виразів, що охоплюють варіації запису. Для звичайних номерів виділено наступні групи шаблонів (де «A» – цифри коду оператора, «X» – інші цифри номера, а «_» – роздільник):

– формат AAA_XXX_XX_XX (напр., «099 123 45 67»);
– формат AAA_XX_XX_XXX (напр., «099 12 34 567»);
– формат AAA_XX_XXX_XX (напр., «099 12 345 67»);
– формат AAAX_XX_XX_XX (напр., «0991 23 45 67»);
– формат AAAXX_X_XX_XX (напр., «09912 3 45 67»).

Для спеціальних номерів виділено шаблони:

– формат AAA_XXX_XXX (напр., «800 123 456»);
– формат AAA_XX_XX_XX (напр., «800 12 34 56»).

У межах цих категорій є різновиди написання коду оператора, роздільників тощо. На основі цього опису сформовано 24 регулярних виразів, наприклад (див. рисунок 4.11):

```
# AA_XX_XX_XXX
# +380_(99)_12_34_567
r"\+?380\s?\(?(\d{2})\)?[\s-]?(\d{1})(\d{1})[\s-]?(\d{1})(\d{1})[\s-]?(\d{1})(\d{1})(\d{1})",
# +38_(099)_12_34_567
r"\+?38\s?\(?0(\d{2})\)?[\s-]?(\d{1})(\d{1})[\s-]?(\d{1})(\d{1})[\s-]?(\d{1})(\d{1})(\d{1})",
# 0_(99)_12_34_567
r"0[\s-]?(\d{2})[\s-]?(\d{1})(\d{1})[\s-]?(\d{1})(\d{1})[\s-]?(\d{1})(\d{1})(\d{1})",
# (099)_12_34_567
r"\(0(\d{2})\)[\s-]?(\d{1})(\d{1})[\s-]?(\d{1})(\d{1})[\s-]?(\d{1})(\d{1})(\d{1})",
# 0 (99)_12_34_567
r"0[\s-]?\((\d{2})\)[\s-]?(\d{1})(\d{1})[\s-]?(\d{1})(\d{1})[\s-]?(\d{1})(\d{1})(\d{1})",
```

Рисунок 4.11 – Приклад регулярних виразів для шаблону AA_XX_XX_XXX (виконано самостійно)

Алгоритм нормалізації телефонних номерів складається з наступних кроків:

– виявлення спеціальних номерів через метод normalize_special_phone_numbers, що застосовує регулярні вирази для пошуку номерів з кодами 800 та 900;
– нормалізація звичайних номерів через метод normalize_regular_phone_numbers, що обробляє номери з двозначними кодами операторів;

– фільтрація кандидатів через метод should_not_match, який перевіряє, чи відповідає код нормам України та, чи не є номер частиною більшого числового значення;
– форматування виявлених номерів до стандартного вигляду через методи format_regular_phone_number та format_special_phone_numbe.

4.4.3 Алгоритм нормалізації апострофів

Нормалізація апострофів розв’язує проблему варіативності символів, що використовуються на позначення апострофа, а також розрізняє апострофи та одинарні лапки. Алгоритм реалізовано у класі ApostropheNormalizer.

Для нормалізації визначено набір символів, що можуть виступати як апострофи:

– символ U+0027 (APOSTROPHE);
– символ U+02B9 (MODIFIER LETTER PRIME);
– символ U+02BB (MODIFIER LETTER TURNED COMMA);
– символ U+02BC (MODIFIER LETTER APOSTROPHE);
– символ U+2018 (LEFT SINGLE QUOTATION MARK);
– символ U+2019 (RIGHT SINGLE QUOTATION MARK);
– символ U+0060 (GRAVE ACCENT).

Алгоритм нормалізації апострофів складається з наступних кроків:

– послідовності з двох однакових символів апострофа замінюються на символ лапок;
– кожен символ апострофа аналізується на основі оточуючих його символів, з підтримкою альтернативних версій заміни (див. рисунок 4.12);
– якщо після заміни апострофів на лапки їх кількість стає непарною, використовується альтернативна версія тексту.

Усі набори символів, що використовуються при нормалізації зафіксовані в окремому класі Constants.

```
for symbol in Constants.APOSTROPHES:
    indices = [m.start() for m in re.finditer(symbol, text)]

    for index in indices:
        if index == 0 or index == len(text) - 1:
            text_list[index] = quote
            text_list_without_space_quote[index] = quote
        elif text[index - 1].isalpha() and text[index + 1].isalpha():
            text_list[index] = apostrophe
            text_list_without_space_quote[index] = apostrophe
        elif text[index - 1].isalpha() or text[index + 1].isalpha():
            text_list[index] = quote
            text_list_without_space_quote[index] = apostrophe
        elif text[index - 1] in Constants.PUNCTUATION:
            text_list[index] = quote
            text_list_without_space_quote[index] = quote
        elif text[index + 1] in Constants.PUNCTUATION:
            text_list[index] = quote
            text_list_without_space_quote[index] = quote
        else:
            warnings.append(f"Warning: {symbol} at position {index}")
```

Рисунок 4.12 – Контекстна заміна апострофів (виконано самостійно)

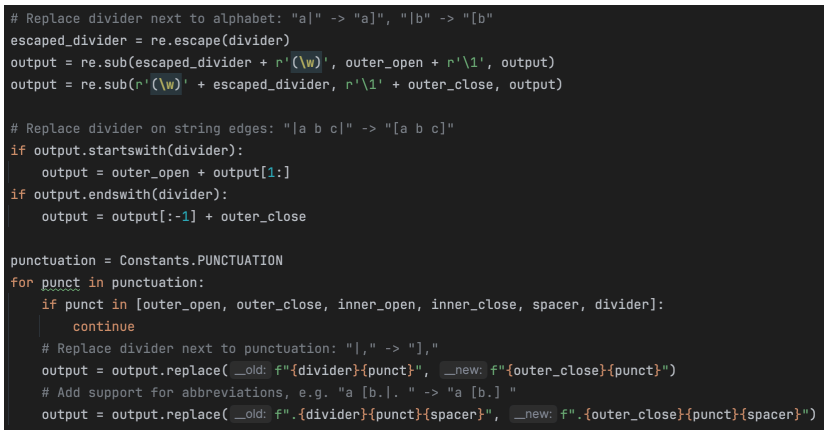
Якщо було виявлено символ в невідомій позиції, генерується попередження.

4.4.4 Процес нормалізації лапок

Нормалізація лапок забезпечує стандартизацію їх представлення відповідно до українських типографських норм, з правильною обробкою вкладених цитат. Алгоритм реалізовано у класі QuotationMarksNormalizer.

Алгоритм складається з трьох послідовних етапів:

– різні типи лапок замінюються на єдиний стандартний символ для спрощення подальшого аналізу;
– стандартизований символ лапок замінюється на відповідні відкриті або закриті лапки залежно від контексту (див. рисунок 4.13). Якщо після виконання залишились стандартні лапки, то генерується помилка та повертається початковий рядок;
– встановлення чергування стилів лапок відповідно до правопису. Якщо кількість лапок непарна, генерується помилка та чергування не відбувається.



```
# Replace divider next to alphabet: "a|" -> "a]", "|b" -> "[b"
escaped_divider = re.escape(divider)
output = re.sub(escaped_divider + r'(\w)', outer_open + r'\1', output)
output = re.sub(r'(\w)' + escaped_divider, r'\1' + outer_close, output)

# Replace divider on string edges: "|a b c|" -> "[a b c]"
if output.startswith(divider):
    output = outer_open + output[1:]
if output.endswith(divider):
    output = output[:-1] + outer_close

punctuation = Constants.PUNCTUATION
for punct in punctuation:
    if punct in [outer_open, outer_close, inner_open, inner_close, spacer, divider]:
        continue
    # Replace divider next to punctuation: "|," -> "],"
    output = output.replace( _old: f"{divider}{punct}", _new: f"{outer_close}{punct}")
    # Add support for abbreviations, e.g. "a [b.|. " -> "a [b.] "
    output = output.replace( _old: f".{divider}{punct}{spacer}", _new: f".{outer_close}{punct}{spacer}")
```

Рисунок 4.13 – Приклад контекстної заміни лапок (виконано самостійно)

Загалом описано тринадцять правил для контекстної заміни, хоча на рисунку 4.13 видно лише шість.

4.5 Реалізація токенізації

Для створення високоякісного текстового корпусу необхідно реалізувати токенізатор, який коректно обробляє семантично цілісні конструкції, зберігаючи їх як єдині токени.

4.5.1 Оцінка наявних токенізаторів

Для визначення оптимальної стратегії розробки токенізатора було проведено оцінку наявних рішень. Сформовано тестовий набір з 26 складних випадків токенізації, що тестує:

– слова з дефісом (будь-який, бозна-що, 2022-му);
– порядкові номери (№11, №3,4, №1-4);
– дату та час (18.05.2021, 22:37);
– числові діапазони (15-21);
– електронні адреси (rename@city.kharkiv.ua);
– телефонні номери (+38 (095) 568 38 77);
– грошові суми ($1,461);
– температурні показники (-21°C);

– одиниці виміру (м/с, м², км²/місяць);
– скорочення (м., вул., пл., ред.);
– слова з апострофом (під'їздів);
– числа з розділювачами (234,5, 1 000 000);
– приблизні значення (~100).

Оцінимо відсоток збереження токенів як один, без розділення (див. рисунок 4.14).

```
Table saved to tokenization_results.csv
Results:
lang_tokenizer: 9/26 (34.62%)
spacy: 14/26 (53.85%)
nltk: 18/26 (69.23%)
language_tool: 13/26 (50.00%)
```

Рисунок 4.14 – Результати тестування токенізаторів (виконано самостійно)

На основі порівняння ефективності різних підходів встановлено, що найкращі результати показав токенізатор NLTK (69.23%), однак він не має гнучкого інтерфейсу для модифікацій. Тому для подальшого вдосконалення обрано токенізатор SpaCy (53.85%), який надає розширені можливості для налаштування.

4.5.2 Архітектура модифікованого токенізатора

На основі аналізу наявних рішень розроблено клас CustomTokenizer, що наслідує AbstractTokenizer і використовує SpaCy як базовий компонент. Токенізатор SpaCy використовує трикомпонентний підхід до розбиття тексту на токени:

– послідовності на початку слів, що відокремлюються, тобто префікси;
– послідовності в кінці слів, що відокремлюються, тобто суфікси;
– послідовності всередині слів, які спричиняють розбиття, тобто інфікси;

Для покращення роботи токенізатора реалізовано модифікації для кожного з цих компонентів, а також додаткові механізми ретокенізації для особливих випадків. Скопіюємо стандартний код для створення об’єкта токенізатора для української мови, однак зробимо наступні зміни:

- додамо підтримку для валют та символу «№» для префіксів. Для підтримки валют у масиві «TOKENIZER_PREFIXES» (далі – префікси) закоментуємо елемент «LIST_CURRENCY». Для підтримки символу номера підмінимо список «LIST_ICONS» (далі – список спеціальних символів) прибравши з нього код символу номера – «\u2116»;
- додамо підтримку для валют, температур та одиниць виміру для суфіксів. Для підтримки температур підмінимо список спеціальних символів прибравши з нього додатково код символу градусів (\u00B0) та підмінивши регулярний вираз розділення температури на «(?<=°)\.|(?<=°[FfCcKk])\.». Для підтримки валют та одиниць виміру закоментуємо регулярні вирази що форматують CURRENCY та UNITS;
- Підмінено список спеціальних символів для послідовності з попередніми кроками. Для підтримки дефісів між буквами замінимо регулярний вираз «(?<=[0-9])[+\-*^](?=[0-9-])» на два інших: «(?<=[0-9])[+*^] (?=[0-9-])» та «(?<=[0-9])-(?=-)». Для підтримки слешу замінено вираз «(?<=[{a}0-9])[:<>=/](?=[{a}])» на «(?<=[{a}0-9])[:<>=](?=[{a}])». Для підтримки стандартного дефіса між цифрами підмінено список «HYPHENS» прибравши з нього символ «\u002D». Також підмінено список «HYPHENS» для масиву комбінованих інфіксів, що також використовується;
- Для підтримки скорочень оновимо список винятків через add_special_case. Видалимо надлишкові абревіатури які токенізатор вже оброблює як один символ. Також виправимо помилку коли токен розбиває абревіатури на підтокени, що складаються з менших абревіатур, наприклад: «пров. арк.» (коли додано окремо «пров.» та «арк.»);

– Додатково реалізуємо ретокенізацію тексту для збігів зі списку регулярних виразів з уникненням конфліктів через збереження найдовшого збігу (filter_spans з spacy.util). Додамо підтримку чисел, розряди якого поділені пробілами за допомогою регулярного виразу «(?<!\d)\b\d{1,3}(?: \d{3})+\b(?! \d)» та номерів телефону: «\+?\b(?:\d{1,3})?0?[\s-]?\(?\d{2,3}\)?[\s-]?\d{2,3}[\s-]?\d{2,3}[\s-]?\d{2,3}\b». Регулярний вираз для номера телефону може бути підмінений на його нормалізовану версію.

4.5.3 Підтримка поділу на речення

Клас CustomTokenizer підтримує два режими роботи: поділ на слова (words=True) та поділ на речення (words=False).

Для підтримки токенізації речень на основі оновленого токенізатора потрібно виконати ініціалізацію інакше, ніж це зроблено для поділу на токени (див. рисунок 4.15).

```
horielov.d
def __init__(self, words: bool = True):

    # Download model if not already downloaded
    try:
        spacy.load("uk_core_news_sm")
    except OSError:
        os.system("python -m spacy download uk_core_news_sm")
    self.words = words

    if self.words:
        self.nlp = spacy.load( name: "uk_core_news_sm", enable=["parser"])
    else:
        self.nlp = spacy.load( name: "uk_core_news_sm", exclude=["parser"])
        self.nlp.enable_pipe("senter")

    self.EXCEPTIONS_WITH_SPACES = []
    self.EXCEPTIONS = []
    self.nlp.tokenizer = self.setup_tokenizer(self.nlp)
```

Рисунок 4.15 – Ініціалізація класу CustomTokenizer (виконано самостійно)

Код ініціалізації виокремлюється через особливості реалізації бібліотеки та порядок використання компонентів. Інший код ініціалізації залишається спільним.

4.6 Інші компоненти системи

Окрім вже описаних детально компонентів, система автоматизованого формування текстових корпусів включає ряд інших компонентів, що забезпечують повноцінне функціонування конвеєра обробки текстів.

4.6.1 Компонент збереження результатів

Для збереження результатів обробки на кожному етапі конвеєра реалізовано клас FileSaver, що успадковує абстрактний інтерфейс AbstractSaver. Цей компонент відповідає за запис документів у файлову систему та створення тек (див. рисунок 4.16).

```
2 usages  horielov.d
class FileSaver(AbstractSaver):
    horielov.d
    def __init__(self, path: str, create_dir: bool = True):
        self.path = path

        if create_dir and not os.path.exists(path):
            os.makedirs(path)

    horielov.d
    @staticmethod
    def name() -> str:
        return "File Saver"

    4 usages (4 dynamic)  horielov.d
    def save(self, text: str, name: str):
        with open(f"{self.path}/{name}", "w") as file:
            file.write(text)
```

Рисунок 4.16 – Компонент збереження файлів (виконано самостійно)

4.6.2 Компоненти оцінки та виправлення тексту

Для оцінки якості тексту та виявлення потенційних проблем реалізовано два компоненти:

– компонент для аналізу тексту за допомогою інструменту LanguageTool, з назвою LanguageToolEvaluator. Компонент використовує бібліотеку language_tool_python для інтеграції, що забезпечує перевірку орфографії

та граматики українських текстів. Реалізація через менеджера контексту забезпечує коректну ініціалізацію та звільнення ресурсів LanguageTool;

– заглушка для компонента виправлення, що не вносить змін у текст, з назвою NoChangesCorrector. Цей компонент служить заглушкою для підсистеми виправлення тексту, яка в майбутніх версіях системи може бути замінена функціональнішим компонентом, наприклад, на основі машинного навчання або нейронних мереж.

4.6.3 Утиліти для роботи з датами та файлами

Система включає ряд утиліт, що спрощують типові операції:

– утиліта для генерації послідовності дат у заданому діапазоні, з назвою daterange. Ця функція використовується для ітерації через діапазон дат при отриманні новин за певний період.;
– утиліта для нормалізації імен файлів, з назвою sanitize_and_transliterate. Ця функція виконує транслітерацію українських символів у латиницю та очищення імен файлів від недопустимих символів, що забезпечує коректне збереження файлів у різних файлових системах.

5 ЕКСПЕРИМЕНТАЛЬНА ОЦІНКА СИСТЕМИ

У цьому розділі представлені результати комплексної оцінки розробленої системи автоматизованого формування текстових корпусів.

5.1 Оцінка ефективності нормалізації

Для оцінки ефективності розроблених алгоритмів нормалізації українських текстів було використано корпус UberText 2.0, зокрема його підкорпус «News Cleaned». Цей корпус обсягом 21.37 ГБ містить колекцію українських новинних статей з різних онлайн-джерел, що забезпечує різноманітні приклади форматування, які потребують нормалізації.

Структура корпусу складається з новинних статей, розділених порожніми рядками, причому текст кожної статті розподілений на кілька рядків, розділених символами CR та LF. Обробка корпусу здійснювалася шляхом ітеративного накопичення рядків до досягнення роздільників, після чого кожен накопичений блок обробляється як окрема новинна стаття.

5.1.1 Нормалізація телефонних номерів

Для виявлення та аналізу різноманітності форматів телефонних номерів у корпусі було написано функцію для отримання шаблону номера зі збігу на реальному номері. Механізм замінював конкретні цифри на шаблонні символи, зберігаючи при цьому форматування. Наприклад, справжні номери телефонів перетворювалися на шаблони на кшталт «+380 (XXX) XXX-XX-XX» або «(0XXX) XX-XX-XX». При обробці корпусу за допомогою наборів регулярних виразів з описаного класу UkrainianPhoneNormalizer для кожного виявленого телефонного номера фіксувалися: знайдений номер, його шаблонне представлення, результат нормалізації.

Аналіз виявив 302 різних формати телефонних номерів у корпусі UberText 2.0, що демонструє значну неоднорідність представлення телефонних номерів у українських новинних текстах. Найпоширеніші формати представлені в таблиці 5.1.

Таблиця 5.1 – Найпоширеніші телефони формати (виконано самостійно)

№	Формат	Кількість збігів	Відсоток від загальної кількості
1	(0XX) XXX-XX-XX	11827	19.258451
2	0XX-XXX-XX-XX	6407	10.432814
3	0XX XXX XX XX	4964	8.083111
4	0XXXXXXXXX	4232	6.891161
5	+380 (XXX) XX-XX-XX	4047	6.589917
6	(0XX) XXX XX XX	3443	5.606396
7	(0XXX) XX-XX-XX	2474	4.028529
8	+38 0XX XXX XX XX	1751	2.851234
9	+380XXXXXXXXX	1723	2.805641
10	0 XXX XXX XXX	1616	2.631408

Решта, а саме 292 формати складали приблизно 30.8% усіх випадків, при цьому 73 формати зустрічались лише один раз у всьому корпусі.

Після застосування розробленого алгоритму нормалізації всі 302 різні формати телефонних номерів були успішно приведені до двох стандартизованих форматів (див. табл. 5.2).

Таблиця 5.2 – Телефоні формати після нормалізації (виконано самостійно)

№	Формат	Кількість збігів	Відсоток від загальної кількості
1	+380 (XX) XXX-XX-XX	51489	83.84192
2	+380 (XXX) XX-XX-XX	9923	16.15808

Алгоритм нормалізації ефективно стандартизував усі українські номери згідно з визначеними в методології вимогами.

5.1.2 Нормалізація апострофів

Аналіз використання апострофів у корпусі виявив значну непослідовність у використанні символів апострофа. У процесі попередньої обробки було виявлено сім різних Unicode-символів, які використовувалися як апострофи у текстах корпусу. Таблиця 5.3 показує розподіл символів апострофа до та після нормалізації.

Таблиця 5.3 – Розподіл апострофів при нормалізації (виконано самостійно)

Символ	До нормалізації		Після нормалізації	
	Кількість	Відсоток	Кількість	Відсоток
U+0027	7465048	60.5074	5519	0.0448
U+2019	4693904	38.0461	1226	0.0100
U+02BC	144521	1.1714	12302968	99.9403
U+0060	20895	0.1694	397	0.0032
U+2018	12824	0.1039	209	0.0017
U+02B9	214	0.0017	0	0
U+02BB	10	0.0001	0	0
Загалом	12337416	100.0000	12310319	100.0000

Найбільш поширеними символами апострофа в корпусі були U+0027 та U+2019, що складали 60.51% та 38.05% від усіх випадків відповідно. Цільовий стандартний символ U+02BC становив лише 1.17% початкових апострофів.

Після застосування алгоритму нормалізації апострофів 99.94% апострофів були успішно перетворені на U+02BC, і лише 0.06% символів залишилися в оригінальній формі. Близько 0.22% початкових символів апострофа були ідентифіковані як знаки лапок і нормалізовані відповідно.

Алгоритм нормалізації згенерував 7372 попередження про апострофи в нестандартному положені, що відповідають 7351 апострофу (0.06% від загальної кількості), які залишилися без змін. Крім того, у 10733 випадках апострофи, які спочатку були конвертовані в лапки, були повернуті назад через механізми розв'язання неоднозначності.

5.1.3 Нормалізація лапок

Нормалізація лапок проводилася відразу після нормалізації апострофів, оскільки алгоритм нормалізації апострофів створює додаткові лапки, які також потребують обробки.

Аналіз використання лапок виявив ще більшу непослідовність, ніж у випадку з апострофами. Таблиця 5.4 представляє розподіл до та після нормалізації.

Таблиця 5.4 – Розподіл лапок при нормалізації (виконано самостійно)

Символ	До нормалізації		Після нормалізації	
	Кількість	Відсоток	Кількість	Відсоток
U+0022	29002764	51.743	7780	0.014
U+00AB	12495485	22.293	26895177	47.961
U+00BB	12333176	22.003	26840011	47.863
U+201D	1065203	1.900	1166659	2.080
U+201C	1046085	1.866	1167062	2.081
U+201E	109199	0.195	0	0.000
U+201F	16	0.000	0	0.000
U+275D	4	0.000	0	0.000
U+275E	4	0.000	0	0.000
Загалом	56051936	100.000	56076689	100.000

Найпоширенішими лапками були U+0022 (QUOTATION MARK), які становили 51.74% від усіх випадків. Лапки-ялинки (U+00AB та U+00BB) разом складали 44.30%, а «лапки-лапки» (U+201C та U+201D) представляли лише 3.77% від загальної кількості.

Після нормалізації лапки-ялинки стали переважною формою для зовнішніх лапок (95.82%), а лапки-лапки послідовно використовуються для вкладених лапок (4.16%). Корекція балансу лапок значно покращилася: для не вкладених лапок баланс покращився у 6.40 разів, а для вкладених лапок – 51.96 разів.

Під час нормалізації було згенеровано 351 помилку (приблизно 0.0006% від загальної кількості лапок після нормалізації). Ці помилки були пов'язані з неможливістю визначити, чи повинен символ лапок бути заміненим на відкритий чи закритий. Також було згенеровано 505085 попереджень (приблизно 0.9% від загальної кількості лапок після нормалізації). Ці попередження були пов'язані з незбалансованими лапками в оригінальному тексті, де новинні статті часто опускають закриті лапки або використовують непослідовні символи.

5.2 Оцінка токенізації

Для перевірки ефективності розробленого токенізатора було створено тестовий текст з уривків новинних текстів (Додаток А). На основі вимог, описаних у розділі 3.2, підготовлено еталонну токенізацію цього тексту (Додаток

Б). Порівняльний аналіз результатів роботи розробленого токенізатора з еталонною розміткою показав повний збіг усіх 507 токенів, що підтверджує коректність роботи алгоритму та його відповідність встановленим вимогам.

5.3 Оцінка системи в цілому

Для комплексної оцінки функціональності розробленої системи 1 квітня 2025 року було проведено її повноцінне тестування на реальних даних. Система успішно обробила 413 новинних текстів, опублікованих на порталі «Суспільне Новини» за 31 березня 2025 року. У результаті роботи системи було сформовано структурований корпус, що містить 413 текстових документів для кожного з чотирьох етапів обробки (загалом 1652 файли). Всі документи збережено в теці «corpus» з відповідною організацією по піддиректоріях:

– base, з текстами після первинного парсингу з вебсторінок, зі збереженням структури та метаданих;
– normalized, з текстами після застосування алгоритмів нормалізації;
– evaluated, з текстами з маркуванням виявлених помилок та попереджень;
– tokenized, з текстами з додатковою розбивкою на речення та токени.

Приклад структури сформованого корпусу та зміст одного з токенізованих документів представлено на рисунку 5.1.

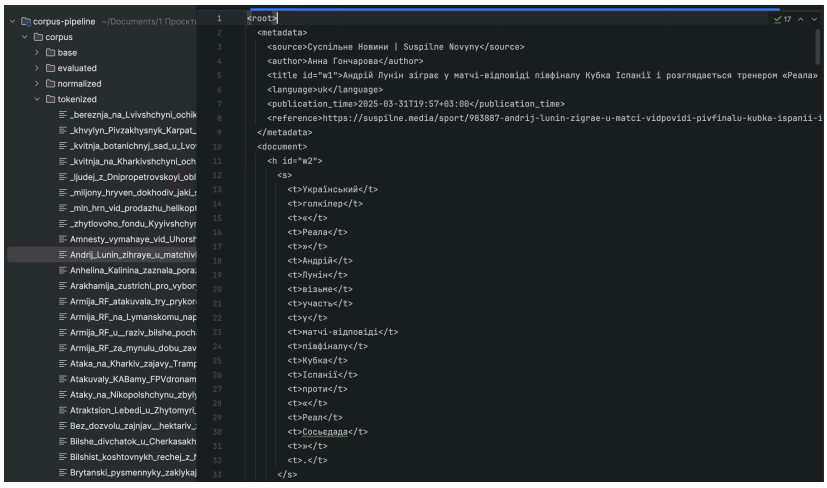


Рисунок 5.1 – Приклад оброблених документів (виконано самостійно)

Час обробки одного документа триває від 3 до 10 секунд, причому найбільш ресурсомістким етапом виявилася оцінка помилок за допомогою LanguageTool. Така тривалість обумовлена особливістю роботи компонента оцінки, який обробляє кожне речення окремо. Попри це, загальна продуктивність системи залишається достатньою для щоденної обробки новинного потоку.

Аналіз результатів роботи системи на реальних даних підтвердив ефективність розроблених алгоритмів нормалізації та токенізації в контексті повного конвеєра обробки тексту. Сформований корпус текстів відповідає всім визначеним вимогам до структури та якості даних, що забезпечує його придатність для подальшого використання в лінгвістичних дослідженнях та розробці моделей обробки української мови.

5 НАПРЯМКИ ПОДАЛЬШОГО ВДОСКОНАЛЕННЯ СИСТЕМИ

Найсуттєвішою проблемою є затримка у використанні LanguageTool, що є пріоритетним напрямком покращення системи. Проблема полягає в тому, що в бібліотеці використовується підняття сервера та запити до нього, а також кожен елемент (параграф) посилає окремі запити. Це реалізовано для відстеження помилок для кожного елемента окремо, однак можна оптимізувати процес, передаючи всю новину для перевірки одразу. Для цього необхідно реалізувати механізм відстеження помилок для кожного елемента або запропонувати альтернативну архітектуру компонента оцінки.

Можливі покращення стосуються також нормалізації телефонних номерів – можна як оптимізувати регулярні вирази, що містять багато повторень, так і додати підтримку іноземних номерів, які можуть зустрічатись в текстах новин, наприклад, номери посольств.

Одним із перспективних напрямків є реалізація компонента виправлення помилок, наприклад, на основі ChatGPT. Такий коректор зможе виправляти не тільки помилки, виявлені LanguageTool, а й проблеми нормалізації, такі як пропущені відкриті чи закриті лапки.

Також можна вдосконалити текст шляхом прибирання повторюваних елементів у корпусі, таких як «Читайте нас у Telegram: Суспільне Дніпро». Проте потрібно зважити доцільність цього фільтру, оскільки корпус може використовуватись для тренування моделей пошуку таких елементів. Потенційне рішення – додавання нового компонента експорту, який перевірятиме документи та експортуватиме їх відповідно до обраних критеріїв або модифікуватиме при експорті. Доцільно також додати в метадані показники розподілу мов, що може знадобитись при прийнятті рішення щодо обробки документів.

Розширення списку джерел для отримання текстів є логічним розвитком системи, однак важливо враховувати авторські права. «Суспільне Новини», наприклад, належить українському суспільству та фінансується з податків громадян, тому таких проблем немає, але для інших джерел потрібно отримати відповідні дозволи.

Можливо також розширити семантичну токенізацію, наприклад, для групування адрес чи ініціалів як єдиних токенів, що підвищить якість аналізу текстів.

Доцільно також інтегрувати додаткові етапи нормалізації, такі як пошук неукраїнських букв в текстах, шляхом інтеграції очищення тексту за допомогою Language-Tool, як це реалізовано в інших текстових корпусах. Однак потрібно оцінити вплив на продуктивність системи.

Автоматизація процесу отримання текстів дозволить не запускати його щодня вручну, а налаштувати автоматичне оновлення корпусу з певною періодичністю. Впровадження детального логування параметрів виконання кожного етапу, включно з часовими показниками нормалізації та оцінювання, допоможе у виявленні вузьких місць та плануванні оптимізацій.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проведено комплексне дослідження, спрямоване на підвищення якості україномовних текстових корпусів та систем для автоматичного їх формування шляхом розробки вдосконаленої САФТК з покращеними методами нормалізації та токенізації текстів.

Проведений аналіз наявних підходів до створення українських корпусів (БрУК, ГРАК, UberText 1.0 та 2.0) виявив критичні обмеження наявних рішень. Система БрУК застосовує підхід з високим ступенем участі людини, що забезпечує якість, але обмежує масштабованість. ГРАК використовує подібний підхід із залученням значної кількості експертів для валідації. UberText 1.0 представляв перший крок до автоматизації, але не зберігав структуру документів. Аналіз публічно доступного коду UberText 2.0 показав суттєві обмеження: відсутність коду для збору початкових даних та неможливість підтвердження збереження структури новинних текстів. За результатами аналізу було виявлено, що наразі немає гнучкого та масштабованого рішення для систем автоматичного формування текстових корпусів, що обґрунтувало необхідність розробки покращеної САФТК, що потребує детального дослідження усіх етапів формування корпусу: вилучення, обробка та збереження тексту.

Проведено комплексний аналіз методів вилучення тексту з різноманітних джерел: оптичне розпізнавання тексту (Tesseract, EasyOCR, Doctr, Keras-OCR), перетворення аудіо в текст (IBM Speech to Text, Microsoft Bing Voice Recognition, Google Cloud, OpenAI Whisper, Amazon Transcribe), роботу з відеоплатформами (YouTube з Pafy, PyTube) та парсинг вебконтенту (Beautiful Soup, Scrapy). На основі проведеного аналізу було обрано HTML-парсинг як найдоцільніший підхід для роботи з новинними ресурсами. Було успішно реалізовано механізми отримання та парсингу даних з вебджерел, що забезпечують витягування основного текстового контенту зі збереженням структури (заголовки, абзаци, цитати, списки) та ефективну фільтрацію стороннього вмісту.

Аналіз наявних підходів до нормалізації (системи ГРАК, робота Вакуленка, UberText 2.0, дослідження Cudak та ін.) показав, що жоден з них не надає

всеосяжного рішення для специфічних проблем нормалізації українських новинних текстів. На основі проведеного аналізу було сформовано комплексні вимоги до нормалізації українських текстів: нормалізація телефонних номерів згідно з дизайн-системою державних сайтів України (+380 (XX) XXX-XX-XX та +380 (XXX) XX-XX-XX), нормалізація лапок згідно з чинним правописом (використання «лапок-ялинок» для зовнішніх та “лапок-лапок” для внутрішніх цитат), нормалізація апострофів до стандарту U+02BC. Розроблені спеціалізовані алгоритми нормалізації успішно пройшли експериментальне тестування на матеріалах корпусу UberText 2.0: 302 різні формати телефонних номерів були зведені до двох стандартизованих форматів, 99.94% апострофів перетворено на цільовий стандарт, а лапки нормалізовано з правильним чергуванням зовнішніх і внутрішніх форм.

Проведено аналіз наявних токенізаторів для української мови (spaCy, Stanza, NLTK, LanguageTool, Lang-UK) та виявлено ключову проблему – відсутність збереження семантичної цілісності складених конструкцій при їх розбитті на окремі токени. Сформовано детальні вимоги до токенізації, що охоплюють збереження як єдиних токенів: слів з дефісом, числових діапазонів, дат і часу, порядкових номерів, електронних адрес, телефонних номерів, одиниць виміру, температурних показників, грошових сум, спеціальних скорочень, слів з апострофом, чисел з розділювачами та приблизних значень. Розроблений токенізатор демонструє 100% ефективність у збереженні семантичної цілісності спеціальних конструкцій для тестового набору через модифікацію трикомпонентної системи токенізації (префікси, суфікси, інфікси) та впровадження механізмів ретокенізації для обробки спеціальних випадків.

Проаналізовано сучасні інструменти автоматичного виявлення помилок (LanguageTool, Grammarly, ChatGPT) та виявлено їх обмеження у роботі з україномовними текстами, особливо щодо контекстного аналізу складних граматичних конструкцій. Реалізовано ітеративний процес оцінки та виправлення помилок з обмеженням на максимальну кількість ітерацій. Розроблено підхід

фіксації кожної проблеми з унікальним ідентифікатором та відповідним типом (попередження чи помилка), що забезпечує прозору діагностику якості тексту.

Запропоновано єдиний формат документа з поступовим розширенням структури на кожному етапі обробки: базовий формат (після парсингу з метаданими), нормалізований формат (з інформацією про проблеми форматування), оцінений формат (з інформацією про лінгвістичні помилки) та токенізований формат (з деталізацією на речення та токени). Такий підхід забезпечує послідовне збереження структури та семантики тексту, включаючи всі необхідні метадані та аналітичну інформацію, що дозволяє відстежувати процес обробки на кожному етапі.

На основі проведених досліджень розроблено комплексну САФТК, що успішно розв'язує виявлені проблеми попередніх рішень. Система реалізована з використанням принципів модульності та конвеєрної обробки, де кожен етап (отримання даних, парсинг, нормалізація, оцінка якості, токенізація, збереження результатів) реалізовано як окремий компонент з чітко визначеним інтерфейсом. Модульна архітектура забезпечує гнучкість, розширюваність та можливість незалежної модифікації окремих компонентів, що дозволяє легко адаптувати систему для роботи з різними джерелами даних та форматами документів.

Практичне тестування системи на матеріалах новинного порталу «Суспільне» підтвердило її ефективність у реальних умовах. Система успішно обробила 413 новинних текстів, забезпечуючи збереження структури документів, стандартизацію форматування та семантично коректну токенізацію. Сформований корпус текстів відповідає всім визначеним вимогам та демонструє значне покращення якості порівняно з наявними рішеннями: повну стандартизацію телефонних номерів та типографських елементів, 100% збереження семантичної цілісності спеціальних конструкцій при токенізації, прозору систему діагностики якості тексту.

Розроблена система автоматизованого формування текстових корпусів повністю вирішує поставлені завдання та забезпечує суттєве підвищення якості обробки українських текстів порівняно з наявними рішеннями. Важливим

внеском роботи є створення спеціалізованих алгоритмів нормалізації та токенізації, що враховують специфіку української мови та сучасні стандарти.

Практичне значення дослідження полягає у створенні універсального інструменту для формування високоякісних україномовних корпусів, що сприятиме розвитку мовних технологій для української мови та її повноцінній інтеграції в сучасні технологічні процеси. Модульна архітектура системи дозволяє її подальше розширення та адаптацію для різних типів текстових джерел та специфічних завдань корпусної лінгвістики.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Корпус сучасної української мови (БрУК). GitHub. URL: https://github.com/brown-uk (дата звернення: 03.04.2025).

2. UberText corpus. Lang-uk projects. URL: https://lang.org.ua/en/corpora/#anchor4 (дата звернення: 03.04.2025).

3. CHAPLYNSKYI, Dmytro. Introducing UberText 2.0: A corpus of modern Ukrainian at scale. In: Proceedings of the Second Ukrainian Natural Language Processing Workshop (UNLP). 2023. p. 1-10

4. Дубок В. Ю. Дослідження сервісів AWS для обробки природної мови / В. Ю. Дубок ; наук. керівник к. т. н., доцент Н. С. Кравець // Радіоелектроніка та молодь у XXI столітті : матеріали 28-го Міжнар. молодіж. форуму, 16–18 квітня 2024 р. – Харків : ХНУРЕ, 2024. – Т. 6 – С. 425-427. – DOI : https://doi.org/10.30837/IYF.IIS.2024.425.

5. Каук В. І. Перетини технологій: штучний інтелект, як каталізатор змін у UX/UI дизайні / В. І. Каук // Поліграфічні, мультимедійні та web-технології. Інновації та розвиток: монографія. – Харків: ТОВ «Друкарня Мадрид», 2024. – С. 226-242.

6. Орфографія та синтаксис. Дизайн система державних сайтів України. URL: https://design.gov.ua/ua/teksty-i-kontent/orfografiya-ta-sintaksis (дата звернення: 03.04.2025).

7. НАН України. Український правопис : довідк. вид. Київ : Вид-во "Наук. думка" НАН України, 2019. 392 с.

8. Новини України та світу. Суспільне | Новини. URL: https://suspilne.media (дата звернення: 03.04.2025).

9. Horielov D. Corpus pipeline. GitHub. URL: https://github.com/SoMWbRa/corpus-pipeline/tree/university (дата звернення: 09.06.2025).

10. ГО «Валентність. Переосмислення». Лінгвоцид. Linguicide. URL: https://linguicide.in.ua/practice/lesson (дата звернення: 01.12.2024).

11. English Corpora: most widely used online corpora. Billions of words of data:

free online access. English Corpora: most widely used online corpora. Billions of words of data: free online access. URL: https://www.english-corpora.org (дата звернення: 01.12.2024).

12. The british national corpus (BNC). [bnc] British National Corpus. URL: http://www.natcorp.ox.ac.uk (дата звернення: 23.05.2025).

13. Корпус текстів української мови. Лінгвістичний портал MOVA.info. URL: http://www.mova.info/corpus.aspx?l1=209 (дата звернення: 01.12.2024).

14. Генеральний регіонально анотований корпус української мови (ГРАК) / М. Шведова, Р. фон Вальденфельс, С. Яригін, А. Рисін, В. Старко, Т. Ніколаєнко та ін. — Київ, Львів, Єна, 2017–2025. — uacorpus.org.

15. Beautiful Soup Documentation – Beautiful Soup 4.12.0 documentation. Crummy: The Site. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (дата звернення: 01.12.2024).

16. GitHub - tesseract-ocr/tesseract: Tesseract Open Source OCR Engine (main repository). GitHub. URL: https://github.com/tesseract-ocr/tesseract (дата звернення: 23.05.2025).

17. PyPDF2 documentation. Welcome to PyPDF2. URL: https://pypdf2.readthedocs.io/en/3.x/ (дата звернення: 01.12.2024).

18. Як росіяни збирають інформацію про військовослужбовців з Полтавщини. Суспільне Полтава. URL: https://suspilne.media/poltava/923947-ak-rosiani-zbiraut-informaciu-pro-vijskovosluzbovciv-z-poltavsini/ (дата звернення: 03.04.2025).

19. Весняний ярмарок, день відкритих дверей і виготовлення окопних свічок: куди піти в Миколаєві вихідними. Суспільне Миколаїв. URL: https://suspilne.media/mykolaiv/959899-vesnanij-armarok-den-vidkritih-dverej-i-vigotovlenna-okopnih-svicok-kudi-piti-v-mikolaevi-vihidnimi/ (дата звернення: 03.04.2025).

20. Найбільші країни ЄС не підтримують пропозицію виділити 20 млрд пакет допомоги Україні – топдипломат ЄС. Суспільне Новини. URL: https://suspilne.media/964001-veliki-kraini-es-ne-pidtrimuut-propoziciu-vidiliti-20-

mlrd-paket-dopomogi-ukraini-topdiplomat-es/ (дата звернення: 03.04.2025).

21. Новий прем’єр-міністр Канади складе присягу 14 березня. Суспільне Новини. URL: https://suspilne.media/969281-novij-premer-ministr-kanadi-sklade-prisagu-14-berezna/ (дата звернення: 03.04.2025).

22. UDPipe is a trainable pipeline for tokenization, tagging, lemmatization and dependency parsing of CoNLL-U files.. LINDAT/CLARIAH-CZ. URL: https://lindat.mff.cuni.cz/services/udpipe/ (дата звернення: 01.12.2024).

23. Natural language toolkit. NLTK. URL: https://www.nltk.org (дата звернення: 01.12.2024).

24. SpaCy · industrial-strength natural language processing in python. spaCy. URL: https://spacy.io (дата звернення: 01.12.2024).

25. GitHub - amakukha/stemmers_ukrainian: A novel stemmer for the Ukrainian language trained with AI. GitHub. URL: https://github.com/amakukha/stemmers_ukrainian (дата звернення: 01.12.2024).

26. CoNLL-U format. Universal Dependencies. URL: https://universaldependencies.org/format.html (дата звернення: 01.12.2024).

27. Text encoding initiative. TEI. URL: https://tei-c.org (дата звернення: 01.12.2024).

28. LanguageTool – це багатомовна перевірка орфографії, стилістики та граматики, яка допомагає виправляти та перефразовувати тексти. LanguageTool. URL: https://languagetool.org/uk (дата звернення: 01.12.2024).

29. GitHub - brown-uk/nlp_uk: this is a project to demonstrate NLP API from languagetool for ukrainian language. GitHub. URL: https://github.com/brown-uk/nlp_uk (дата звернення: 03.04.2025).

30. Lang.org.ua/languk/corpus at master · lang-uk/lang.org.ua. GitHub. URL: https://github.com/lang-uk/lang.org.ua/tree/master/languk/corpus (дата звернення: 03.04.2025).

31. Lang-uk. Головна: lang-uk. URL: https://lang.org.ua/en/ubertext/ (дата звернення: 03.04.2025).

32. Python-tesseract is a python wrapper for Google's Tesseract-OCR. PyPI.

URL: https://pypi.org/project/pytesseract/ (дата звернення: 01.12.2024).

33. End-to-End multi-lingual optical character recognition (OCR) solution. PyPI. URL: https://pypi.org/project/easyocr/ (дата звернення: 01.12.2024).

34. Document text recognition - a seamless, high-performing & accessible library for ocr-related tasks powered by deep learning. GitHub. URL: https://github.com/mindee/doctr (дата звернення: 01.12.2024).

35. This is a slightly polished and packaged version of the keras CRNN implementation and the published CRAFT text detection model. github.com. URL: https://github.com/faustomorales/keras-ocr (дата звернення: 01.12.2024).

36. Open-source character recognition. GOCR. URL: https://jocr.sourceforge.net (дата звернення: 01.12.2024).

37. IBM watson speech to text. IBM - United States. URL: https://www.ibm.com/products/speech-to-text (дата звернення: 01.12.2024).

38. Speech to text overview - Speech service - Azure AI services. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/azure/ai-services/speech-service/speech-to-text (дата звернення: 01.12.2024).

39. Speech-to-Text AI: speech recognition and transcription. Google Cloud. URL: https://cloud.google.com/speech-to-text (дата звернення: 01.12.2024).

40. Whisper AI: speech to text. OpenAI Platform. URL: https://platform.openai.com/docs/guides/speech-to-text (дата звернення: 01.12.2024).

41. Pafy Documentation – pafy 0.5.1 documentation. PyPI Package and Documentation Storage. URL: https://pythonhosted.org/pafy/ (дата звернення: 01.12.2024).

42. Pytube is a lightweight, pythonic, dependency-free, library (and command-line utility) for downloading youtube videos. pytube – pytube 15.0.0 documentation. URL: https://pytube.io/en/latest/ (дата звернення: 01.12.2024).

43. A fast and powerful scraping and web crawling framework. Scrapy. URL: https://scrapy.org (дата звернення: 01.12.2024).

44. Systematic review on text normalization techniques and its approach to non-standard words / A. A. Aliero та ін. International journal of computer applications.

2023. Т. 185, № 33. С. 44–55. URL: https://doi.org/10.5120/ijca2023923106 (дата звернення: 23.05.2025).

45. Starko V., Rysin A., Shvedova M. Ukrainian text preprocessing in GRAC. 2021 IEEE 16th international conference on computer sciences and information technologies (CSIT). 2021. Т. 2. С. 101–104.

46. Vakulenko M. Normalization of Ukrainian letters, numerals, and measures for natural language processing. Digital scholarship in the humanities. 2022. URL: https://doi.org/10.1093/llc/fqac090 (дата звернення: 23.05.2025).

47. Cudak M., Piech M., Marcjan R. Sparse data classifier based on the first-past-the-post voting system. Computer science. 2022. Т. 23, № 2. URL: https://doi.org/10.7494/csci.2022.23.2.4086 (дата звернення: 23.05.2025).

48. Van der Goot R., Çetinoğlu Ö. Lexical normalization for code-switched data and its effect on POS tagging. Proceedings of the 16th conference of the european chapter of the association for computational linguistics: main volume. 2021. С. 2352–2365.

49. Grammarly: free AI writing assistance. Grammarly. URL: https://www.grammarly.com (дата звернення: 26.05.2025).

50. OpenAI. Introducing ChatGPT. OpenAI. URL: https://openai.com/index/chatgpt/ (дата звернення: 26.05.2025).

51. ДСТУ 3582:2013. Інформація та документація. Бібліографічний опис. Скорочення слів і словосполучень українською мовою. Загальні вимоги та правила (ISO 4:1984, NEQ; ISO 832:1994, NEQ). На заміну ДСТУ 3582-97 ; чинний від 2014-01-01. Вид. офіц. Київ : Мінекономрозвитку України, 2014. 15 с.

52. Stanza – A python NLP package for many human languages. Stanza. URL: https://stanfordnlp.github.io/stanza/ (дата звернення: 26.05.2025).

53. GitHub - lang-uk/tokenize-uk: Simple python lib to tokenize texts into sentences and sentences to words. Small, fast and robust. Comes with ukrainian flavour. GitHub. URL: https://github.com/lang-uk/tokenize-uk (дата звернення: 26.05.2025).

54. Horielov D., Vechur O. Text normalization challenges in ukrainian news corpus: problem analysis and research framework // Сучасні інформаційні технології

та системи штучного інтелекту: матеріали 1ї Міжнародної науково-практичної конференції. 2025. С. 210–213.

55. Орфографія та синтаксис. Дизайн система державних сайтів України. URL: https://design.gov.ua/ua/teksty-i-kontent/orfografiya-ta-sintaksis (дата звернення: 03.04.2025).

56. Як дізнатися, який оператор використовує код? | Vodafone Україна. Vodafone Україна - Мобільний зв'язок, Інтернет, Тарифи та Послуги. URL: https://www.vodafone.ua/support/faq/jak-diznatysj-kod-operatora (дата звернення: 03.04.2025).

57. Український правопис / ред.: Є. І. Мазніченко та ін. Наук. думка, 2019. 390 с.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

4. Дубок В. Ю. Дослідження сервісів AWS для обробки природної мови / В. Ю. Дубок ; наук. керівник к. т. н., доцент Н. С. Кравець // Радіоелектроніка та молодь у XXI столітті : матеріали 28-го Міжнар. молодіж. форуму, 16–18 квітня 2024 р. – Харків : ХНУРЕ, 2024. – Т. 6 – С. 425-427. – DOI : https://doi.org/10.30837/IYF.IIS.2024.425.

5. Каук В. І. Перетини технологій: штучний інтелект, як каталізатор змін у UX/UI дизайні / В. І. Каук // Поліграфічні, мультимедійні та web-технології. Інновації та розвиток: монографія. – Харків: ТОВ «Друкарня Мадрид», 2024. – С. 226-242.

54. Horielov D., Vechur O. Text normalization challenges in ukrainian news corpus: problem analysis and research framework // Сучасні інформаційні технології та системи штучного інтелекту: матеріали 1ї Міжнародної науково-практичної конференції. 2025. С. 210–213.