Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Інтеграція технологій Explainable AI та великих мовних моделей для виправлення та пояснення помилок в англомовних текстах
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-5
Анна Стрелкова
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект
(повна назва освітньої програми)
Керівник доц. Марія Головянко
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Стрелковій Анні Олександрівні
(прізвище, ім'я, по батькові)

1. Тема роботи Інтеграція технологій Explainable AI та великих мовних моделей для виправлення та пояснення помилок в англомовних текстах

затверджена наказом університету від 19 травня 20 25 р. № 378Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18 червня 20 25 р.

3. Вихідні дані до роботи Python, LIME, OpenAI API, NLTK, Scikit-learn, LanguageTool, HTML, CSS, JavaScript, FastAPI, Visual Studio Code, інформація з відкритих джерел про обробку природної мови

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та постановка задачі
2) Теоретичні основи обраних методів
3) Проєктування та розробка вебдодатку
4) Вебдодаток для виправлення і пояснення помилок в англомовних текстах

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Аналіз предметної галузі	20.05 – 23.05	виконано
3	Постановка задачі	24.05 – 26.05	виконано
4	Обґрунтування вибору використаних технологій	27.05 – 30.05	виконано
5	Програмна реалізація	31.05 – 06.06	виконано
6	Оформлення пояснювальної записки	07.06 – 14.06	виконано
7	Підготовка до захисту	15.06 – 17.06	виконано
8	Захист перед ЕК	18.06.2025	

Дата видачі завдання 19 травня 20 25 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Марія Головянко
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 90 с., 29 рис., 2 табл., 3 дод., 22 джерела.

ВЕБДОДАТОК, ВИПРАВЛЕННЯ ТЕКСТУ, ЛІНГВІСТИКА, ОБРОБКА ПРИРОДНОЇ МОВИ, ПОЯСНЕННЯ ВИПРАВЛЕНЬ, GPT, LIME, REST-INTERFACE, XAI.

Об’єктом дослідження є текстова інформація, представлена природною мовою, що підлягає автоматичному аналізу, оцінці та редагуванню.

Предметом дослідження є методи обробки природної мови та пояснюваного штучного інтелекту для автоматичного виправлення тексту та пояснення виправлень.

Мета роботи – розробка вебдодатку, який інтегрує автоматичну корекцію текстів із формуванням природномовних пояснень до кожного виправлення задля забезпечення технічної точності редагування, а також підвищення розуміння користувачем суті помилок, що особливо важливо в контексті навчання або редагування.

Методи дослідження – аналіз наукової літератури з прикладної лінгвістики, обробки природної мови та застосовуваних технологій, практична реалізація з застосуванням вебтехнологій та машинного навчання.

У результаті дослідження було створено вебдодаток, що дозволяє користуватися функціями інтелектуального виправлення тексту, написаного англійською мовою, та інтелектуального пояснення помилок.

Дане дослідження демонструє приклад ефективного застосування інструментів пояснюваного штучного інтелекту та великих мовних моделей для задачі пояснення рішення моделі.

ABSTRACT

Bachelor’s thesis contains: 90 pp., 29 fig., 2 tabl., 3 ann., 22 references.

EXPLANATION OF CORRECTIONS, GPT, LIME, LINGUISTICS, NATURAL LANGUAGE PROCESSING, REST-INTERFACE, TEXT CORRECTION, WEB APPLICATION, XAI.

The object of the study is text information presented in natural language, subject to automatic analysis, evaluation and editing.

The subject of the study is the methods of processing natural language and explainable artificial intelligence for automatic text correction and explanation of corrections.

The purpose of the work is developing a web application that integrates automatic text correction with the formation of natural language explanations for each correction in order to ensure technical accuracy of editing, as well as to increase the user's understanding of the errors’ essence, which is especially important in the context of studying or text editing.

The research methods are: analysis of scientific literature on applied linguistics, natural language processing and applied technologies, practical implementation using web technologies and machine learning.

As a result of the study, a web application that allows the usage of the functions of intellectual correction of texts written in English and intellectual explanation of errors was created.

This study demonstrates an example of effective application of the tools of explained artificial intelligence and large language models to the task of explaining the model’s solution.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект;

API – Application Programming Interface – програмний інтерфейс;

CORS – Cross-Origin Resource Sharing – спільне використання ресурсів між різними джерелами;

CSS – Cascading Style Sheets – каскадні таблиці стилів;

GPT – Generative Pre-trained Transformer – генеративна модель на основі трансформера;

HTML – HyperText Markup Language – мова розмітки гіпертексту;

HTTP – Hypertext Transfer Protocol – протокол передавання гіпертексту;

JSON – JavaScript Object Notation – запис об’єктів JavaScript;

LIME – Local Interpretable Model-agnostic Explanations – локальні пояснення, незалежні від моделі;

NLP – Natural Language Processing – обробка природної мови;

NLTK – Natural Language Toolkit – набір інструментів для природної мови;

T5 – Text-To-Text Transfer Transformer – трансформер «текст у текст»;

XAI – eXplainable Artificial Intelligence – пояснюваний штучний інтелект;

XML – eXtensible Markup Language – розширювана мова розмітки.

ВСТУП

У сучасному цифровому середовищі текстова інформація є основною формою комунікації в освіті, бізнесі, науці та повсякденному житті. Якість письмового мовлення має безпосередній вплив на ефективність взаємодії, сприйняття інформації та професійну репутацію автора. Водночас, проблема граматичних, лексичних та стилістичних помилок залишається актуальною для широкого кола користувачів – від студентів до фахівців, які працюють з текстами.

Особливої значущості ця проблема набуває у зв’язку з поширеністю англійської мови. У сучасному світі англійська є однією з основних міжнародних мов, якою спілкуються близько двох мільярдів людей. Вона є першою у світі за кількістю мовців та третьою за кількістю носіїв, використовується у більшості сфер життя далеко поза межами англомовних країн. Для тих, хто послуговується нею щодня, важливим є коректне використання мови – як граматичне, так і семантичне. У наш час замість традиційних джерел для перевірки правильності – таких як словники та підручники – набагато частіше застосовуються інтернет-ресурси, включаючи сервіси на основі штучного інтелекту: перекладачі, чат-боти, системи виправлення граматики тощо.

Розвиток великих мовних моделей (LLM), зокрема GPT, відкрив нові можливості для створення інструментів, здатних не лише виявляти помилки в текстах, але й пропонувати коректні виправлення з урахуванням контексту. Проте більшість таких систем орієнтовані на кінцевий результат – редагований текст – і не пояснюють користувачеві, чому саме було внесено ту чи іншу зміну. Це знижує їхню ефективність у навчальному середовищі, де важливо не лише виправити, а й зрозуміти помилку.

Деякі сервіси, як-от Grammarly або LanguageTool, частково реалізують функцію пояснення, але базуються переважно на шаблонних правилах або є закритими комерційними системами з обмеженою

підтримкою мов. У цьому контексті виникає потреба у відкритому інструменті, що поєднує точність автоматичної корекції з прозорістю пояснень, сформульованих природною мовою, зрозумілою користувачеві.

Дане дослідження присвячене розробці вебдодатку, який поєднує механізм автоматичного виправлення англомовних текстів із генерацією зрозумілих пояснень до кожного виправлення. У межах проєкту було використано велику мовну модель GPT для формування коригованої версії тексту та алгоритм пояснюваного ШІ (LIME) для побудови інтерпретацій рішень моделі. Такий підхід орієнтований не лише на технічну точність, але й на підвищення мовної грамотності користувача.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі

1.1.1 Опис предметної галузі

Лінгвістика, або мовознавство – це наука, яка досліджує мову як систему та засіб спілкування. Головним об'єктом її вивчення є структура, функції, закономірності еволюції та варіативності мов. Мова аналізується не лише як інструмент комунікації, а й як когнітивний та культурний феномен, що формує світогляд особистості. Протягом століть лінгвістика розвивалася як фундаментальна гуманітарна наука, проте у XXI столітті вона все більше інтегрується у прикладні сфери, зокрема, цифрові технології та штучний інтелект [1].

У межах лінгвістики виділяють такі основні напрями, як теоретичний та практичний. Теоретична лінгвістика зосереджується на аналізі внутрішньої будови мови та загальних принципів її функціонування. Основними розділами є фонетика, яка вивчає звуки мови; морфологія – структуру слів; синтаксис – правила побудови речень; семантика – значення слів і висловів; прагматика – контекстуальне використання мови. Мета теоретичної лінгвістики полягає у створенні узагальнених моделей мовної системи, що допомагають краще зрозуміти як конкретні мови, так і універсальні мовні механізми.

Прикладна лінгвістика використовує знання про мову для вирішення практичних задач. До них належать галузі перекладознавства, лінгводидактики, термінознавства, лінгвістичної експертизи, а також комп’ютерної лінгвістики – дисципліни, що знаходиться на перетині мовознавства та інформаційних технологій.

Прикладна лінгвістика прагне не тільки описати мову, а й створити інструменти для її ефективного застосування в освітніх, правових,

міжнародних та цифрових сферах [1]. Сьогодні дедалі більшої значущості набувають такі напрями, як когнітивна лінгвістика, соціолінгвістика, корпусна лінгвістика, а також моделювання мови з використанням штучного інтелекту.

На перетині мовознавства та комп'ютерних наук виникає обробка природної мови. Вона включає в себе автоматичну перевірку правопису, синтаксичний розбір, розпізнавання сенсу, генерацію текстів тощо. Без фундаментальних лінгвістичних знань – зокрема, граматичних правил, морфологічних моделей та семантичних полів – неможливо розробити ефективні NLP системи. Саме лінгвістика забезпечує основу для створення алгоритмів машинного розуміння мови [1].

Застосування лінгвістики в сфері ШІ передбачає розробку моделей мовлення, діалогових систем, генерацію текстів та створення інтерпретованих пояснень. Сучасні великі мовні моделі ґрунтуються на статистичному аналізі мовних структур, що поєднується з глибинними нейронними мережами. Лінгвістика допомагає не тільки в навчанні таких моделей, а й у створенні інтерфейсів, здатних зрозуміло пояснювати свої дії користувачам.

В даній кваліфікаційній роботі лінгвістика виступає фундаментом для розробки системи автоматичної корекції текстів. Теоретичні знання про синтаксис і морфологію застосовуються для виявлення граматичних порушень, а прагматика – для оцінки доречності висловлювань у контексті. Прикладна лінгвістика забезпечує інструментарій для інтеграції цих знань у цифрове середовище через NLP.

Крім того, використання пояснюваного ШІ (LIME) у поєднанні з OpenAI API дозволяє генерувати не тільки виправлення, а й аргументацію – тобто, пояснення причин, чому певна форма вважається помилковою [1].

1.1.2 Обробка природної мови як інструмент цифрової лінгвістики

Обробка природної мови (англ. Natural Language Processing, NLP) є міждисциплінарною сферою, що поєднує лінгвістику, інформатику та штучний інтелект. Основна мета NLP полягає в тому, щоб навчити комп’ютери розуміти, інтерпретувати та генерувати мову, якою спілкуються люди. Це дозволяє створювати технології, що забезпечують природну взаємодію між людиною та машиною [2].

До ключових задач NLP належать аналіз морфологічної структури слів, синтаксичний розбір речень, семантична інтерпретація, виявлення емоцій у тексті, розпізнавання мовлення, а також генерація тексту. Використовуються як класичні лінгвістичні правила, так і сучасні алгоритми машинного навчання. Особливої популярності набули нейромережеві підходи, що здатні обробляти великі обсяги текстів і виявляти закономірності, що не піддаються простому формальному опису [2].

Інструменти NLP працюють з текстом як із послідовністю символів, слів або речень, які спочатку проходять етапи токенізації, нормалізації, морфологічного аналізу, синтаксичного розбору, а іноді – і семантичного аналізу. Кожен із цих етапів дає змогу машині отримати дедалі більше структурованої інформації про текст.

Токенізація є одним з перших та найбільш важливих етапів попередньої обробки тексту. Вона являє собою розбиття тексту на токени, тобто менші одиниці – в залежності від задачі, це можуть бути речення або слова та знаки. Для розбиття тексту на слова використовуються регулярні вирази або спеціальні правила розбиття, що враховують спеціальні символи, такі як пробіли, пунктуація, лапки тощо. Для розбиття на речення використовують статистичні моделі, що навчалися на великих корпусах даних і не лише використовують крапку як роздільник речень, а й

враховують винятки, такі як «Dr.», «Inc.», «e.g.». Разом із токенізацією часто проводиться видалення стоп-слів – сполучників, прийменників тощо.

Наступним етапом препроцесингу, або попередньої обробки тексту, є нормалізація, що здійснюється за допомогою лематизації та/або стемінгу. Задачею цього етапу є зведення слів до їхніх основних форм задля того, щоб різні граматичні форми слова сприймались однаково. Стемінг – простіший спосіб – обрізає слово до кореня, використовуючи прості правила, іноді припускаючись помилок. Лематизація, в свою чергу, використовує словники та граматичні правила, щоб точно визначити лексичну основу слова.

Наступним етапом часто йде визначення частин мови, або POS-tagging. Цей етап має на меті визначення граматичних ролей кожного слова у реченні – іменник, прикметник, дієслово тощо. Бібліотеки для обробки природної мови часто використовують Hidden Markov Models (HMM) або інші статистичні моделі, які аналізують контекст слова та порівнюють з імовірностями того, до якої частини мови слово може належати в конкретному місці.

Синтаксичний аналіз, або парсинг – це побудова структури, що описує взаємозв’язки між словами в реченні, наприклад, граматичного дерева. Для реалізації використовуються контекстно-вільні, залежні та інші граматики, що описують можливі структури речень, а парсер будує дерево залежностей, у якому фіксує, які слова є головними, а які – залежними. Іноді застосовуються рекурсивний спуск, діаграмні парсери або нейромережеві моделі.

За потреби проводиться семантичний аналіз – розуміння значення та сенсу тексту: що саме мається на увазі, які об’єкти, події згадуються тощо. Це реалізовується за допомогою лексичних баз даних, векторних моделей або мовних моделей. До семантичного аналізу належать наступні методи:

– Named Entity Recognition (NER), тобто розпізнавання імен (людей, місць, організацій);

– Coreference Resolution, тобто визначення, які слова/займенники відносяться до одного об’єкта (наприклад, «Анна взяла книгу. Вона читала її...» – хто «вона», що «її»);

– Semantic Role Labeling (SRL) – виявлення дій і ролей учасників (хто що робить?);

– Word Sense Disambiguation (WSD) – розпізнавання значення багатозначного слова у контексті.

Практична реалізація модулів NLP базується на використанні спеціалізованих бібліотек і платформ: NLTK, spaCy, Stanza, Hugging Face Transformers, а також TensorFlow або PyTorch. Для навчання NLP систем критично важливими є лінгвістичні корпуси, частотні словники, тезауруси та анотовані текстові бази. Їхнє використання забезпечує узгодженість, коректність та адаптивність роботи алгоритмів до різних мов та стилів [3].

Цифрова лінгвістика використовує потенціал NLP для масштабної обробки текстових масивів у лінгвістичних розвідках. Це забезпечує автоматизацію роботи з текстами, виявлення мовних шаблонів у живому мовленні, частотний аналіз, створення лексичних графів та проведення стилістичного моделювання. Завдяки NLP відчиняються двері до новітніх методів дослідження мови, що були недоступні при традиційних підходах.

Попри динамічний розвиток NLP, індустрія постає перед певними викликами: контекстна неоднозначність, обробка багатомовних даних, труднощі семантичного аналізу та необхідність етичного нагляду за результатами роботи ШІ моделей. При цьому нові дослідження демонструють потенціал великих мовних моделей, зокрема GPT, не лише в генерації текстів, а й у підтримці когнітивної інтерпретації, порівняння знань і діалогової аналітики, що наближає NLP до рівня високорівневої наукової взаємодії [4].

1.2 Аналіз наявних рішень

1.2.1 Grammarly та LanguageTool

Серед сучасних систем автоматичного виправлення текстів найрозвиненішими в аспекті пояснення запропонованих змін є Grammarly та LanguageTool. Обидва сервіси дають змогу не тільки виявляти граматичні, орфографічні та стилістичні помилки, але й пояснюють причини їхнього виправлення, що має неабияке значення для освітніх цілей та самостійного навчання користувача.

Grammarly – комерційний онлайн сервіс, що інтегрується в текстові редактори, браузери та мобільні додатки. Його особливість полягає в поєднанні глибоких мовних моделей і лінгвістичних правил, що дозволяє не лише знаходити помилки, але й надавати користувачеві пояснення, чому певна конструкція є некоректною. З технологій Grammarly використовує методи глибокого навчання, зокрема трансформерні моделі, аналогічні до GPT, а також машинне навчання для персоналізації стилю та виявлення типових помилок [5]. Для пояснення застосовується комбінація правил і шаблонів, створених на основі корпусного аналізу та лінгвістичних досліджень.

LanguageTool – open-source альтернатива Grammarly, яка підтримує понад 25 мов, включно з українською. Вона базується переважно на лінгвістичних правилах, сформованих у спеціальній XML-структурі, що дозволяє легко додавати або змінювати правила вручну. Крім того, LanguageTool підтримує використання n-грамів і нейромережевих моделей для складніших перевірок, як-от узгодження відмінків або виявлення омонімічних конструкцій. Система часто супроводжує виправлення коротким поясненням: наприклад, «в даному випадку слово вжите у неправильному відмінку», або «можливо, ви мали на увазі інше слово». Хоча ці пояснення мають менше контексту, ніж у Grammarly, вони

будуються на чітко прописаних лінгвістичних правилах і мають високу передбачуваність [6].

1.2.2 Інструменти синтаксичного редагування тексту

Microsoft Editor і Google Docs – це повсякденні інструменти передусім для редагування тексту. Крім того, обидва сервіси реалізують автоматичну перевірку правопису та граматики, однак мають значні відхилення та обмеження, коли йдеться про пояснення виправлень.

Microsoft Editor є складовою екосистеми Microsoft 365 і інтегрується у Word, Outlook, браузер Edge та інші продукти компанії. Система перевіряє орфографію, граматику, пунктуацію, а також надає рекомендації щодо ясності, формальності та лексичної доречності. Пояснення виправлень стислі та шаблонні – наприклад, повідомлення «орфографія» або «уникати повторів» не містять глибокого аналізу, але можуть бути підказкою для досвідченого користувача. З технологій Microsoft Editor використовує нейронні трансформерні моделі, розроблені у рамках ініціативи Turing NLG. Зокрема, для граматичної перевірки застосовуються автогенеративні моделі, які вчаться перетворювати текст із помилками на граматично правильні конструкції [7]. Водночас пояснення формуються на основі вбудованих мовних правил, які діють як інтерпретаційний шар для користувача.

Google Docs, як складова G Suite, також реалізує автоматичну перевірку правопису та граматики. Система підкреслює помилки червоним або синім кольором, пропонує варіанти виправлень, однак майже не надає пояснень, обмежуючись вказівками на тип помилки. Основна технологічна особливість полягає в тому, що граматична корекція в Google реалізована як задача машинного перекладу – текст із помилками перетворюється на «переклад» граматично правильного варіанта, подібно до перекладу з однієї мови на іншу. Для цього використовуються нейронні мережі,

натреновані на великих корпусах текстів [8]. Такий підхід ефективний у плані точності, але не передбачає прозорості – тобто, пояснення, чому саме система внесла ту чи іншу зміну, зазвичай відсутнє.

Обидві системи демонструють високу інтегрованість та зручність для широкого кола користувачів. Проте, з точки зору пояснення виправлень, вони значно поступаються системам на кшталт Grammarly або LanguageTool.

1.2.3 Сервіси стилістичного редагування тексту

Окрім традиційних систем граматичної перевірки, існують сервіси, що акцентують увагу на стилістичному редагуванні та перефразуванні. З-поміж таких інструментів слід виділити DeepL Write, QuillBot та Hemingway Editor, які показують високу ефективність у поліпшенні текстів, однак не надають роз'яснень до запропонованих змін.

DeepL Write – це нова розробка від творців популярного перекладача DeepL, націлена на стилістичне редагування текстів англійською та німецькою мовами. Система пропонує користувачеві покращення формулювань, заміну окремих слів, спрощення конструкцій. При цьому відсутні будь які обґрунтування щодо змін – система демонструє лише варіант «кращого» написання. В технічному плані DeepL Write використовує трансформерні нейромережі, схожі на ті, що використовуються у GPT моделях, навчені на паралельних корпусах текстів [9]. Її основна мета – підвищення зрозумілості та природності мовлення, а не навчання користувача через роз’яснення.

QuillBot – популярний сервіс для перефразування та стилістичного редагування текстів англійською мовою. Він пропонує декілька «тонів» або стилів рерайту, зокрема «formal», «simple», «creative» та інші. Подібно до DeepL Write, цей сервіс не пояснює причин змін – він просто демонструє альтернативні варіанти. Технічно QuillBot базується на нейронних мовних

моделях, які аналізують структуру речень та генерують переформульовані версії відповідно до обраного стилю. У фокусі – зручність і швидкість переписування, а не поглиблений лінгвістичний аналіз [10].

Hemingway Editor – офлайн або вебдодаток для аналізу стилістичної складності англомовного тексту. Система підсвічує складні конструкції, пасивні форми, надмірне використання прислівників та інші аспекти, що ускладнюють сприйняття. Водночас інструмент не надає лінгвістичних пояснень – користувач бачить лише візуальне виділення проблемних фрагментів без коментарів. Алгоритм Hemingway базується переважно на фіксованих лінгвістичних правилах з використанням простого синтаксичного аналізу, без глибокого машинного навчання [11].

Таким чином, хоча згадані сервіси й покращують якість тексту, вони не виконують пояснювальну функцію. Це обмежує їхнє використання в освітньому середовищі або в інших ситуаціях, де користувачеві важливо зрозуміти причини помилки.

1.3 Порівняння розробленого рішення з наявними

Проведений аналіз наявних рішень для автоматичного виправлення текстів виокремлює кілька важливих зауважень щодо їхніх функціональних обмежень та недоліків, що, у свою чергу, підкреслює актуальність та новизну запропонованого підходу в межах даного дослідження.

Більшість поширених інструментів (Microsoft Editor, Google Docs, DeepL Write, QuillBot) акцентують увагу винятково на фінальному результаті – виправленому або перефразованому тексті. Вони не супроводжують ці зміни поясненнями або коментарями, які могли б допомогти користувачеві зрозуміти суть помилки. Системи, які все ж реалізують пояснення (зокрема Grammarly та LanguageTool), базуються або на попередньо визначених лінгвістичних правилах, або на шаблонних моделях пояснень. Вони не пояснюють логіку конкретного виправлення

з точки зору моделі, яка його здійснила, а лише накладають описову інтерпретацію постфактум. Крім того, такі системи або є комерційними та закритими (Grammarly), або мають обмежений обсяг підтримки мов та стилістичних контекстів (LanguageTool).

На відміну від них, розроблене в межах дипломного проєкту рішення поєднуватиме автоматичну корекцію текстів з генерацією пояснень до кожного виправлення, що формуються на основі аналізу впливу вхідних даних за допомогою LIME (Local Interpretable Model agnostic Explanations). Це забезпечить високу гнучкість, адаптивність до стилістичних варіантів, а головне – прозорість та зрозумілість дій системи для користувача.

1.4 Постановка задачі

В рамках даного дослідження розробляється інтелектуальна система, призначена для автоматичного виправлення текстів, написаних англійською мовою, супроводжуючи кожне виправлення зрозумілими поясненнями. Актуальність цього проєкту зумовлена потребою у засобах, що не лише виправляють помилки, але й сприяють підвищенню мовної грамотності користувачів.

Об’єктом дослідження є текстова інформація, представлена природною мовою, що підлягає автоматичному аналізу, оцінці та редагуванню.

Предметом дослідження є методи обробки природної мови (NLP) та пояснюваного штучного інтелекту (XAI), зокрема технології генерування виправлень на основі великих мовних моделей, а також побудови пояснень з використанням локальних інтерпретаторів моделей.

Метою даної дипломної роботи є створення вебдодатку, який інтегрує автоматичну корекцію текстів із формуванням природномовних пояснень до кожного виправлення. Такий підхід має на меті не лише забезпечення

технічної точності редагування, але й підвищення розуміння користувачем суті помилок, що особливо важливо в контексті навчання або редагування.

Для досягнення поставленої мети заплановано виконати такі завдання:

– проаналізувати наявні сервіси автоматичної корекції текстів, виявивши їхні слабкі сторони;

– обґрунтувати вибір архітектури вебдодатку, яка включає мовну модель (наприклад, GPT) та інструменти інтерпретації (скажімо, LIME);

– реалізувати бекенд на основі сучасного фреймворку (наприклад, FastAPI) з підтримкою REST інтерфейсу;

– розробити фронтенд, що забезпечує зручне введення тексту, перегляд результатів та пояснень до кожного виправлення;

– протестувати систему на різних типах текстів та провести якісну оцінку пояснень.

Очікуваним результатом є функціональний вебдодаток, що дозволяє користувачам взаємодіяти з текстом в режимі реального часу: вводити текст, отримувати граматично та стилістично виправлену версію, а також ознайомлюватися з аргументованими поясненнями до кожної зміни. Проєкт поєднує аспекти лінгвістики, обробки природної мови, великих мовних моделей, вебпрограмування та UI/UX дизайну, що робить його міждисциплінарно значущим.

2 ТЕОРЕТИЧНІ ОСНОВИ ОБРАНИХ МЕТОДІВ

2.1 Методи виправлення помилок

2.1.1 Класифікація помилок

З точки зору лінгвістики, помилки зазвичай поділяють на кілька основних типів, кожен з яких потребує окремого підходу до виявлення та виправлення. Правильна класифікація помилок є критично важливою для побудови точних моделей їх автоматичного виправлення, оскільки різні типи помилок вимагають застосування різних підходів.

Найпростішими для виправлення як людиною, так і моделями машинного навчання, є орфографічні та пунктуаційні помилки. Орфографічні помилки пов’язані з неправильним написанням слів, зокрема через пропущені літери, зайві символи або неправильний порядок – наприклад, definately замість definitely. Пунктуаційні помилки виникають через неправильне використання розділових знаків (ком, крапок, апострофів, лапок тощо), що впливає на зрозумілість тексту. Прикладом такої помилки може бути «Lets eat grandma» замість «Let’s eat, grandma».

Граматичні помилки охоплюють порушення правил узгодження підмета й присудка, вживання неправильних форм дієслів, артиклів, неправильне використання часів тощо. Наприклад: «He go to school» замість «He goes to school».

Стилістичні помилки стосуються надмірного повторення, зайвих слів, невідповідності тону або реєстру тексту (формального/неформального стилю), логічної зв’язності. Такі помилки не завжди є «граматичними», але знижують якість письма.

Лексичні помилки виникають унаслідок неправильного вибору слова з погляду контексту, синонімії, колокацій або частоти вживання. Наприклад, слово big може бути граматично правильним, але

в академічному стилі доречнішим буде large або significant. Такі помилки важко виявити без аналізу контексту, тому для них часто використовують глибокі моделі або лексичні бази даних (наприклад, WordNet).

На сьогоднішній день найскладнішим типом помилок для ШІ вважаються семантичні помилки – це помилки змісту, коли речення граматично правильне, але сенс порушено або не відповідає намірам мовця. Наприклад, «He drank the sandwich» – формально правильна структура, але семантично хибна. Виявлення таких помилок вимагає розуміння глибокого контексту, знань про світ і часто залучення великих мовних моделей.

Автоматичне виявлення та виправлення помилок у текстах англійською мовою є складною міждисциплінарною задачею, яка охоплює лінгвістику, машинне навчання та обробку природної мови. Протягом останніх десятиліть було розроблено різноманітні методи, що дозволяють комп’ютерним системам аналізувати текст, виявляти відхилення від нормативної мови та пропонувати виправлення. Усі ці методи можна умовно класифікувати на чотири основні групи: rule-based (на основі правил), statistical (статистичні), machine learning-based (на основі машинного навчання) та neural (нейронні моделі). Кожен підхід має власні сильні сторони та недоліки, і найефективніші сучасні системи зазвичай використовують гібридні стратегії, комбінуючи декілька підходів для досягнення високої точності та узагальнюваності.

2.1.2 Методи виправлення помилок на основі правил

Rule-based системи, або системи на основі правил, базуються на заздалегідь визначених лінгвістичних правилах, створених вручну експертами. Під час перевірки за допомогою системи правил текст проходить через такі етапи, як токенізація, POS-tagging, аналіз залежностей та застосування кожного правила до відповідного шаблону у тексті. Правила можуть бути задані у різних форматах:

– XML-правила, що задаються у вигляді лексем, POS-тегів та лінгвістичних ознак;

– регулярні вирази, що використовуються для виявлення повторів та орфографічних шаблонів;

– правила на базі мови програмування Java, що використовуються у складніших випадках, таких як ті, що потребують логіки та чітких умов, наприклад, узгодження родів.

Яскравим прикладом системи на основі правил є LanguageTool, який включає тисячі правил для перевірки граматики, орфографії, пунктуації, стилю та формальності. Інструмент працює з понад 30 мовами, може бути використаний як в якості самостійного редактора, так і інтегровано у браузери, текстові процесори або сторонні додатки. Основною перевагою LanguageTool є його можливість створення правил для виявлення помилок за допомогою шаблонів і регулярних виразів, що робить його гнучким і придатним як для простих перевірок, так і для складного мовного аналізу: «Воно знаходить багато помилок, які не може виявити звичайна перевірка орфографії, наприклад, плутанина their/there, a/an або повторення слів» [12].

Іншою важливою перевагою цього підходу є висока прозорість: користувач або розробник завжди може зрозуміти, чому система виявила помилку. Проте його недоліки полягають у складності масштабування та підтримки: створення нових правил вимагає значних витрат ресурсів і не дозволяє охопити всі можливі мовні варіанти.

2.1.3 Трансформерні моделі в задачі виправлення

Трансформер – це тип архітектури нейронних мереж, що вважається революційним у сфері обробки природної мови, як і в багатьох інших сферах штучного інтелекту, є наступним поколінням після RNN та LSTM. RNN – Recurrent Neural Network – працює послідовно, обробляючи слова

одне за одним, що спричиняє низьку швидкість та низьку ефективність роботи з довгими залежностями. LSTM – Long Short-Term Memory – покращена версія RNN, яка має довшу «пам’ять» для важливої інформації, але все ще дуже обмежена, якщо порівнювати з трансформерним типом архітектури.

Вперше трансформери були представлені у науковій статті «Attention is All You Need», що була опублікована в 2017 році, та з тих пір стали основою для моделей типу BERT, GPT, T5, RoBERTa, XLNet тощо.

Основною відмінністю трансформерів від попередників є механізм уваги – attention, що дозволяє паралельну обробку тексту. За допомогою цього механізму кожне слово «звертає увагу» на всі інші слова в реченні незалежно від їхніх позицій [13]. З математичної точки зору кожне слово перетворюється на три вектори: Query – «що я шукаю?», Key – «який ключ має слово?» і Value – «яку інформацію несе це слово?». Після цього модель обчислює схожість між Query одного слова та Keys всіх інших, в результаті чого створюється комбінація залежних Value-векторів.

Основними компонентами трансформерів є:

– embedding, що перетворює токени на числові вектори, які ще називаються векторними уявленнями слів;

– position encoding, що забезпечує інформацію про порядок слів, бо сам трансформер є непослідовним;

– attention mechanism, зокрема, self-attention, що дозволяє кожному слову оцінювати значимість інших слів у контексті;

– feedforward layers, тобто щільні шари, що обробляють вектори після механізму уваги;

– normalization і residuals, що використовуються задля стабілізації тренування та щоб дозволити глибокі архітектури.

Детальніше архітектуру трансформерів представлено на рисунку 2.1.

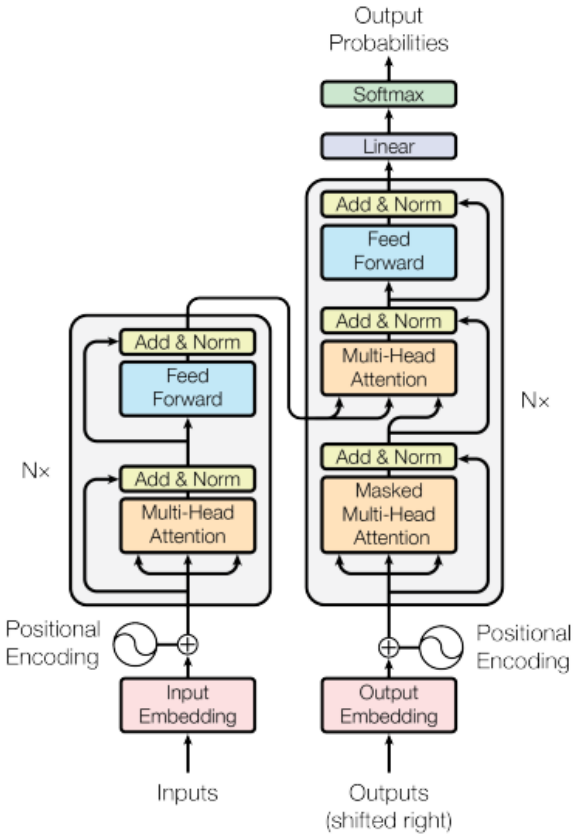

Рисунок 2.1 – Архітектура трансформерів

На сьогоднішній день трансформери використовуються у багатьох задачах обробки природної мови. Найпоширеніші великі мовні моделі, такі як GPT-3, GPT-4, BERT, T5, Claude, LLaMA тощо, мають в основі трансформерну архітектуру. Так само розповсюджена вона і в задачах перекладу, синтезу та розпізнавання мови, класифікації текстів та генерації коду, зображень та музики.

У проекті, що розробляється, використано декілька моделей типу трансформер. Першою є модель T5 (Text-To-Text Transfer Transformer), що використовується в якості першого етапу корекції та виправляє граматичні помилки. Другою є модель типу GPT, що використовується в кількох місцях програми – а саме, для корекції семантики та стилістики, а також для «перекладу» пояснення виправлень від LIME на природну мову, зрозумілу користувачеві.

2.1.3.1 Родина моделей на основі архітектури T5

T5 (Text-to-Text Transfer Transformer) – це потужна модель обробки природної мови (NLP), розроблена дослідницькою командою Google Research у межах проєкту Text-To-Text Transfer Transformer (T5) у 2019 році. Основна ідея моделі – уніфікувати всі NLP-задачі під один формат: перетворення тексту в текст. Це означає, що незалежно від того, чи йдеться про класифікацію, переклад, узагальнення або виправлення граматики – усі завдання представляються як вхідний текст, що потрібно трансформувати у цільовий текст. Такий підхід забезпечує високу гнучкість і дозволяє використовувати єдину архітектуру та метод навчання для багатьох задач, спрощуючи загальний фреймворк для NLP.

T5 базується на класичній архітектурі трансформера, запропонованій у статті «Attention is All You Need» [13], але адаптована до задач повного перетворення послідовностей. Модель має дві основні частини: енкодер, що отримує вхідний текст, обробляє його за допомогою багатошарового механізму attention, і формує внутрішнє уявлення (embedding) тексту; та декодер, який бере це уявлення і, послідовно генеруючи токени, формує вихідний текст. Ключова особливість T5 – це умовне декодування: декодер генерує вихідний текст, враховуючи не лише попередні слова, але й контекст з енкодера. Детальніше архітектуру T5 показано на рисунку 2.2.

T5 була навчена на масивному корпусі C4 (Colossal Clean Crawled Corpus) – очищеній версії Common Crawl [14]. Предтренування виконувалося за допомогою методу span corruption, де з вхідного тексту випадково видаляються фрагменти (токени), і завдання моделі – відновити їх. Формат навчання виглядав так: вхід – «Translate English to German: That is good.», вихід – «Das ist gut.». Цей уніфікований формат забезпечує високу узагальнюваність та спрощує інтеграцію моделі в різноманітні завдання.

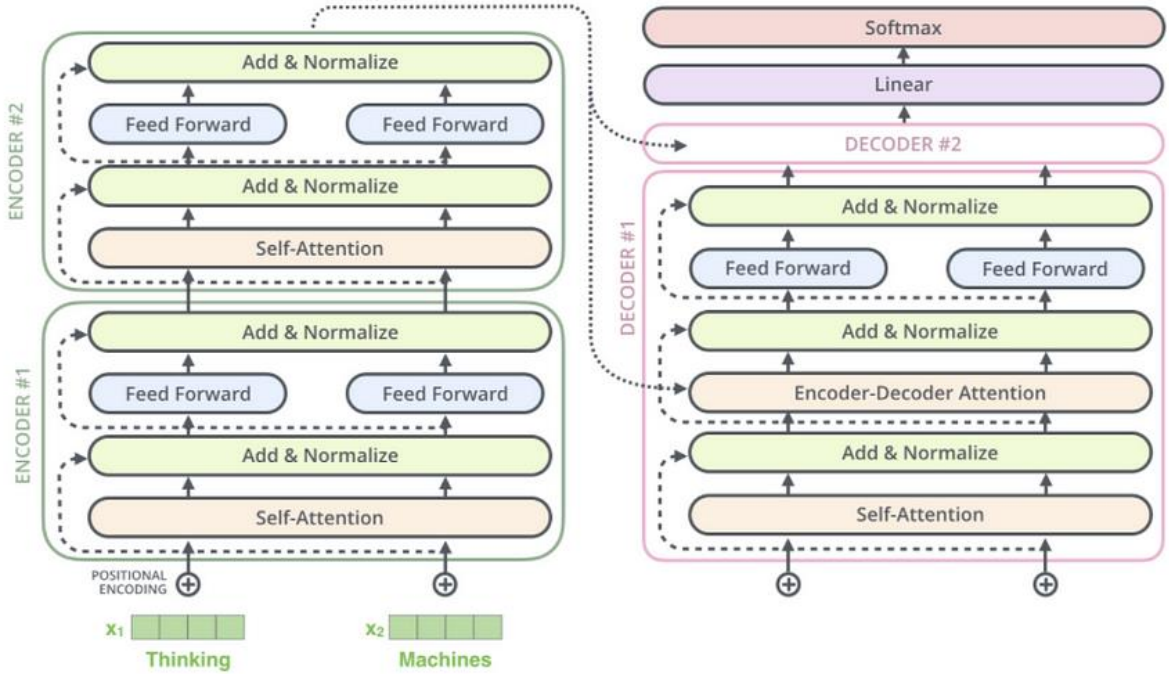

Рисунок 2.2 – Архітектура моделей типу T5

Модель T5 була випущена в кількох версіях: T5-Small, T5-Base, T5-Large, T5-3B, T5-11B. Вони відрізняються кількістю параметрів: 60 млн, 220 млн, 770 млн, 3 млрд, 11 млрд параметрів відповідно.

2.1.3.2 Родина моделей на основі GPT-архітектури

GPT, що розшифровується, як Generative Pre-trained Transformer – це мовна модель, що генерує текст на основі вхідного запиту, та має трансформерну архітектуру, натреновану на величезних обсягах тексту. Вона працює за принципом sequence-to-sequence, тобто, обробляє вхідний текст як послідовність токенів та на вихід повертає нову послідовність токенів в якості відповіді. При роботі з GPT-моделями важливо пам’ятати, що цей тип моделей не «розуміє» мову буквально, а статистично вгадує наступне слово на основі контексту.

На більш глибокому рівні робота GPT складається з таких етапів:

– pre-training, тобто етапу навчання – модель «читає» величезні обсяги тексту та навчається передбачати наступне слово в реченні, а також розпізнавати граматику, структуру мови, логіку тощо;

– fine-tuning, тобто додаткове навчання, що відбувається за допомогою спеціальних корпусів або RLHF (Reinforcement Learning from Human Feedback) для того, щоб модель «поводилась» відповідно до очікувань, наприклад, не видавала образливих відповідей;

– inference, тобто використання, що складається з приймання на вхід тексту природною мовою та генерацію відповіді, розраховуючи ймовірності того, яким буде наступне слово, на кожному кроці.

На сьогоднішній день, найбільшою популярністю користуються моделі GPT-3.5 Turbo, GPT-4 та GPT-4o, від більш старого до більш нового покоління відповідно. Кожна з них має свої переваги та недоліки, а також оптимальна для різних видів задач.

GPT-3.5 Turbo є оптимальною для легких запитів, таких як генерація шаблонного тексту, перевірка граматики, автозаповнення форм, проста класифікація та використання у чатботах. З переваг – висока швидкість генерації та підтримка системних (role-based) повідомлень. Тим не менш, цю модель не рекомендується використовувати для складних логічних задач, інструкцій, що містять багато етапів, або детального аналізу тексту.

GPT-4 добре впорається з семантичним аналізом, написанням складних пояснень або інструкцій, перекладів зі збереженням стилістики через глибше «розуміння» контексту та кращу точність у багатомовному середовищі. З недоліків – значно менша швидкість генерації та більша вартість використання як у браузері, так і через API, а також обмеження по токенах – 32 тисячі.

GPT-4o, випущена у травні 2024 року, є найкращим вибором для мультимодальних задач, що включають голос, зображення або файли, інтерактивних інтерфейсів, швидких систем з високим навантаженням.

Основні переваги – поєднання високої швидкості та точності, а також доступ до різних типів файлів.

У даному дипломному проєкті GPT-моделі використовуються для двох задач: виправлення семантичних та стилістичних помилок у тексті, а також для інтерпретації пояснення від LIME природною мовою. Для полегшення роботи у проєкті використано API через проксі/API-шлюз OpenRouter, що забезпечує простіший доступ до моделей.

2.1.3.3 Порівняння використаних типів архітектур

У таблиці 2.1 наведено порівняння архітектур T5 та GPT.

Таблиця 2.1 – Порівняння використаних архітектур трансформерів

Характеристика	T5	GPT
Розробник	Google	OpenAI
Архітектура	Encoder–Decoder (Seq2Seq)	Decoder-only
Формат задачі	Перетворення тексту в текст	Прогнозування наступного токена
Підходи до навчання	Навчання з підказками (prompt + output)	Автогенерація наступного слова
Типове використання	Переклад, узагальнення, виправлення	Генерація тексту, чат-боти, креативне письмо
Підхід до обробки	Обробляє весь вхід (input) й генерує відповідь	Генерує текст послідовно, знаючи лише контекст
Приклади моделей	T5, mT5	GPT-2, GPT-3, GPT-3.5, GPT-4, GPT-4o

2.2 Методи пояснення виправлень

2.2.1 Explainable Artificial Intelligence

Explainable Artificial Intelligence, або XAI, можна дослівно перекласти як «пояснюваний штучний інтелект». Ключова відмінність цієї галузі штучного інтелекту у тому, що моделі такого типу можуть «пояснити» рішення, які вони приймають, шляхом «заглядання всередину» самого алгоритму та інтерпретації його логіки у зрозумілому для людини вигляді [15]. Основною метою XAI є зробити рішення ШІ зрозумілими та контрольованими для розробників, а також підвищити довіру до моделей штучного інтелекту зі сторони як розробників, так і користувачів.

На сьогоднішній день потреба в пояснюваності та прозорості роботи штучного інтелекту стає все більш важливою разом із тим, як росте частка участі штучного інтелекту в прийнятті критично важливих рішень – наприклад, у галузях медицини, права, освіти, банківської справи тощо. Якщо модель, до прикладу, діагностує захворювання в пацієнта, у лікаря виникає закономірне питання – чому саме модель прийняла таке рішення? З цим допомагає пояснюваний штучний інтелект, забезпечуючи прозорість та даючи простір для самостійних міркувань та рішень людини.

У штучному інтелекті існують два основні підходи до побудови моделей: «чорна скринька» та пояснювана модель. До перших відносяться моделі GPT, нейронні мережі, random forest тощо – їхня внутрішня логіка прихована у складних математичних зв’язках, які майже неможливо інтерпретувати людиною.

До других належать лінійна регресія (рисунок 2.3) та дерево рішень – в таких моделях можна досить легко побачити, яким чином значення кожної ознаки привело до результату. Цей тип пояснюваності, коли модель від початку будується зрозумілою для людини, виділяється XAI як intrinsic explainability, тобто вбудована пояснюваність.

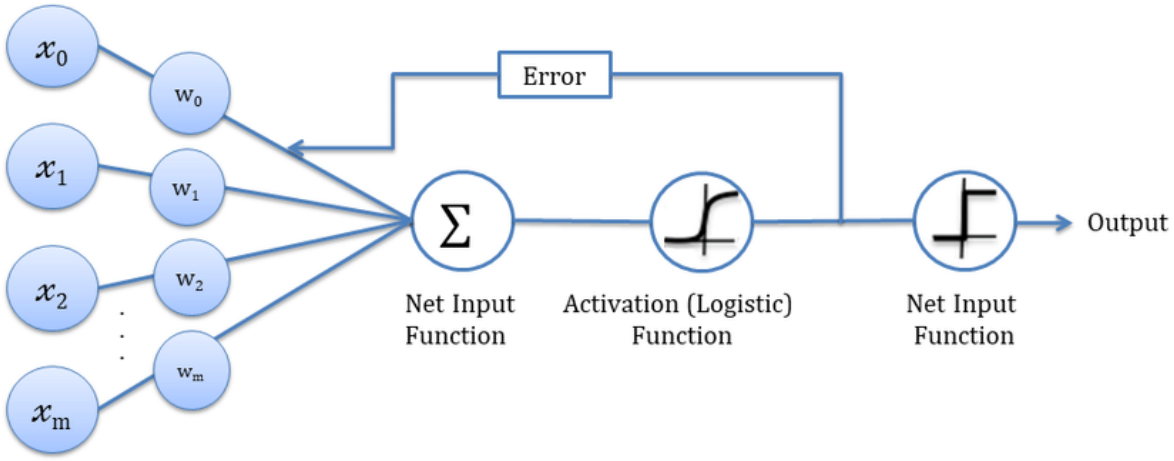

Рисунок 2.3 – Схема роботи моделі лінійної регресії

Однак існує спосіб інтерпретувати і моделі типу «чорна скринька»: цей підхід називається post-hoc explainability та вирізняється тим, що пояснення відбувається вже після роботи сторонньої моделі [15]. Для цього застосовують зовнішні інструменти пояснення, такі як LIME, SHAP, Anchors, counterfactual explanations тощо.

SHAP (повна назва – SHapley Additive exPlanations) заснований на концепції вартості Шеплі з теорії ігор. Він визначає, який внесок зробила кожна ознака у фінальне рішення моделі (рисунок 2.4). Цей інструмент працює з табличними даними, текстовими даними та графами та часто використовується в бізнес-аналітиці, фінансовому прогнозуванні та медицині. Цей інструмент є популярним через свою високу точність та математичну обґрунтованість.

Anchors виявляє логічно сформульовані «якірні» правила – правила, за яких модель завжди робить певне однакове передбачення. Пояснення зазвичай виглядають так: «якщо в реченні є слова А і Б, то в передбаченні буде слово С». Цей інструмент вирізняється своєю зручністю для контекстуальних пояснень у текстах.

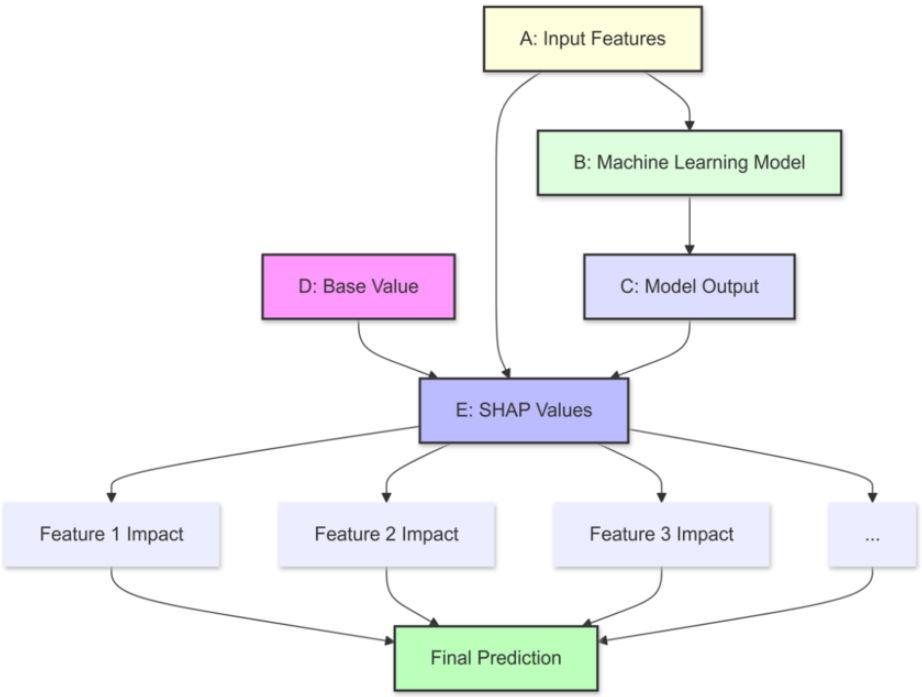

Рисунок 2.4 – Схема роботи методу SHAP

Counterfactual Explanations замість пояснення певного вибору моделі ставить питання, що потрібно змінити у вхідних даних, щоб рішення моделі стало іншим. Цей інструмент є особливо цінним у етично чутливих сферах, таких як кредитний скорінг, медичні діагнози, автоматизований відбір тощо.

2.2.2 Local Interpretable Model-agnostic Explanations

Local Interpretable Model-agnostic Explanations, або LIME – це один з інструментів пояснення пост-фактум для роботи з моделями типу «чорна скринька». Його особливість полягає у тому, що пояснення є локальними – тобто, інструмент пояснює поведінку моделі в одному конкретному випадку, а не загалом.

Покроковий механізм роботи LIME виглядає так:

– на вхід подається приклад, що може бути у текстовому, табличному форматі або зображенням, а також модель, що робить передбачення;

– LIME генерує велику множину схожих на оригінал псевдо-прикладів (perturbed samples), в яких варіює різні ознаки вхідних даних: наприклад, в табличних даних про фізичні характеристики людини у штучних прикладах може змінюватися вік, стать, зріст тощо;

– кожен псевдо-приклад передається в основну модель – «чорну скриньку», що видає прогноз;

– LIME проводить аналіз того, як змінювались прогнози моделі в залежності від змін у вхідних даних та будує просту модель, наприклад, лінійну регресію, що за поведінкою наближена до поведінки основної моделі в контексті конкретного прикладу;

– відбувається обчислення того, які саме елементи або ознаки вхідних даних мали найбільший вплив на рішення основної моделі;

– результат виводиться у вигляді графіків або списків, де різними кольорами підсвічено частини вхідних даних, що вплинули на рішення моделі, а також вказані числові значення впливу [15].

Графічно процес роботи даного інструменту показано на рисунку 2.5.

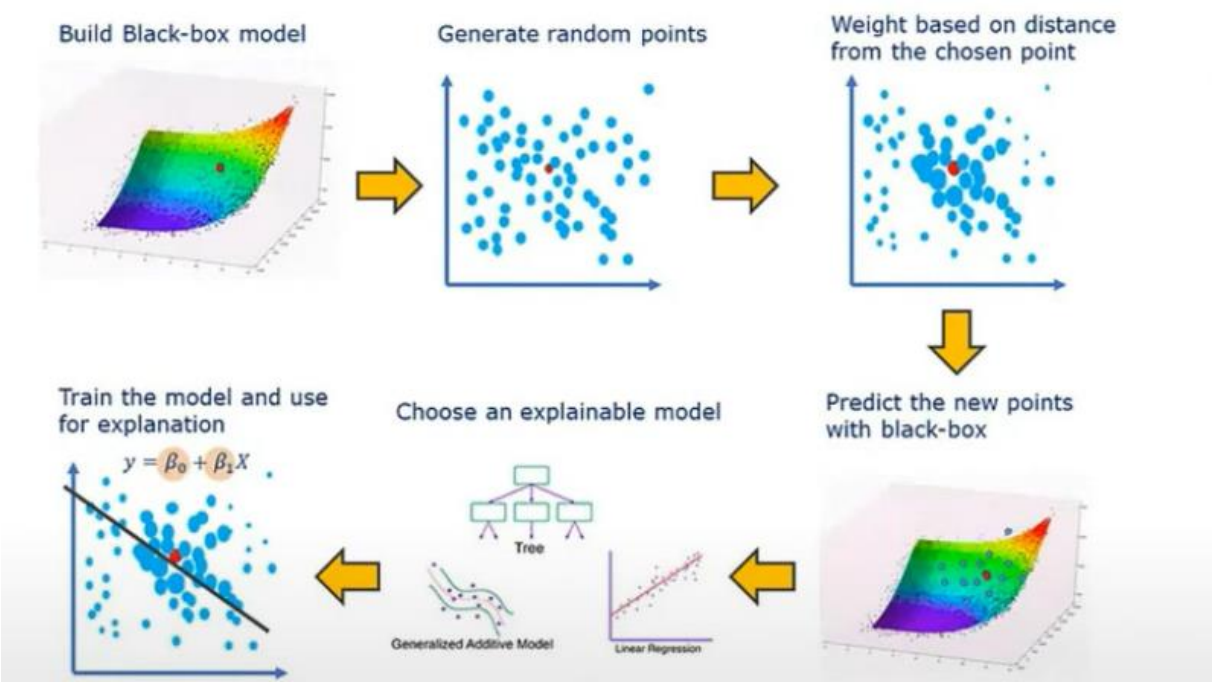

Рисунок 2.5 – Схема роботи інструменту-пояснювача

LIME має різні варіанти пояснювачів для різних типів даних: це LimeTabularExplainer – для числових або табличних даних, LimeTextExplainer – для текстових даних та LimeImageExplainer – для зображень, де можна візуалізувати, які області малюнка яким чином вплинули на рішення. Процес роботи LimeTextExplainer, що використано у даному проєкті, графічно показано на рисунку 2.6.

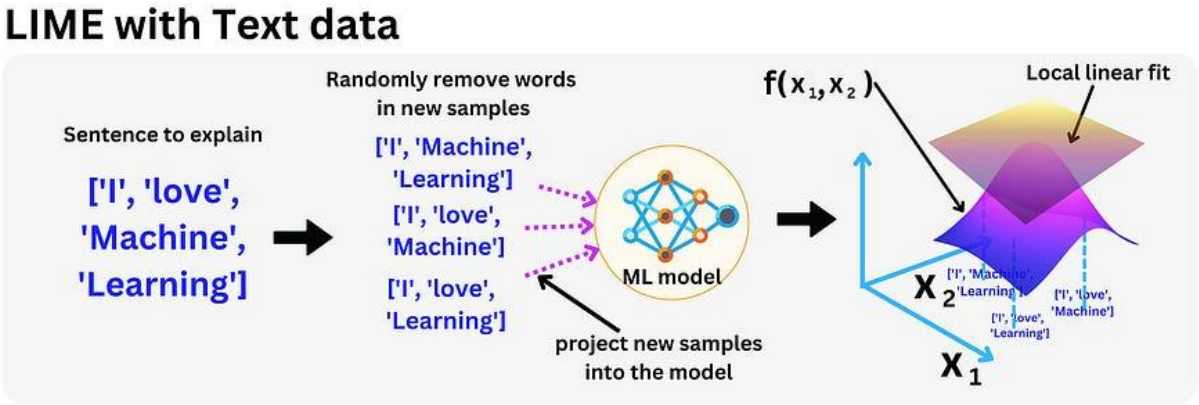

Рисунок 2.6 – Схема роботи пояснювача для текстових даних

Основними перевагими інструменту LIME є простота інтеграції в уже наявний проєкт, адже не потрібно вносити зміни в основну модель, зрозумілість вихідних даних за допомогою графічного пояснення та гнучкість, адже даний інструмент здатен працювати майже з будь-якою моделлю.

Проте, основними недоліками є: локальне пояснення – хоч для деяких задач це перевага, загалом неможливо створити універсальне пояснення роботи моделі, що означає більші витрати ресурсів; шуми в псевдо-прикладах, які можуть виявитися неприродними або дивними, що впливає на реакцію моделі.

У даному проєкті застосування LIME є одною з головних особливостей, що відрізняють розроблений додаток від інших пояснювачів,

які переважно користуються шаблонними правилами або генеративними моделями.

Використано LimeTextExplainer, який порівнює вхідний та виправлений варіанти тексту, будує локальну модель на основі TfidfVectorizer та LogisticRegression, виявляє, які слова основна модель вважає правильними, а які – помилковими, та виводить це у вигляді графіків та вихідного тексту з підсвіченими потрібними словами. Однак, не всім користувачам може бути достатньо саме такого формату – саме через це пояснення LIME в подальшому інтерпретується природною мовою.

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБДОДАТКУ

3.1 Архітектура проєкту

У ході розробки архітектури проєкту було прийняте рішення реалізувати програмне забезпечення у форматі вебдодатку, оскільки цей формат зручний для користувачів, універсальний та легко впроваджуваний. Веб інтерфейс дає змогу використовувати додаток у будь якому сучасному браузері, не встановлюючи додаткове програмне забезпечення. Це найбільш важливо для кінцевих користувачів, що працюють на різних платформах (Windows, macOS, Linux, мобільні ОС). Цей підхід також спрощує оновлення та обслуговування системи: всі зміни на боці сервера одразу ж доступні для клієнтів.

Розроблене програмне забезпечення базується на чітко структурованій двокомпонентній архітектурі, в основі якої лежить модель «клієнт-сервер». В такій моделі клієнтська частина – фронтенд – відповідає за взаємодію з користувачем, а серверна частина – бекенд – за основний функціонал, тобто обробляє запити, проводить текстовий аналіз, генерує виправлення та відповідні пояснення. Цей підхід забезпечує високий рівень модульності, даючи змогу чітко розділити обов'язки між компонентами, забезпечує гнучкість розробки та супроводу проєкту, гарантує легке масштабування в майбутньому, забезпечуючи стабільну роботу навіть при зміні вимог або розширенні функціоналу.

Ключовим фактором вибору даної архітектури стали простота та гнучкість. Поділ на фронтенд та бекенд дозволяє:

– окремо тестувати інтерфейс та логіку обробки тексту;

– за необхідності – замінювати або вдосконалювати один із компонентів без значного впливу на інший (наприклад, підключити нову модель GPT або перевірку тексту іншою мовою);

– легко адаптувати проєкт для різних платформ (веб, мобільні клієнти, браузерні розширення);

– масштабувати функціональність, додаючи нові API ендпоінти (наприклад, підтримку історії перевірок або порівняння версій тексту).

Використання стандартного протоколу HTTP для взаємодії між клієнтом і сервером забезпечує можливість розгортання системи як локально, так і в хмарному середовищі.

Функціональна архітектура вебдодатку організована у вигляді взаємопов'язаних компонентів, відповідно до схеми, поданої на рисунку 3.1.

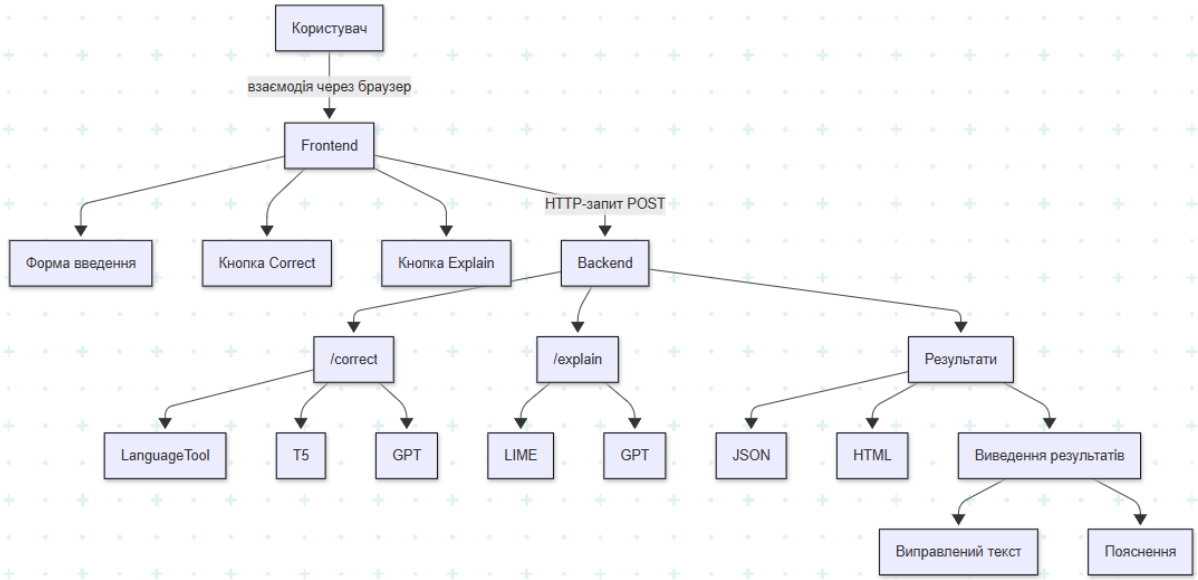

Рисунок 3.1 – Функціональна архітектура програмного забезпечення

У папці frontend/ знаходяться файли клієнтської частини додатку, яка відповідає за візуальне представлення та взаємодію користувача з додатком. Клієнтська частина складається з:

– index.html, що є головною вебсторінкою з полями вводу, кнопками та фреймом для пояснень;

– styles.css, що забезпечує стилізацію елементів інтерфейсу, адаптивний дизайн, кольори та шрифти;

– script.js, який реалізує логіку відправлення HTTP запитів до бекенду, обробку введеного тексту та відображення пояснень.

У папці backend/ розташовано серверну частину додатку, що відповідає за обробку тексту, виправлення помилок та надання пояснень користувачеві, а також тимчасові файли, що використовуються для збереження результатів роботи моделей. Дана папка складається з:

– main.py, що реалізує API за допомогою FastAPI, приймає текст та виконує всі основні функції системи, такі як корекція тексту, виправлення за допомогою технологій пояснюваного штучного інтелекту та пояснення виправлень за допомогою великої мовної моделі.

– тимчасові файли corrected_text.txt, lime_explanation.json та text_explanation.html використовуються для збереження виправленого тексту, пояснення у «сирому» вигляді, та пояснення, згенерованого природною мовою, відповідно.

Файл requirements.txt – це файл з конфігурацією залежностей, потрібних для запуску серверної частини. Він включає в себе бібліотеки, що потрібно встановити для запуску проєкту: fastapi, transformers, nltk, lime, language tool python, scikit learn, openai, requests та інші.

Завдяки обраній архітектурі стало можливим:

– ефективно інтегрувати зовнішні сервіси та моделі штучного інтелекту;

– зберігати пояснення в HTML форматі, що дозволяє без додаткової логіки виводити результат безпосередньо у вікні браузера;

– забезпечити розширюваність, наприклад, можливість в майбутньому додати інші мови, додаткові типи перевірок або моделі без потреби повністю переробляти систему.

Отже, запропонована архітектура є раціональною, адаптивною та повністю відповідає цілям проєкту.

3.2 Використані технології

3.2.1 Python

Python – це високорівнева інтерпретована мова програмування широкого профілю, розроблена Гвідо ван Россумом та вперше представлена у 1991 році [16]. Вона використовує динамічну типізацію та автоматичне керування пам'яттю, що значно полегшує процес розробки програм та робить його більш гнучким. Ця мова дозволяє програмувати з використанням кількох парадигм, зокрема об’єктно-орієнтованої, процедурної та функціональної. Серед ключових переваг Python слід виділити:

– інтуїтивний синтаксис, що сприяє пришвидшенню розробки та полегшує процес навчання;

– велика кількість бібліотек для машинного навчання, обробки тексту, вебсерверів тощо;

– висока читабельність коду, що полегшує командну роботу та обслуговування проєкту;

– кросплатформеність, тобто здатність запуску коду на різних операційних системах без необхідності змін.

У даному проєкті використовується Python версії 3.10, оскільки вона є стабільною, підтримує всі необхідні бібліотеки та сумісна з FastAPI, а також забезпечує коректну роботу нових мовних конструкцій і синтаксичних можливостей, зокрема, match case, покращене типізування та оптимізації продуктивності.

Python в цьому проєкті виступає основною серверною мовою. Саме на ній реалізовано всю логічну складову вебсерверу у файлі main.py, що відповідає за обробку запитів користувача, виправлення тексту (через модель T5), граматичний аналіз (за допомогою language-tool-python), роботу з LIME для пояснень, а також взаємодію з моделями GPT через зовнішній

API (OpenAI). Крім того, Python застосовується для зберігання тимчасових даних, роботи з файлами та побудови векторизаторів тексту для машинного навчання.

3.2.2 NLTK

NLTK (Natural Language Toolkit) – це одна з найширше вживаних бібліотек для обробки природної мови на Python. Вона містить великий арсенал інструментів і корпусів для дослідження, розробки та тестування моделей NLP. Основна мета NLTK – забезпечення дослідників, студентів і програмістів зручними інструментами для аналізу природної мови та створення прототипів алгоритмів. Серед функціоналу, що підтримується бібліотекою, є токенізація на слова та речення, визначення частин мови, пошук іменованих сутностей (NER), формування n-грам, робота з корпусами, такими як WordNet, побудова синтаксичних дерев тощо.

Всі ці засоби втілені у вигляді зрозумілих у використанні модулів і функцій. Бібліотека підтримується спільнотою розробників та регулярно оновлюється, відповідно до нових тенденцій у сфері обробки мови [17].

На рівні програмного коду NLTK реалізує аналіз тексту у вигляді послідовності перетворень, що дозволяє розробникам легко об'єднувати етапи обробки у власних пайплайнах. У більш комплексних завданнях (POS-тегінг, chunking) NLTK здатний працювати як з вбудованими моделями, так і з інтегрованими зовнішніми корпусами.

У даному проєкті бібліотека NLTK використовується як інструмент для розбиття введеного користувачем тексту на речення. Це є необхідним попереднім етапом перед передачею кожного речення до моделей корекції. Розбиття реалізоване за допомогою функції sent_tokenize() модуля punkt – токенізатор, натренований коректно визначати межі речень, враховуючи не лише крапки, а й скорочення, пунктуацію та контекст.

Серед переваг вибору NLTK для даного проєкту – простота в інтеграції та використанні, локальна обробка та придатність до прототипування, що дає змогу швидко створити основу обробки тексту.

3.2.3 Scikit-learn

Scikit-learn є однією з найбільш популярних та найбільш потужних бібліотек мови Python для побудови, тренування, тестування та інтеграції моделей машинного навчання. Вона забезпечує розробників зручним уніфікованим високорівневим інтерфейсом для використання широкого спектру алгоритмів машинного навчання, таких як моделей регресії, кластеризації, класифікації, зниження розмірності, побудови конвеєрів, оцінювання ефективності тощо. Scikit-learn було розроблено у 2007 році як відкритий проєкт з відкритим вихідним кодом, що також є частиною екосистеми SciPy – наукового стеку мови Python. Назва «Scikit», відповідно, походить від «SciPy Toolkit». Бібліотека широко використовується у наукових дослідженнях, розробці MVP та прототипів, навчанні та освіті, а також у виробничих системах – часто в якості частини більших пайплайнів.

У таблиці 3.1 наведено приклади задач та інструменти, які має бібліотека Scikit-learn для їхнього вирішення.

Таблиця 3.1 – Основні можливості scikit-learn

Категорія задач	Приклади алгоритмів
Класифікація	Logistic Regression, SVM, Decision Trees, KNN
Регресія	Linear Regression, Ridge, Lasso
Кластеризація	KMeans, DBSCAN, Agglomerative Clustering
Зниження розмірності	PCA, Truncated SVD, t-SNE
Моделювання ансамблів	Random Forest, Gradient Boosting

Продовження таблиці 3.1

Оцінювання моделей	Крос-валідація, метрики точності
Конвеєри (pipelines)	Зв’язування перетворень та моделей в один блок
Передобробка даних	StandardScaler, TfidfVectorizer, LabelEncoder

У даному проєкті бібліотека Scikit-learn використовується для створення локальної моделі класифікації, яку згодом аналізує пояснювальний інструмент LIME. Для її створення використовуються такі інструменти, як TfidfVectorizer та LogisticRegression.

TfidfVectorizer – це інструмент для перетворення текстових даних у числові вектори за методом TF-IDF (Term Frequency – Inverse Document Frequency). Цей метод оцінює важливість кожного токена в межах тексту або документа стосовно всього корпусу. Алгоритм працює наступним чином: спочатку підраховує, скільки разів кожне слово зустрічається в окремому документі, а потім знижує вагу слів, що часто трапляються в багатьох документах, наприклад, «and», «he», «is». Таким чином, увага фокусується на унікальних та значущих словах, що дозволяє ефективно перетворювати текст на числові вектори для подальшої роботи моделей. Алгоритм також може враховувати біграми, очищати стоп-слова та нормалізувати текст [18].

LogisticRegression – це алгоритм класифікації, що передбачає ймовірність належності об’єкта до певного класу та може використовуватися як для бінарної, так і для мультикласової класифікації. Алгоритм працює таким чином: спочатку обчислюється зважена сума вхідних ознак, наприклад, числових векторів, а потім до цієї суми застосовується логістична функція (сигмоїда), що перетворює результат на значення від 0 до 1, що і є ймовірністю належності об’єкта до певного класу. Навчання моделі полягає у підборі таких ваг, щоб мінімізувати похибку

передбачення. Модель є лінійною та добре інтерпретованою, тому часто використовується в задачах з поясненням рішень [18].

3.2.4 Технології клієнтської частини

Усі вебдодатки складаються з двох основних частин – бекенду, тобто функціоналу, та фронтенду, тобто інтерфейсу, який видимий користувачеві. За структуру і зовнішній вигляд вебсторінок додатку відповідають HTML та CSS.

HTML (HyperText Markup Language) – це мова розмітки гіпертексту, що описує структуру вебсторінки. В класичному сенсі вона не є мовою програмування, адже не має функцій, циклів або умов. Замість цього HTML є декларативною мовою, яка описує, що і в якому вигляді має бути на сторінці, на відміну від імперативних мов, що задають покрокові інструкції. Мова HTML визначає розміщення, ієрархію та типи контенту, який бачить користувач у браузері: заголовки, абзаци, зображення, посилання, таблиці, форми тощо [19].

HTML-файл по своїй суті є текстовим документом, який містить теги в трикутних дужках: наприклад, <h1>. Теги вказують браузеру, яким чином відображати кожен елемент. Ця мова має деревоподібну структуру DOM (Document Object Model), що визначає кожен тег як вузол у дереві.

CSS (Cascading Style Sheets) – це мова таблиці стилів, що визначає зовнішній вигляд структурних елементів HTML. CSS не змінює структуру документа, але дає змогу гнучко керувати його дизайном.

CSS-правила застосовуються до HTML-елементів за допомогою селекторів, що вказують, які саме елементи яким чином потрібно стилізувати. Наприклад, можна встановити колір, шрифт тексту, параметри графічних елементів, кордони елементів та відстані між ними тощо. CSS працює каскадно – це означає, що якщо до одного елемента застосовується

кілька правил, вони об’єднуються за певними правилами пріоритетності [20].

Для кращого розуміння взаємодії HTML і CSS їх можна уявити так: HTML є так званим «скелетом» сторінки, визначаючи, що голова знаходиться вгорі, а ноги – паралельно одна одній, тоді як CSS є «одягом», тобто візуальним оформленням, якому можна задати кольори, розміри тощо. HTML-файл підключає CSS-стилі через тег <link> у <head> або вбудовано через тег <style> чи атрибути style [20].

JavaScript – це високорівнева інтерпретована мова програмування, що разом з HTML та CSS є основною технологією, що використовується для розробки клієнтського інтерфейсу у сучасних вебдодатках. На відміну від декларативних HTML і CSS, ця імперативна мова забезпечує реалізацію логіки вебдодатку, реагування на дії користувача, маніпулювання структурою HTML та динамічну поведінку вебсторінок. Мова була створена у 1995 році у форматі простого скриптового засобу, проте з того часу перетворилась на повноцінну мову програмування, що може використовуватись не тільки в браузерах, а і на серверах, у мобільних додатках, десктопних додатках та IoT (Internet of Things) [21]. Основні функції мови JavaScript включають:

– динамічну зміну контенту на сторінці без перезавантаження (DOM manipulation);

– валідацію форм та обробку подій, наприклад, натискання кнопок, введення тексту;

– роботу з асинхронними запитами через fetch, XMLHttpRequest, async/await;

– збереження локальних даних у браузері через localStorage, sessionStorage;

– інтерактивність, наприклад, анімовані елементи, модальні вікна, сповіщення тощо;

– комунікацію з сервером у режимі реального часу через WebSockets, AJAX.

Дана мова програмування особлива своєю динамічною типізацією – тип змінної визначається під час виконання, як і в мові Python, на відміну від С-подібних мов зі статичною типізацією. JavaScript працює за подієвою моделлю, тобто реагує на дії користувача, таймери та мережеві події. Вона підтримує такі парадигми, як об’єктно-орієнтоване, функціональне та імперативне програмування, а також є єдиною мовою, яку браузери підтримують нативно на клієнтському боці [21].

У даному проєкті використано найновішу версію JavaScript, орієнтовану на ES6+ (ECMAScript 2015 і вище). Синтаксис цієї версії є стандартом сучасної розробки та дозволяє створити більш читабельний, надійний та асинхронно орієнтований код, що є надзвичайно важливим для інтерактивних додатків.

3.2.5 FastAPI та інфраструктура REST API

FastAPI – це сучасний високопродуктивний фреймворк для мови Python, створений для розробки RESTful API у 2018 році. Основною його особливістю є глибока інтеграція з типізацією Python та використання бібліотек Pydantic для валідації і Starlette для маршрутизації та роботи з запитами. FastAPI побудований на основі ASGI (Asynchronous Server Gateway Interface), що дозволяє асинхронну обробку запитів, на відміну від старіших фреймворків, побудованих на WSGI (Web Server Gateway Interface) – до них відносяться Flask та Django. Коли клієнт надсилає HTTP-запит, FastAPI примає його, автоматично перевіряє структуру запиту, виконує відповідну функцію обробки та формує відповідь у форматі JSON або HTML. Також за потреби після формування відповіді можна вивести сформовану API-документацію [22].

Pydantic – це бібліотека, що використовується для валідації та серіалізації даних. Вона дозволяє оголошувати структуру запитів у вигляді Python-класів, що мають типи даних. Ці класи створюють гарантію того, що запити від клієнтської частини містять правильні поля у коректному форматі. Якщо це не так – FastAPI повертає помилку 422.

CORS Middleware (Cross-Origin Resource Sharing) – це механізм, що дозволяє або забороняє доступ до API з інших доменів або портів. Без відповідного дозволу браузер блокує запити з джерела, вважаючи їх потенційно небезпечними.

Uvicorn – це ASGI-сервер, що забезпечує виконання FastAPI-додатка. У даному проєкті він запускається через відповідну команду «uvicorn main:app –reload» та відповідає за обробку HTTP-запитів до FastAPI. У підсумку, всі елементи інфраструктури складають єдину систему: Uvicorn запускає FastAPI, який приймає запити, перевіряє дані через Pydantic, дозволяє доступ через CORS і надсилає відповідь.

3.3 Функціонал для виправлення тексту

Виправлення тексту в додатку реалізовано поетапно за допомогою трьох основних моделей – для виправлення орфографії, граматики та семантики і лексики. На початку текст розбивається на речення, кожне з яких проходить через три кроки виправлень – такий підхід був розроблений для досягнення найвищої можливої якості та точності виправлень. Після виправлення всі речення об’єднуються в цільний текст.

Важливо зазначити, що моделі, використані для кожного етапу виправлень, можуть виправляти не лише той тип помилок, який для них призначений. Це означає, що рішення моделей можуть перетинатися – наприклад, модель для виправлення граматики виправила і орфографію також, а модель для виправлення семантики ще додатково «підправила» граматику. Тим не менш, підхід використання послідовності з кількох

різних моделей був обраний експериментальним шляхом: різні моделі було протестовано на текстах з різними типами помилок та помічено, що не всі моделі здатні однаково ефективно виправляти всі типи помилок – наприклад, модель LanguageTool майже не виправляє лексичні помилки. Даний дослід призвів до висновку, що кращим рішенням у межах проєкту буде накладка виправлень різних моделей, ніж недостатньо висока точність. Таким чином, терміни «модель для виправлення граматики/орфографії/семантики» є певною мірою умовними, та внутрішні процеси роботи моделей майже ніяк не обмежуються у коді.

Для першого етапу – виправлення граматики – було задіяно модель-трансформер T5, програмний код для якої подано у лістингу 3.1.

Лістинг 3.1 – Програмний код моделі для виправлення граматики

```
corrector = pipeline("text2text-generation",
model="vennify/t5-base-grammar-correction")
correct_grammar = corrector(
            sent,
            max_length=max_length,
            do_sample=False,
            num_beams=4,
            early_stopping=True
        )
```

Для другого етапу, яким є виправлення орфографії та пунктуації, використано систему на базі правил (rule-based) LanguageTool, код для якої надано у лістингу 3.2.

Лістинг 3.2 – Програмний код для моделі виправлення орфографії

```
def correction_languagetool(text):
    tool = language_tool_python.LanguageTool('en-US')
    matches = tool.check(text)
```

Продовження лістингу 3.2

```
    corrected_text =
language_tool_python.utils.correct(text, matches)
    return corrected_text
```

Для третього етапу – семантики, стилістики та лексики – використано GPT-асистент через API. Для асистента заздалегідь задано промпт – інструкцію виправити лише вищевказані види помилок без зміни стилю чи структури. Це дозволяє зробити речення змістовно точними й стилістично доречними. Програмний код промпту наведено у лістингу 3.3.

Лістинг 3.3 – Програмний код для виправлення семантичних помилок

```
prompt = f"""
    You are an English writing assistant. The user will
provide you with a text that contains grammatical,
spelling, punctuation, and lexical mistakes.

    Your task:
    1. Correct only lexical and semantic mistakes in the
text, if there are any. Make sure all the sentences make
sence separately and as the whole text. Pick up the style
of the text - formal, informal or something in between.
Make corrections according to that style - for example, if
the text is clearly informal, do not turn the informal
frases into formal ones.
    2. Output only the corrected version at the end. Do
not include the word \"corrected\" at the beginning.

    Text: {text}
    """
```

Функція correct_text_by_sentences збирає в одне ціле весь процес виправлень: розбиття тексту на речення, проходження кожним з речень

трьох етапів корекції та збирання цілісного тексту. Програмний код функції надано у лістингу 3.4.

Лістинг 3.4 – Програмний код функції correct_text_by_sentences

```
def correct_text_by_sentences(text, max_length=128):
    sentences = nltk.sent_tokenize(text)
    corrected_sentences = []

    for sent in sentences:
        correct_grammar = corrector(
            sent,
            max_length=max_length,
            do_sample=False,
            num_beams=4,
            early_stopping=True
        )
        correct_spelling =
correction_languagetool(correct_grammar[0]['generated_text
'])
        correct_semantic =
correction_gpt(correct_spelling)
        corrected_sentences.append(correct_semantic)
    return " ".join(corrected_sentences)
```

Коли на сервер потрапляє POST-запит за адресою /correct, викликається функція, що виправляє текст та створює пояснення для нього. У функції спочатку відбувається отримання початкового тексту, введеного користувачем, та його виправлення за допомогою функції correct_text_by_sentences, описаної вище. З двох текстів – правильного і неправильного – створюється міні-корпус для навчання проміжної моделі.

Проміжна модель – це локальна одноразова модель Logistic Regression. Вона потрібна тому, що LIME приймає на вхід моделі, що мають функцію predict_proba (передбачення ймовірностей), а в моделі

типу «чорна скринька», що є кінцевим етапом виправлення тексту, такої функції немає. Через це спочатку TF-IDF перетворює слова на числові вектори, що відображають важливість кожного слова, а потім модель Logistic Regression навчається на міні-корпусі з двох текстів (правильного і неправильного), щоб передбачити, які слова у двох текстах відрізняються. Pipeline об'єднує обидва кроки в єдину модель pipe, що в підсумку має ймовірності для кожного слова, які й потрібні для LIME.

Наступним кроком є створення пояснення через LIME – для цього використовується LimeTextExplainer та метод explain_instance для пояснення конкретного випадку. Після цього формується JSON-файл із результатом у форматі, що показаний у лістингу 3.5.

Лістинг 3.5 – Формат JSON-файлу для збереження пояснення

```
lime_explanation_json = {
        "text": start_text,
        "top_tokens": [
            {"token": word, "importance": score}
            for word, score in features
        ],
        "predicted_label":
int(pipe.predict([start_text])[0]),
        "class_names": explanation.class_names,
    }
```

Останнім кроком є збереження результатів роботи у файли: JSON-файл з поясненням, TXT-файл з виправленим текстом і HTML-файл з графічним поясненням від LIME.

3.4 Функціонал для пояснення тексту

Пояснення роботи моделі, отримані від LIME, є специфічними – не кожен кінцевий користувач без досвіду з подібними інструментами здатен їх зрозуміти. Тим більше, графічні пояснення лише виділяють слова у тексті,

що є потенційно правильними або помилковими, та показують відсоток впливу певних слів на текст, тоді як кінцевий користувач прагне також зрозуміти з точки зору правил мови, чому правильний варіант – саме такий. Для цієї цілі наступним кроком розробки стала інтерпретація пояснення від LIME за допомогою асистента GPT.

Для цього спочатку формується інструкція для асистента, що містить оригінальний та виправлений тексти, рішення моделі, список важливих слів, виявлених LIME та вказівки щодо того, як саме сформувати пояснення. В числі вказівок – пояснювати, як викладач мови та простими словами, не вигадувати неіснуючі помилки, використовувати певне форматування тощо.

В поточній версії додатку природномовне пояснення виводиться лише англійською мовою, але у перспективі однією з найголовніших цілей є додавання можливості виводу пояснення різними мовами, адже при вивченні іноземної мови важливо повністю розуміти правила, і деяким користувачам може бути корисно побачити їх пояснення рідною мовою.

При отриманні POST-запиту на /explain запускається функція explain(), що в першу чергу завантажує пояснення від LIME у форматі JSON-файлу, а також виправленого тексту у форматі файлу TXT. Після цього формується промпт для GPT-асистента за допомогою функції створення промпту, що описана вище. У лістингу 3.6 надано програмний код для формування запиту до асистента через OpenRouter.

Лістинг 3.6 – Програмний код запиту до GPT-асистента

```
headers = {
        "Authorization": f"Bearer {OPENROUTER_API_KEY}",
        "Content-Type": "application/json",
        "HTTP-Referer": "https://yourdomain.com",
        "X-Title": "lime-text-explainer"
    }

data = {
```

Продовження лістингу 3.6

```
    "model": "openai/gpt-4o-2024-11-20",
    "messages": [
            {"role": "user", "content": prompt}
    ]
}
```

Після цього запит надсилається, а отримана відповідь додається до окремого HTML-файлу, що вставлено в основну вебсторінку – такий підхід використано для зручності перегляду та прокрутки пояснення. Повний програмний код серверної частини вебдодатку наведено у лістингу А.1 додатку А.

3.5 Клієнтська частина додатку

Основою вебсторінки є структура, написана мовою HTML. У частині <head> задаються такі параметри сторінки, як мова, кодування, адаптивність для мобільних пристроїв та підключення зовнішнього файлу CSS зі стилями. Далі описуються основні структурні елементи сторінки в елементі <body>. Угорі посередині розташовано логотип, під ним – текст з короткою інструкцією дій для користувача.

Нижче розташовані основні елементи сторінки: два текстових поля для вводу і виводу, кнопка у вигляді стрілки, яку потрібно натиснути для виправлення тексту, а також кнопка «Explain», після натискання якої виводиться пояснення.

Внизу знаходиться елемент iframe, що слугує полем для відображення пояснення, яке надходить із сервера. Доки не натиснуто кнопку «Explain», поле є невидимим:

```
<iframe id="explanation-frame" style="width: 100%; height: 600px; border: none; display: none;"></iframe>
```

Повний код HTML-файлу наведений у лістингу А.2 в додатку А.

Стилі вебсторінки задані у CSS-файлі, що підключено до файлу HTML. «Тіло» сторінки <body> має світло-сірий колір фону, нульові зовнішні відступи, вирівнювання по центру та шрифт Arial.

Контейнер (.container) є головним блоком контенту, який містить усі інші елементи. Для нього встановлено максимальну ширину і вирівнювання по центру, а також використовується flexbox для вертикального розміщення дочірніх елементів.

Текстові поля для вводу та виводу тексту – це два однакових поля шириною 40%, висотою 300 пікселів і з заблокованою можливістю зміни розміру. Для них також прописано округлення кутів та світлий фон, що не є повністю білим, але є світлішим за колір фону сторінки.

Кнопки для виправлення тексту та його пояснення є великими, округлими та мають різні кольори: кнопка стрілки, що активує виправлення, є світло-блакитною, тоді як кнопка для виводу пояснення – світло-зеленою. Разом це забезпечує гармонійну колірну гаму вебсторінки, яка не «ріже» погляд користувача. Повний програмний код CSS-файлу наведено у лістингу А.3 додатку А.

У даному проєкті взаємодію клієнтської та серверної сторін реалізовано за допомогою JavaScript. Так, як у користувача є дві основні наявні дії – виправлення та пояснення, JavaScript-файл містить дві відповідні асинхронні функції.

Функція correctText() надсилає текст, введений користувачем, на виправлення: вона запускається при натисканні кнопки-стрілки на вебсторінці. Текст зчитується з поля для вводу, при тому, якщо текст порожній – функція зупиняє роботу та показує відповідне повідомлення (рисунок Б.1 додатку Б). Якщо текст зчитався коректно, у полі виводу показується візуальний індикатор обробки – слово «Correcting» (рисунок Б.2 додатку Б). Тим часом, до бекенду надсилається запит /correct, а в тілі запиту передається текст, введений користувачем, у форматі JSON. Відповідь отримується також у JSON, а

якщо виправлень не було здійснено – виводиться повідомлення «Done, but no corrections returned». Якщо під час обробки тексту відбулася помилка на стороні серверу, у полі виводу показується відповідний текст помилки (рисунок Б.3 додатку Б). Якщо текст обробився коректно, він виводиться у полі праворуч.

Функція explainCorrections() викликається при натисканні кнопки «Explain» та отримує пояснення. Спочатку елемент iframe, що використовується як поле для виводу пояснення, очищається, щоб прибрати попереднє пояснення, після чого робиться запит до /explain. Сервер повертає відповідь, після якої створюється тимчасовий об’єкт-URL із HTML-тексту. Він вставляється в iframe, який стає видимим. Якщо на якомусь з етапів відбулась помилка, користувач бачить повідомлення про неї угорі вікна браузера (рисунок Б.4 додатку Б). Якщо пояснення було оброблене правильно, користувач бачить його внизу під кнопкою «Explain» з можливістю прокрутки. Повний код JS-файлу наведено у лістингу А.4 додатку А.

4 ВЕБДОДАТОК ДЛЯ ВИПРАВЛЕННЯ І ПОЯСНЕННЯ ПОМИЛОК В АНГЛОМОВНИХ ТЕКСТАХ

4.1 Виявлення орфографічних та пунктуаційних помилок

Орфографічні, або друкарські, помилки, є найпростішим типом помилок для виправлення – разом із пунктуаційними. Наприклад, текст «Ths is an exmple of a txt with severl speling erors» містить пропуски букв та переплутані букви, а також не має точки в кінці речення. На рисунку 4.1 показано виправлення, які зробила модель, на рисунках Б.5 та Б.6 додатку Б – пояснення, що зробив LIME та мовна модель.

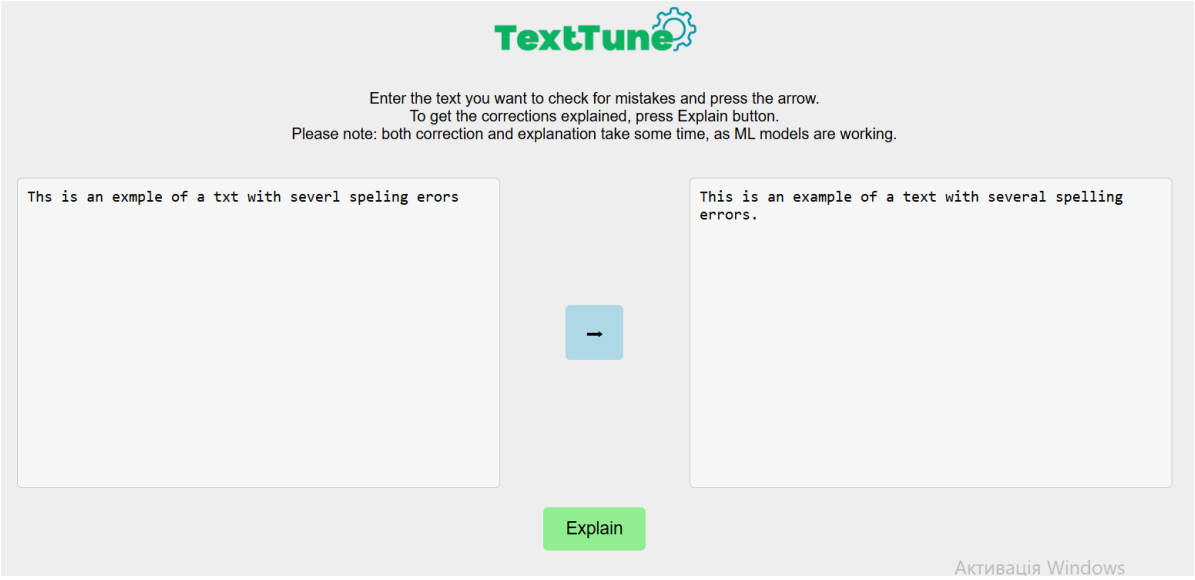

Рисунок 4.1 – Виправлення орфографічних та пунктуаційних помилок

4.2 Виявлення граматичних помилок

Граматичні помилки, ймовірно, є найпоширенішими серед тих, хто вивчає іноземну мову. Вони включають неправильне використання часів, форм дієслів, артиклів тощо.

На рисунку 4.2 показано виправлення, які зробила модель, на рисунках Б.7 та Б.8 додатку Б – пояснення, що зробив LIME та мовна модель.

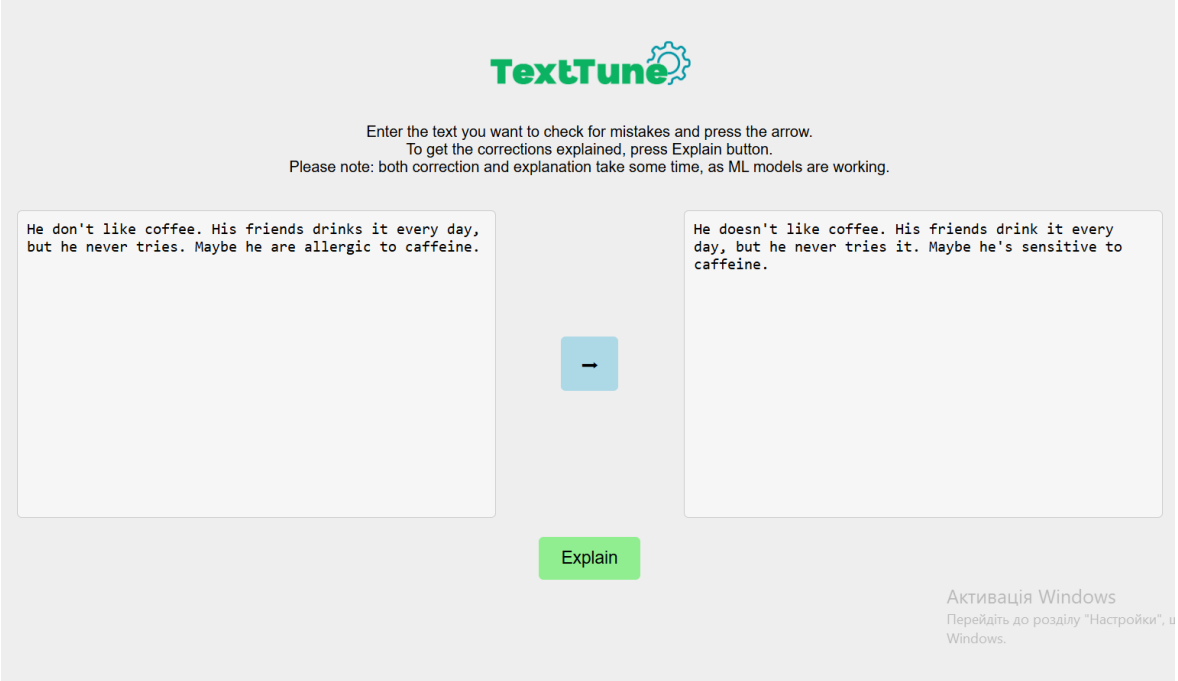

Рисунок 4.2 – Виправлення граматичних помилок

4.3 Виявлення стилістичних помилок

Стилістичними помилками рахуються такі випадки, коли, попри граматичну та орфографічну правильність, слово не є доречним у певному контексті за стилем – наприклад, використання неформальної лексики або сленгу в академічному тексті.

Стилістичні помилки погіршують сприйняття тексту, створюють враження непрофесійності або недбалості та можуть негативно вплинути на імідж автора. Виправлення таких помилок потребує не лише базових знань мови, а й розуміння норм жанру, цільової аудиторії та контексту використання тексту.

Приклад такого виправлення показано на рисунку 4.3, а на рисунках Б.9 та Б.10 – пояснення, що зробив LIME та мовна модель.

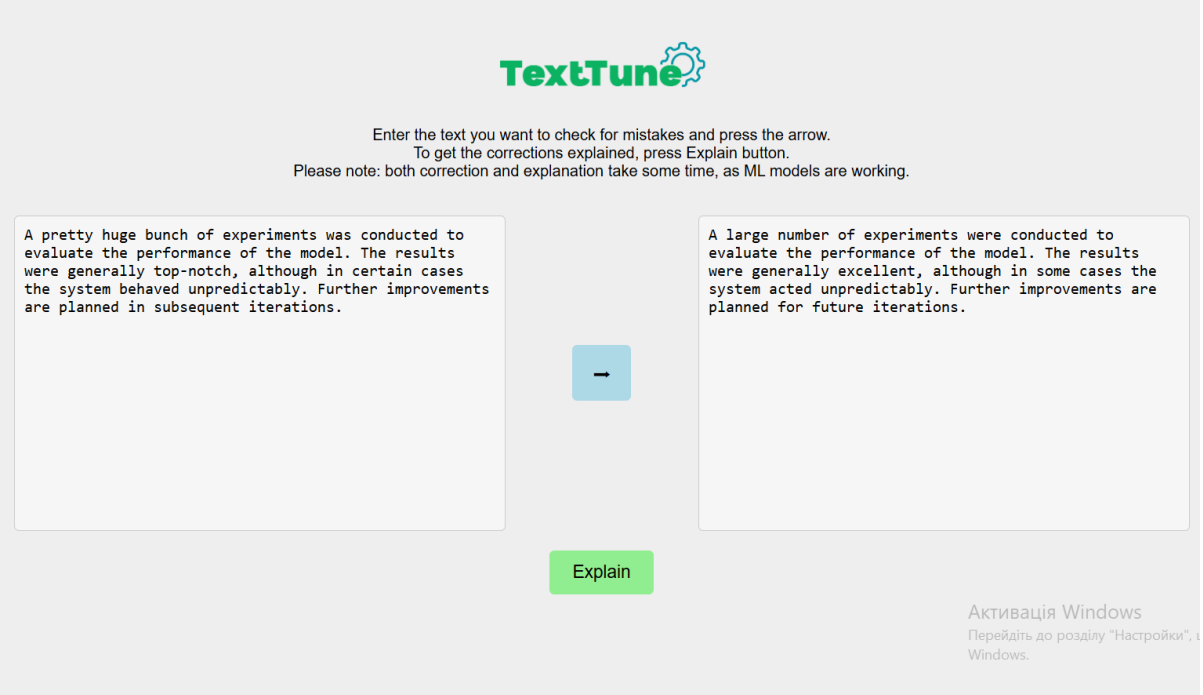

Рисунок 4.3 – Виправлення стилістичних помилок

4.4 Виявлення лексичних помилок

Лексичні помилки відносяться до складних випадків, коли слово є граматично правильним, але вживається в невластивому йому значенні. Також до цього виду помилок належать тавтологія, багатослів'я, «калька» з іншої мови, дублювання значення в двох словах тощо. Подібні помилки можуть не порушувати граматичних норм, однак суттєво знижують якість тексту, ускладнюють його сприйняття та можуть змінювати передбачений зміст або викликати двозначність. Виявлення лексичних неточностей вимагає глибокого аналізу контексту та знання мовних норм. Особливо складними є випадки, коли необхідно розрізнити близькі за звучанням, але різні за значенням слова, як-от «affect» та «effect», або уникнути запозичень, які не відповідають нормам літературної мови.

Приклад виправлення лексичної помилки показано на рисунку 4.4, а пояснення – на рисунках Б.11 та Б.12 додатку Б.

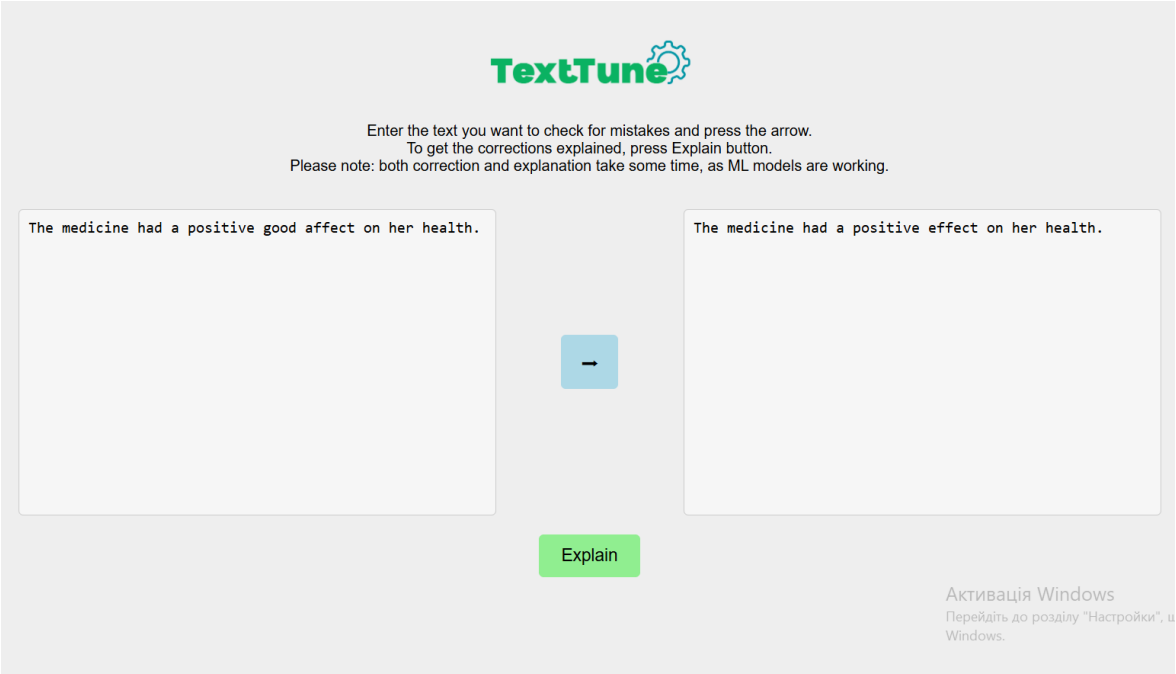

Рисунок 4.4 – виправлення лексичних помилок

4.5 Виявлення семантичних помилок

Семантичні помилки – це помилки невідповідності певних слів або виразів контексту. Вони виникають, коли окремі елементи тексту не відповідають загальному змісту або логіці висловлювання, що призводить до порушення смислової цілісності. Пошук та виправлення таких помилок, ймовірно, є найменш розвиненою гілкою у виправленні помилок системами штучного інтелекту через складність глибинного аналізу контексту та необхідність розуміння прихованих зв'язків між словами.

Автоматичні системи повинні враховувати не тільки граматичні конструкції, а й змістовні відношення між частинами тексту, що вимагає використання потужних нейронних моделей із глибоким навчанням, таких як трансформери.

Приклад виправлення такої помилки показано на рисунку 4.5, а пояснення – на рисунках Б.13 та Б.14 додатку Б.

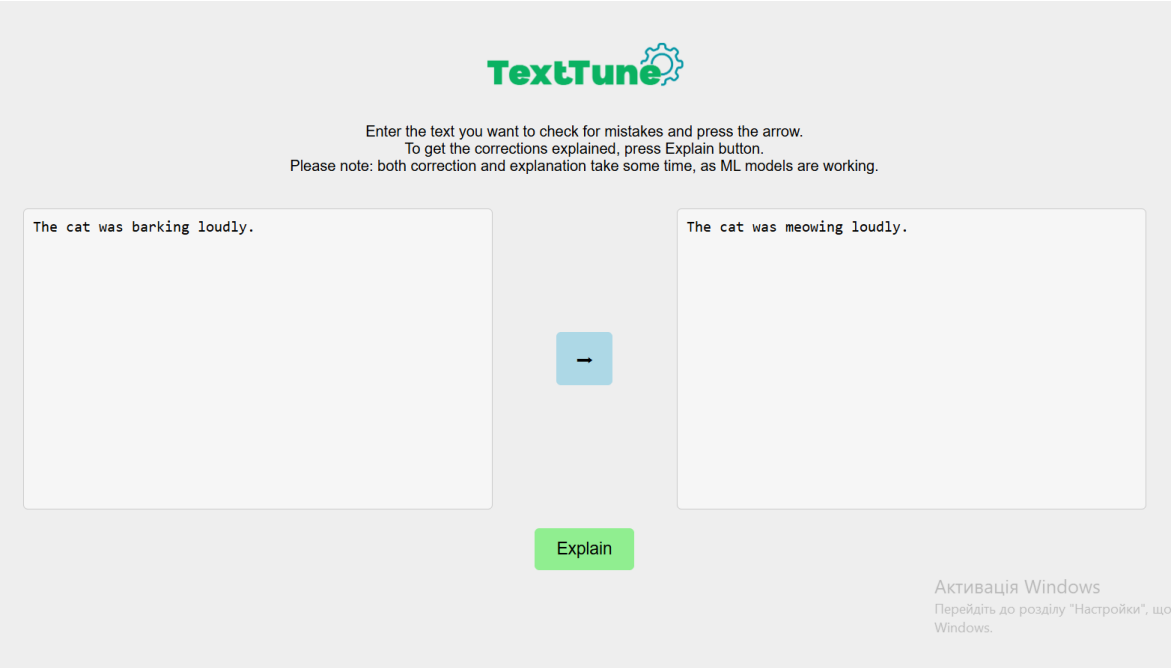

Рисунок 4.5 – Виправлення семантичних помилок

4.6 Робота з текстами з комбінованими типами помилок

Більшість текстів з помилками містять не один конкретний тип помилок, а комбінують у собі різні типи – наприклад, граматично неправильно використаний час та пропущена буква або використання неправильної частини мови.

Саме для таких випадків у розробленій системі передбачено ланцюжок з кількох моделей для виправлення – це потрібно тому, що внаслідок дослідження моделей та методів штучного інтелекту було виявлено, що різні застосовані моделі найкраще працюють на різних типах помилок. Формат «ланцюжка» з моделей забезпечує максимальну увагу та покриття усіх типів помилок та їхніх комбінацій у текстах.

Приклад виправлення тексту, що комбінує у собі різні помилки, наведено на рисунку 4.6, а їх пояснення – на рисунках Б.15 та Б.16 додатку Б.

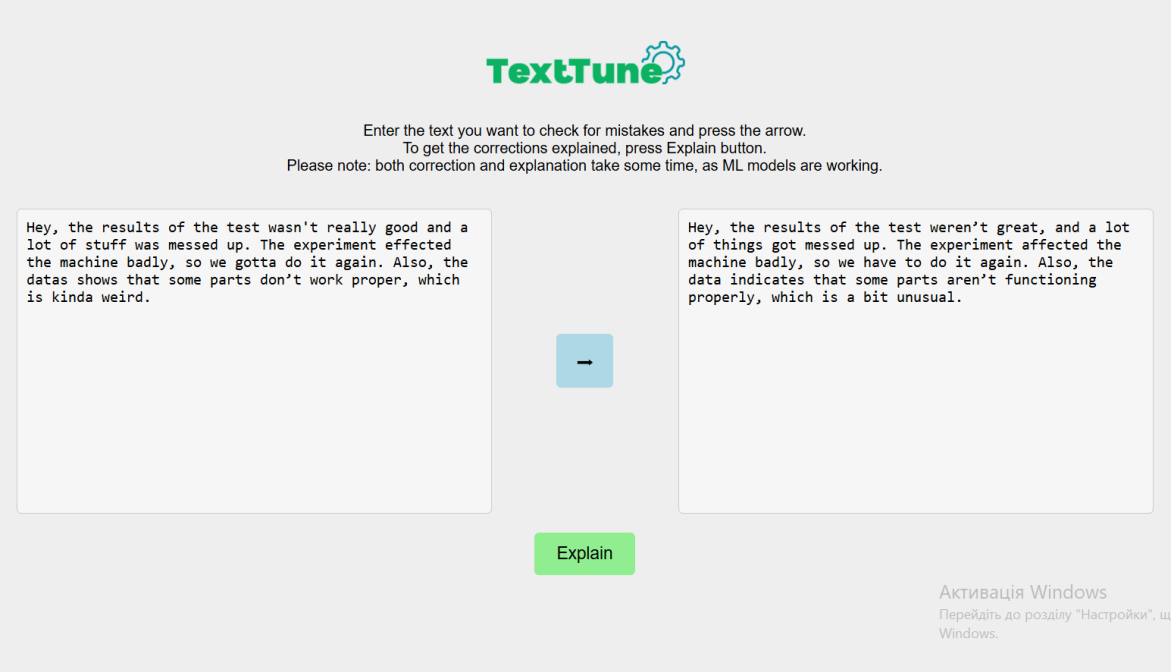

Рисунок 4.6 – Виправлення тексту з кількома типами помилок

Таким чином, здатність розробленої системи працювати з текстами, які містять кілька типів помилок одночасно, свідчить про її універсальність та адаптивність. Завдяки каскадному підходу з використанням різних моделей, кожна з яких переважно спеціалізується на певному класі помилок, досягається висока точність виправлень навіть у складних випадках. Такий підхід є ключовим для якісної роботи з реальними текстами, які рідко містять лише одну окрему помилку, і дозволяє використовувати систему в більш широкому спектрі задач – від навчальних до професійно-редакторських.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проведено всебічний аналіз предметної галузі, що охоплює сучасні уявлення про лінгвістику, обробку природної мови, сучасні інструменти автоматичної корекції текстів та підходи до їх пояснення. Ця частина дослідження заклала теоретичне підґрунтя для подальшої реалізації вебдодатку, який дозволяє не лише виправляти англомовні тексти, а й супроводжувати ці виправлення зрозумілими для користувача поясненнями.

Важливою частиною дослідження стало проведення порівняльного аналізу існуючих рішень. Було розглянуто функціональні особливості таких систем, як Grammarly, LanguageTool, Microsoft Editor, Google Docs, DeepL Write, QuillBot і Hemingway Editor. Було виявлено, що усі вони в тій чи іншій мірі виконують автоматичне редагування, однак більшість з них або зовсім не пояснюють суть змін, або подають шаблонні пояснення, не пов’язані з конкретним контекстом. Це підтвердило гіпотезу про те, що пояснювальна складова – критично важливий, але часто ігнорований компонент сучасних мовних сервісів.

У межах практичної частини роботи було реалізовано повноцінний вебдодаток для автоматичного виправлення англомовних текстів із можливістю генерації пояснень до внесених змін. Архітектура додатку побудована за принципом поділу на клієнтську та серверну частини. Фронтенд реалізовано засобами HTML, CSS та JavaScript, що забезпечило простий та інтуїтивно зрозумілий інтерфейс. Серверну частину створено на основі фреймворку FastAPI з використанням сучасних мовних моделей, а також статистичних та основаних на правилах інструментів обробки тексту. Такий підхід дозволив реалізувати поетапну корекцію та забезпечити пояснюваність результату за допомогою пояснюваного штучного інтелекту та великих мовних моделей. Усі функціональні компоненти було протестовано й інтегровано в єдину систему, що дозволяє користувачу в

інтерактивному режимі не лише отримувати виправлений текст, а й розуміти причини внесених змін. Реалізований функціонал підтвердив практичну доцільність застосування гібридного підходу у задачах автоматичного редагування природної мови.

Проведена дослідницька робота дала змогу поглибити розуміння принципів роботи систем обробки природної мови, засвоїти поняття інтерпретованого ШІ, розширити знання про можливості сучасних вебтехнологій, а також власноруч розробити та протестувати новітній підхід до пояснення помилок у текстах шляхом пояснення роботи моделі, що їх виправляє.

Отже, поставлені завдання було виконано в повному обсязі, що дозволило зробити однозначні висновки про технічну реалізованість поставленої мети, обґрунтувати новизну і значущість обраного підходу та підкреслити навчальну цінність дослідження у межах освітньої програми університету.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Філіппова Н. М. Вступ до прикладної лінгвістики : Навчальний посібник. Миколаїв, 2014. 152 с. URL: https://nuos.edu.ua/wp-content/uploads/2024/11/Filippova-41.pdf (дата звернення: 09.04.2025).

2. IBM. What Is NLP (Natural Language Processing)? | *IBM*. *IBM - United States*. URL: https://www.ibm.com/think/topics/natural-language-processing (дата звернення: 11.04.2025).

3. Natural Language Processing. *NNLM*. URL: https://www.nnlm.gov/guides/data-glossary/natural-language-processing (дата звернення: 14.04.2025).

4. Can ChatGPT Challenge the Scientific Impact of Published Research, Particularly in the Context of Industry 4.0 and Smart Manufacturing? / M. Golovianko та ін. 2024. URL: https://doi.org/10.1016/j.procs.2024.02.072 (дата звернення: 15.04.2025).

5. How We Use AI to Enhance Your Writing | Grammarly Spotlight. *Grammarly Spotlight: How We Use AI to Enhance Your Writing*. URL: https://www.grammarly.com/blog/product/how-grammarly-uses-ai (дата звернення: 24.04.2025).

6. GitHub - languagetool-org/languagetool: Style and Grammar Checker for 25+ Languages. *GitHub*. URL: https://github.com/languagetool-org/languagetool (дата звернення: 25.04.2025).

7. Achieving Zero-COGS with Microsoft Editor Neural Grammar Checker. *Microsoft Research*. URL: https://www.microsoft.com/en-us/research/blog/achieving-zero-cogs-with-microsoft-editor-neural-grammar-checker/ (дата звернення: 28.04.2025).

8. Kumar S., Tong S., Hoskere J. Using neural machine translation to correct grammatical faux pas in Google Docs | Google Workspace Blog. Google Workspace Blog. URL: https://workspace.google.com/blog/ai-and-machine-

learning/using-neural-machine-translation-to-correct-grammatical-in-google-docs (дата звернення: 28.04.2025).

9. DeepL Write: AI-powered writing companion. *DeepL Translate: The world's most accurate translator*. URL: https://www.deepl.com/write (дата звернення: 29.04.2025).

10. QuillBot. *QuillBot*. URL: https://quillbot.com/ (дата звернення: 30.04.2025).

11. Hemingway Editor. *Hemingway Editor*. URL: https://hemingwayapp.com/ (дата звернення: 30.04.2025).

12. LanguageTool. *LanguageTool*. URL: https://languagetool.org/?force_language=1 (дата звернення: 20.05.2025).

13. Attention Is All You Need / A. Vaswani та ін. м. Long Beach. 2017. URL: https://arxiv.org/pdf/1706.03762 (дата звернення: 20.05.2025).

14. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer / C. Raffel та ін. *Journal of Machine Learning Research*. 2020. № 21. URL: https://arxiv.org/pdf/1910.10683 (дата звернення: 20.05.2025).

15. Molnar C. Interpretable Machine Learning. lulu.com, 2020. 318 с. URL: https://christophm.github.io/interpretable-ml-book/ (дата звернення: 21.05.2025).

16. Welcome to Python.org. *Python.org*. URL: https://www.python.org/ (дата звернення: 19.05.2025).

17. Klein E., Bird S., Loper E. Natural Language Processing with Python. O'Reilly Media, Incorporated, 2017. URL: https://www.nltk.org/book/ (дата звернення: 19.05.2025).

18. User Guide. *scikit-learn*. URL: https://scikit-learn.org/stable/user_guide.html (дата звернення: 20.05.2025).

19. W3Schools.com. *W3Schools Online Web Tutorials*. URL: https://www.w3schools.com/html/ (дата звернення: 22.05.2025).

20. W3Schools.com. *W3Schools Online Web Tutorials*. URL: https://www.w3schools.com/css/ (дата звернення: 22.05.2025).

21. JavaScript | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 23.05.2025).

22. FastAPI. *FastAPI*. URL: https://fastapi.tiangolo.com/ (дата звернення: 24.05.2025).